

---

## **Предисловие - История создания языка Clarion**

Брюс Д. Баррингтон, главный инженер TopSpeed Corporation:

Как это часто бывает, я просто пытался доставить себе удовольствие. Я купил свой первый персональный компьютер, смотрел на него и хотел писать программы для него. Это моя работа. Паскаль имел тогда слишком ограниченные возможности, а С на этой платформе еще не было. Поэтому я попробовал BASIC. Все что мне было нужно - это несколько интеллектуальных программ, работающих с экраном и клавиатурой. Так? Возможно, иногда, индексно-последовательный доступ. Так?

Неправильно! Я мог все это сделать. Но я не мог сделать это просто и элегантно. К тому времени я уже лет 10 работал с инструментальными средствами программирования собственной разработки. Они мне нравились. Тут мне пришло в голову, что может быть всем действительно нужен еще один язык программирования - универсальный, ориентированный на административные и экономические задачи. Разработанный специально для персональных компьютеров.

Может показаться противоречивым сочетание “универсальный язык, ориентированный на экономические задачи”, но в мире ПК много “языков”, предназначенных для решения экономических задач, но которые трудно назвать универсальными. Допускаю, что пользоваться макросами электронной таблицы - это программирование, но вряд ли можно назвать язык макроопределений универсальным языком программирования. В силу этого большинство языков баз данных не являются универсальными языками. Это скорее набор инструкций, предназначенных для исполнения программой управления базами данных. Даже язык СУБД dBase, который можно компилировать и выполнять отдельно от самой СУБД, не может, честно говоря, считаться универсальным языком программирования.

По моему определению универсальный язык программирования должен управлять всей совокупностью возможностей, заложенных в компьютер, на котором он установлен. Это значит, что программа должна быть в состоянии прочесть любой раздел любого файла, доступного операционной системе. Он должен поддерживать все разнообразие элементов взаимодействия с пользователем. Он должен стандартным образом сопрягаться с другими языками программирования и программными системами. Универсальный язык программирования не должен засорять программу собственным “интуитивно понятным интерфейсом”. Он не должен ставить преград, которые приходится преодолевать. Наоборот, он гарантирует программисту широкий диапазон возможностей и подходов для решения в любом стиле большого круга задач.

Но почему ограничивать новый язык только персональными компьютерами? В других широко известных языках преносимость на другие платформы тщательно продумана. Я решил, что персональные компьютеры заслуживают особого обращения. Даже в 1984 г., когда эта работа начиналась, ПК уже составляли существенную долю всех установленных

в мире компьютеров. Но ПК и отличались от других типов ЭВМ тем, что они были задуманы, как устройства для одного пользователя, в состав которых входили клавиатура и монитор. Доступ к клавиатуре и монитору можно было получить непосредственно, без модемов и линий связи. Для этих устройств нужны были интерактивные прикладные программы, быстро реагирующие на ввод данных. Я решил поддержать эти особенности ПК, включив в язык программирования отображение видеопамати. Если программа на Clarion'е будет работать “лишь” на 40 или 50 милл. компьютеров, я буду удовлетворен.

Моим стимулом в работе было убеждение, что программирование должно быть проще, что языки программирования должны облегчать чтение и написание программ и что низкая производительность, связанная с процессом разработки программ, коренится в неадекватных и слабо спроектированных инструментальных средствах программирования.

Эти убеждения начались с “вечных вопросов”: Зачем каждый раз оформлять оператор IF в виде IF...THEN BEGIN; “операторы”; END ELSE...(Pascal)? Какой особый смысл ключевых слов THEN, BEGIN, END в этой структуре? Зачем пользоваться знаком “:=” вместо “=” в операторе присваивания (Pascal, Modula-2, ADA)? Разве не знал разработчик языка, что оператор присваивания - это самый распространенный оператор в программе или что знак “:=” трудно набирать на клавиатуре, так как она состоит из комбинации клавиш, нажимаемых с клавишей Shift и без нее? Ну а предложение READ...AT END (COBOL), которое присваивает значение переменной, которая проверяется для прекращения цикла чтения по достижению конца файла? Почему нельзя просто проверять в цикле конец файла? Почему, объявив переменную, нужно напоминать компилятору, чтобы он преобразовал ее в выражении с переменными различных типов. Компилятор не может помнить об этом сам? Вы когда-нибудь выполняли операцию lint collection? А не спрашивали себя, почему? Ну а шестнадцатиричные дампы? Вспомните ШЕСТНАДЦАТИРИЧНЫЕ ДАМПЫ! Двадцать лет программирования породили во мне такое же чувство, как у главного героя фильма “Сеть”, который, высунувшись из окна, закричал: “Я с ума сойду. Я не могу больше этого выносить!”

## **Выбор стиля**

---

Итак я приступил к разработке нового языка программирования, который был бы компактным ( простым для написания) и выразительным (простым для чтения). Я продвигался в направлении, противоположном общепринятому: сначала я написал много программ, экспериментируя с синтаксисом и семантикой, пока они не стали на что-то похожи. Потом я работал над небольшим справочником по языку. Когда он был закончен, бригада программистов начала писать компилятор. Язык менялся ежедневно. Недавно я перечитал старые записки разработчиков и вспомнил, сколько раз и сколькими людьми был обдуман каждый элемент языка. Одни идеи предлагались и отвергались по эстетическим соображениям, другие - в силу их слабой технологичности, были и просто безумные мысли. Уцелели, по Дарвину, только сильные.

С точки зрения стиля я разделяю языки программирования на три группы: ориентированные на лексемы, на предложения и на операторы. Языки, ориентированные на лексемы, типа Pascal и C, компактны, но не очень выразительны. В этих языках программа - набор лексем (ключевых слов, имен данных, констант, знаков пунктуации и т.д.), разделенных “пропусками” ( пробелами, символами CR/LF ( возврат каретки и новая строка), комментариями и иногда запятыми). Компилятор собирает лексемы и игнорирует пропуски. Языки лексемного типа - одномерные, поэтому программисты пользуются пропусками, чтобы добавить своей программе второе измерение:

```
typedef struct {
    unsigned   char Type;           /* тип структуры */
    unsigned   Vlen;               /* длина переменной*/
    unsigned   char Dplac;         /* количество десятичных знаков, если тип decimal*/
    void       *Use;              /* указатель на переменную*/
}USEdef
```

В этом фрагменте на языке C программист сделал практически все возможное, чтобы определение типа получилось читаемым. Но левая фигурная скобка некрасиво “болтается” справа от ключевого слова **struct**, а правая - слева от **USEdef**, что тоже не очень выразительно. В любом случае, фигурные скобки с художественной точки зрения, - это не очень подходящие вертикальные разделители.

Языки, ориентированные на предложения, типа COBOL'a и языков баз данных, выразительны, но не очень компактны. Иногда операторы этих языков читаются в точности как английские предложения — например, следующий оператор из Кобола:

```
MULTIPLY PRINCIPLE BY RATE GIVING PAYMENT ROUNDED.
(Умножить PRINCIPLE на RATE, результат PAYMENT округлить)
```

Об этой записи такого не скажешь:

Payment = Principle \* Rate

Но я смею утверждать, что для программы в целом второе предложение воспринимать легче, чем первое, поскольку уже несколько подобных словесных операторов вместе образуют несколько абзацев мало понятного текста, так как другие форматы операторов еще хуже соответствуют синтаксису естественного английского языка. Вот пример из руководства по языку xBase:

```
EDIT [FIELDS <field list>] [<scope>][FDR <expL1>]
[WHILE <expL2>][FREEZE <field>]
[KEY<expr1> [,<expr2>]] [LAST] [LEDIT] [REDIT]
[LPARTITION] [NOAPPEND] [NOCLEAR] [NODELETE]
[NOEDIT | NOMODIFY] [NOLINK] [NOMENU] [NOOPTIMIZE]
[NORMAL][NOWAIT][PARTITION <expN1>][PREFERENCE <expC1>]
[SAVE][TIMEOUT <expN2>] [TITLE <expC2>]
[VALID [:F] <expL3> [ERROR <expC3>]] [WHEN <expL4>]
[WIDTH <expN3>] [[WINDOW <window name1>]
[IN [WINDOW] <window name2> | IN SCREEN]]
[COLOR SCHEME <expN4>] | COLOR <color pair list>]
```

Да, это несомненно английские слова, но что они означают? Может ли программист без справочного руководства понять формат любого из этих операторов? Кроме ответов на другие вопросы хотелось бы знать: Кто придумал поставить предложения WHILE и WHEN вместе в одном операторе? Хочется высунуться из окна и выть!

В своих экспериментальных программах я ориентировался на операторы - старомодный стиль FORTRAN'а и BASIC'а. В языках, ориентированных на операторы, программы существует в виде ASCII-файлов с исходным текстом - каждая строка программы соответствует записи файла. Поэтому вместо знаков препинания можно воспользоваться границами записей. Я остановился на формате оператора, который оказался компактным, выразительным и универсальным:

метка            ОПЕРАТОР[(параметры)][,АТРИБУТЫ[(параметры)]...

Атрибуты нужны только для объявления данных. Исполняемые операторы используют формат стандартного вызова процедур. И, конечно, было определено несколько форматов операторов присваивания ( $A=B$ ) и операторов управления (**IF**, **CASE**, и т.д.).

Метка оператора начинается с первой колонки (первая позиция записи). Оператор без метки не должен занимать первой колонки. Оператор заканчивается символом конца строки, если не продолжается на следующую строку с помощью символа “|” (вертикальная черта). Кроме того, для разделения нескольких операторов на одной строке можно использовать “;” (точка с запятой). Игнорируя пустые операторы, как в языке Modula-2, я устранил различия между разделителями операторов и признаками конца операторов.

Такое построение предложений языка позволяет избавиться от знаков пунктуации, необходимых в противном случае для идентификации меток и отдельных операторов. Блок операторов начинается со сложного оператора типа **IF** и заканчивается разделителем типа **ELSE** (который сам начинает блок операторов) или оператором **END** (или точкой). “Болтающихся” элементов, как в примере выше, совсем нет.

## Объявление данных

---

Одной из важных составляющих структуры COBOL'а, способствующей его “самодокументируемости”, является раздел данных. Любой элемент, обрабатываемый программой на COBOL'е, объявляется в разделе данных: переменные, константы, файлы, записи, индексы и даже последовательности сортировки и форматы выходных форм - отчетов. Я понимал, что объявления данных нужны для улучшения документированности программ для административных и экономических применений, но при этом чувствовал, что формат разработанных нами операторов существенно улучшит читаемость текста программ.

В конце 60х гг. фирма IBM активно пропагандировала PL/1 в качестве преемника COBOL'а. Новый язык разочаровал многих, но тем не менее привнес ряд новых идей. Сократив ключевые слова типов данных и введя вложенные комментарии (/

\*комментарий\*/), PL/1 обеспечил достаточно места для записи комментариев в операторы описания данных. В COBOL'e же предусматривались длинные, описательные имена данных. Но программисты не пользовались длинными именами. И для этого были основательные причины. Во-первых, для улучшения читаемости программы ее текст хорошо расположить “в столбик”, а организация по этому принципу раздела данных произвольно ограничивает максимальную длину имен. Во-вторых, программисты не любят пользоваться длинными именами в разделе процедур. Длинные имена порождают громоздкие выражения и увеличивают трудности автора, вызванные и без того многословным языком. Поэтому программисты на COBOL'e пользовались короткими, таинственными идентификаторами и писали программы, которые трудно было назвать самодокументируемыми, каковыми им следовало бы быть.

Программисты на PL/1 вышли из положения, снабдив операторы описания данных комментариями. Если возникает вопрос о значении имени данных, можно обратиться к их объявлению. В 60-х гг я руководил большим проектом на языке PL/1 и убедился, что операторы описания данных должны состоять из трех частей: метки оператора, типа данных и комментария.

Новый формат оператора оказался замечательным. Метка располагалась слева и была хорошо заметна. Ключевые слова типов данных были кратки ( BYTE, REAL, DIM и т.д.), что давало возможность максимум места использовать под комментарии. И наконец, сэкономило место и то, что в качестве символа, обозначающего начало комментария, использовался единственный знак “!” (восклицательный знак).

В COBOL'e и PL/1 для объявления структур данных используются “уровни”. У каждой переменной есть свой номер уровня. Если переменная не является частью структуры данных, ей присваивается уровень номер “01” или “77”. Мне никогда не нравилось пользоваться “уровнями” и я был неприятно удивлен, когда узнал, что они были перенесены в PL/1. Мне это показалось надуманной и пустой тратой места (что означают цифры “77” и зачем неструктурированным переменным вообще уровень?). Я воспользовался словом GROUP (от выражения “group item”, используемого в COBOL'e) для обозначения составных операторов, стоящих в начале структуры данных (мы их тогда называли “группами”). Этот способ аналогичен записи record ...end, принятой в Pascal'e, Modula-2 и АДЕ, в С используется запись struct {...}. Размещение вложенных операторов GROUP со сдвигом позволяет получить вполне читаемое объявление данных:

Error	GROUP,PRE(Err)	! Ошибочные данные
Date	DATE	! Дата ошибки
Time	TIME	! Время ошибки
Device	STRING(12)	! Активное устройство
Message	GROUP	! Сообщение об ошибке
MsgCode	STRING(@P###P)	! Код сообщения
	STRING(' - ')	
MsgText	STRING(32)	! Текст сообщения
	End	
	End	

Языки COBOL и PL/1 дают возможность использовать одни и те же имена данных в различных структурах. На такие данные можно ссылаться по имени, уточненному именем структуры. Это полезное свойство, так как одни и те же поля часто появляются в нескольких структурах данных (например, ACCT-NO в OLD-VENDOR, ACCT-NO в CURRENT-PAYEE и т.д.). Но многие программисты отказываются от этой практики, поскольку она порождает длинные ссылки. Вместо этого они придумывают мнемонические префиксы для каждого поля (напр., VND-ACCT-NO). Для кодирования этого нужно время, кроме того, уменьшается место в памяти, отводимое для имени.

Чтобы выйти из положения я ввел необязательный атрибут - префикс, который можно использовать с любой структурой данных ( напр., PRE(Vnd)). Элементы структуры уточняются с помощью префикса и двоеточия перед именем данных.

Для присвоения значения совпадающих элементов группы добавился оператор “группового” присваивания, соответствующий оператору “MOVE CORRESPONDING” в COBOLе и присвоению “BY NAME” в PL/1.

ГруппаНазначение :=: ГруппаИсточник

В качестве языка административных и экономических применений Clarion нуждался в развитом наборе базовых типов данных: были включены все имеющиеся длины целых и действительных чисел, чтобы обеспечить совместимость с форматом внешних записей и списком параметров. Упакованные десятичные числа были включены, чтобы решить проблемы округления величин и уменьшить потребности в памяти (они могут быть объявлены в зависимости от диапазона значений). Были также включены различные форматы строк (фиксированный формат, форматы Pascal'я и C), а также полный набор строковых функций. И наконец, были разработаны типы данных для обозначения дат и времени, чтобы иметь возможность производить арифметические действия над этими переменными ( напр., Tomorrow = Today+1 ( Завтра=Сегодня+1 )).

Но оставался без ответа еще один важный вопрос: нужны ли языку Clarion типы, определенные пользователем? Мне казалось, что нет. Какие еще типы данных требовались? В языках с объявлением типов, напр., - Pascal, Modula-2, ADA, C, - группы и массивы объявляются как типы: сначала объявляется тип, а потом экземпляр этого типа, задающий переменную. Но в программах экономического характера большинство групп и массивов объявляются только один раз. Обдумывание имени типа и кодирование оператора TYPE - это обычно ненужная работа. Я вообще никогда не считал группу или массив типом данных. Группы и массивы описывают не типы данных, а то, как эти данные располагаются в памяти.

Поэтому я сделал объявление типов необязательным. В Clarion оператор объявления с атрибутом TYPE объявляет тип данных, который можно использовать для повторяющихся структур и структур, передаваемых в качестве параметров. Оператор объявления без атрибута TYPE объявляет и тип данных и имя переменной. Для объявления переменных ранее объявленного типа я принял оператор PL/1 **LIKE**. Я чувствовал, что такая конструкция

представляет собой объединение лучшего из обоих подходов.

```
Totals      GROUP,PRE(QTR)
GrossPay    DECIMAL(12.2)
Deduction   DECIMAL(12.2)
NetPay      DECIMAL(12.2)
            END
YTD:Totals  LIKE (Totals), PRE (YTD)
```

## Типы данных без проблем

---

Язык программирования называется строго типизированным, если каждый элемент данных имеет свой единственный тип, и синтаксис языка запрещает рассматривать этот элемент как другой тип данных. Многие эксперты считают, что строгое типизирование повышает надежность программ. Может быть. Но писать программы становится труднее, поскольку ограничивается сфера действия универсальных процедур и требуется дополнительный контроль за типом данных. Более того, я никогда не слышал, чтобы программист на COBOL'e жаловался, что REDEFINES (используется для присваивания разных типов данных одной и той же области памяти) понижает надежность программ. (А программисты на COBOL'e критикуют свой язык программирования не так уж редко. Оператор ALTER вышел из употребления много лет назад, потому что с ним программы получались менее устойчивыми.)

Я не хотел, чтобы наш язык был строго типизированным. Во-первых, мне хотелось сохранить переопределения типа, подобные REDEFINES в COBOL'e или **union** в C. Переопределения полезны при задании типов записей (вариантных записей в Pascal'e) и для некоторых других случаев. Я ввел атрибут OVER для этого:

```
Name      STRING(24)
NameChar   STRING(1),DIM(24),OVER(Name)
```

Во-вторых, мне хотелось, чтобы групповые структуры обрабатывались, как строки. Это ослабляло требования к типам, потому что в группы могут входить не только строки, но и другие типы данных. Главное требование к группам - их функциональность. Они должны присваиваться, передаваться в качестве параметров и даже сравниваться (корректно). Именно корректно, так как большинство числовых данных сравниваются не как строки, поэтому группы, содержащие числовые данные, обычно правильно не сравниваются. Отрицательные целые с точки зрения битового представления выглядят большими, чем положительные целые, а числа с плавающей точкой при таком подходе вообще нельзя сравнить правильно. При проектировании языка возможен компромисс, и я выбрал функциональность, допустив некоторую вероятность риска.

Очень важно, чтобы типы данных Clarion допускали простое построение процедур общего характера. Если процедура ожидает числовой параметр, то должен быть допустим параметр любого числового типа данных. Я подумал, что смешно требовать наличия разных числовых функций, которые манипулируют данными разных числовых типов, как это имеет места в

производных от ALGOL'a языках. Больше того, я думаю, что полиморфизм, реализованный в C++, который предполагает отдельные функции для каждого типа данных, но который позволяет обращаться к ним по одному и тому же имени функции, представляет собой чисто внешнюю симуляцию полиморфизма.

В первоначальной версии языка Clarion параметры даже не надо было описывать в прототипе. Процедура использовала все, что бы ни было указано в списке аргументов при вызове. Теперь Clarion требует, чтобы параметры были представлены в прототипе, но допускает отсутствие типа данных параметра. Относительно неструктурированных типов данных процедуры в Clarion всегда были истинно полиморфными.

Параметры в Clarion представляются в прототипе для того, чтобы передаваться посредством адреса или самим значением. Сам же язык указатели не поддерживает. По двум причинам: во-первых указатели не несут в себе информации о типе данных и легко могут быть нарушены. Во-вторых обозначения косвенных ссылок (синтаксические отличия указателя от того, на что он указывает) неоправданно усложняют программу. Как показывает мой опыт большинство ошибок в программах на C связано со сбоями в указателях.

Для того, чтобы обеспечить поддержку косвенных ссылок мы выбрали переменные-указатели, наподобие тех, что реализованы в C++. Переменная-указатель содержит и тип данные и указатель на данные. При использовании указатель в переменной автоматически изменяется. И невозможно различение между переменной-указателем и тем, на что она указывает. Посмотрите:

```

CompanyA  FILE
          :
          END
CompanyB  FILE
          :
          END
Company  &FILE                                !Обрабатываемая компания
CODE
CASE CompanyLetter                             !Какую компанию обработать
OF 'A'
  Company &= CompanyA                          !Указательна компанию A
OF 'B'
  Company &= CompanyB                          !Указательна компанию B
END
OPEN(Company)                                !Открыть выбранный файл

```

Значение переменной-указателя *Company* устанавливается оператором присваивания указателя (&=). И компилятор не допустит, если тип данных не будет соответствовать. И с этого момента в любом контексте, где допустимо использование данных, которым соответствует переменная-указатель, можно использовать саму переменную-указатель.



## Промежуточные значения

---

Еще один важный вопрос связан с автоматическим преобразованием типов данных. Я ясно чувствовал, что раз типы данных объявлены, то компилятор обязан их знать! И услужливый компилятор по мере необходимости должен генерировать преобразование типов данных. Кроме того, я был убежден, что хороший компилятор должен проверять значение выражения и обеспечивать логичные преобразования.

Например, если я прибавляю строку к целому числу, то резонно, чтобы компилятор предположил, что строка содержит число в символах ASCII и сгенерировал соответствующее преобразование. И в противоположность этому, если я конкатенирую целое число со строкой, то я неявно прошу компилятор сначала преобразовать целое число в строку. Выбирая соответствующий тип данных для промежуточного значения, компилятор может аккуратно преобразовывать типы данных в выражении, не теряя информации. Если вы делите одно целое число на другое, то хороший компилятор сохранит промежуточное значение, которое будет содержать дробную часть. Если вы прибавляете целое число к строке, то компилятор будет также использовать промежуточное значение с дробной частью, потому что в строке может содержаться дробное значение.

Конечно, информация может теряться при пересылке, например при присвоении или при передаче параметров при обращении к процедуре. При присвоении действительного числа целочисленной переменной отсекается дробная часть. При присвоении действительного числа десятичной упакованной переменной значение округляется до последней значащей десятичной цифры. В некоторых языках, как например в Pascal'e, требуется, чтобы преобразования данных были явно указаны. Я был убежден, что объявляя данные, программист неявно требовал от компилятора ограничить значение элемента данных данным диапазоном.

В первых версиях компилятора Clarion для числовых промежуточных значений использовались всего два типа данных: 32-х битовое целое со знаком (LONG) и 64-х битовое с плавающей точкой. Операция деления или любая операция, в которой используется хотя бы один операнд типа REAL, даст промежуточное значение типа REAL. Такая стратегия обеспечивала достаточную точность, поскольку REAL может давать максимальную значимость числа (15 десятичных цифр), поддерживаемую в Clarion. Хотя значения с плавающей точкой и имеют высокую точность, есть один нюанс. Два эквивалентных выражения такие как  $1/2$  и  $2/4$  могут давать значения с плавающей точкой различающиеся в последнем двоичном разряде. В вычислениях это обычно не имеет значения.

Но не при сравнении. Программист ожидает, что одна вторая равна двум четвертям. Мне можно запретить сравнение значений с плавающей точкой, но я вправе ожидать что приведенное далее логическое выражение будет правильно работать независимо от времени суток:

```
IF Hours > Normal * 1.5
```

Использование промежуточного значения типа REAL для выражения справа от знака ‘>’ бросает тень на результаты сравнения. Мы решили эту проблему в Clarion for Windows, реализовав промежуточное значение с фиксированной точкой, имеющее по 31 десятичной цифре в целой и дробной части. Кроме того, это новшество повысило точность числа до 31-й цифры.

## Управляющие структуры

---

Если языки коммерческих применений COBOL’ и PL/1 имели предпочтительную систему объявления данных, то языки, производные от Алгола, особенно Modula-2, отличались более удобными структурами управления обработкой коммерческих данных. Я модифицировал оператор **IF** языка Modula-2, заменив ключевое слово **THEN** разделителем. В результате исчезли избыточные **THEN** из **IF** структур, занимающих несколько строк. Взяв **ELSEIF** из Modula-2 я избавил язык от необходимости многократных отступов вправо и многочисленных разделителей в сильно вложенных **IF** структурах:

```
IF Number < 0
  Sign = -1
ELSEIF Number > 0
  Sign = +1
ELSE
  Sign = 0
End
```

Оператор **CASE** языка Clarion был создан на базе аналогичного оператора выбора языка Modula-2, в котором метки элементов оператора и интервалы значений относятся к перечисляемому типу данных, что очень полезно. Но мне не нравились разделители. Ключевое слово **OF** предшествует первой метке, последующие же метки начинаются с вертикальной черточки (“|”). Мне эта система разделительных знаков показалась не очень красивой и естественной. Вместо нее я поставил **OF** в начале каждой метки, внутри меток воспользовался ключевым словом **OROF**, а для интервалов значений - ключевым словом **TO**. В результате получилась привлекательная с точки зрения синтаксиса структура:

```
CASE SUB(Name,1,1)
OF('A') TO ('M') OROF('a') TO ('m')
  DO FirstHalf
OF('N') TO ('Z') OROF('n') TO ('z')
  DO SecondHalf
ELSE
  DO FirstHalf
END
```

Только в Modula-2 я видел использование ключевого слова **LOOP** в нормальном контексте. В Модуле-2 **LOOP...END** выполняет безусловный цикл, который завершается выполнением оператора **EXIT**. Я развил эту идею, добавив оператор **CYCLE**, чтобы

перезапустить цикл изнутри. (Мне также пришлось заменить **EXIT** на **BREAK**, потому что **EXIT** использовался для другой цели). Условные циклы были реализованы путем добавления четырех дополнительных предложений к оператору **LOOP**:

```
LOOP I = 1 TO 100 BY 2
LOOP 10000 TIMES
LOOP WHILE Count > 0
LOOP UNTIL EndOfFile
```

Хорошая организация программы требует наличия внутренних подпрограмм. Внутренняя подпрограмма - это блок операторов, отделенных от текста основной программы и выполняемых с помощью оператора вызова внутренней подпрограммы. Если к тому же подпрограмме уметь дать имя, текст основной программы становится короче, ясность же при этом не теряется. В COBOL'e и BASIC'e для выполнения внутренних подпрограмм используются ключевые слова **PERFORM** и **GOSUB**. Локальные процедуры Pascal'я и Modula-2 почти точно подходили, но для объявления типов параметров они требовали наличия операторов прототипов. Я же не хотел, чтобы локальные процедуры имели параметры, потому что считал, что все данные вызывающей процедуры должны быть доступны локальной подпрограмме. Я придумал оператор **ROUTINE** для инициализации внутренней подпрограммы. Подпрограммы располагаются в конце процедуры или функции и выполняются оператором **DO**.

В ряде языков предусмотрен механизм выбора для выполнения одного оператора из списка в зависимости от значения целого числа, определяющего выбор. В FORTRAN'e используется вычисляемый **GOTO**, в COBOL'e - **GOTO...DEPENDING ON**, в BASIC'e - **ON...GOTO** и **ON...GOSUB**. Мне тоже хотелось реализовать подобную возможность, позволяющую выполнять любой тип оператора из списка в зависимости от значения выражения целого типа. Я назвал эту структуру **EXECUTE**, поскольку известная машинная команда **HEX** является сокращением этого же английского слова. Указанная команда выполняет одну из следующих команд, на которую указывает ее операнд. Эта новая структура, как мне кажется, существует только в языке Clarion, но уже доказала свою пользу:

```
EXECUTE UpdateAction
ADD(Master)
PUT(Master)
DELETE(Master)
END
```

## Покорение пользовательского интерфейса

---

В 1970 г., когда я работал в фирме McDonnell Douglas Automation Company, мы купили один из первых компьютеров IV/70 фирмы FOUR Phase Systems, Inc. Это была замечательная машина: 96К памяти на полупроводниковых схемах, что не намного больше, чем в PC (фирма IBM в то время все еще пользовалась магнитными сердечниками). Однако

у этого ящика была одна интересная особенность, а именно: возможность поддержки до 32-х дисплеев на ЭЛТ, информация на которых обновлялась непосредственно из основной памяти компьютера. До IV/70 каждый дисплей, который я использовал был коммуникационным устройством. Можно было видеть как по мере приема отдельных символов, они выводились на экран. В IV/70 экран целиком обновлялся 30 раз в секунду. Это была отличная платформа для диалоговых программ. Но никто не знал, как этим воспользоваться. Фирма Four Phase Systems продавала свою систему взамен сгруппированных дисплеев фирмы IBM и как многотерминальный клавишный перфоратор.

Я пошел дальше этих применений и в 1987 г. основал компанию для разработки “под ключ” медицинской информационной системы на базе компьютера IV/70. Мне пришлось написать многопользовательскую операционную систему (ОС) и макро-язык, для работы с ней. Потом был написан макро-препроцессор и небольшая медицинская информационная система - на все ушло 9 месяцев.

В макро-языке дисплеи рассматривались как области памяти (так на самом деле и было!). Доступ к ним осуществлялся посредством макроопределений MOVE. Описание полей для ввода данных помещалось в таблицу полей, после чего управление передавалось ОС для последующей обработки. При завершении обработки поля или при нажатии специальной клавиши, ОС возвращала управление программе. Эта стратегия имела отчетливое смещение “центра тяжести” на операционную систему. Функциональные клавиши связывались с экранными процедурами. Экранные процедуры создавали таблицы полей, которые связывались с процедурами редактирования полей. Не было “выполнения программы” в обычном значении этого термина. Фактически не было как таковой и программы - просто набор процедур, которые реагировали на события в операционной системе. Во главе была операционная система. Она доводила до программиста сигналы о том, что для нее требовалось выполнить. Со временем наши программисты настолько преуспели в таком подходе, что многие больничные системы проектировались, реализовывались и всесторонне тестировались быстрее чем в больнице производился монтаж терминалов и кабельных сетей.

Но это было вовсе не просто. Каждый из наших программистов преодолел крутую кривую обучения. Событийно-управляемое программирование очень тяжело для понимания. Позднее, во один из самых ярких моментов озарения, которые когда-либо переживал, я вдруг понял, что событийно-управляемая операционная система могла бы в свою очередь управляться обычной программой. Обработка пользовательского интерфейса могла бы выполняться одним оператором. В Clarion я назвал его АССЕРТ. Одна часть оператора АССЕРТ должна возвращать управление операционной системе, а другая могла служить точкой входа для обработки всех событий. Небольшой набор функций служил бы для идентификации происходящих событий и связанных с ними полей.

Событийно-управляемая система всегда казалась мне внешней по отношению ко мне. В ней я был рабом на галере, прикованным к веслу и подчиняющимся барабанщику,

обрабатывающим его события.

Я осознавал, что АСCEPT сделает меня снова главным. Теперь барабан у меня! Я хотел обращаться к операционной системе и никак иначе.

А как Clarion будет описывать структуру экрана? Если символы и поля на экране - данные, тогда формат экрана должен быть структурой данных. Не мудрствуя лукаво я назвал эту структуру **SCREEN**. Оператор **OPEN(MyScreen)** должен открывать экран. АСCEPT подключать клавиатуру и отслеживать работу пользователя по вводу данных. Когда пользователь завершает ввод поля или нажимает “горячую” клавишу, оператор АСCEPT “проваливается”, возвращает управление программе. Оператор **CLOSE(MyScreen)** восстанавливает состояние экрана до его открытия.

Возможность объявлять форматы экрана упростила не только обработку, но и проектирование форматов. Бригада разработчиков включила в состав исходного текста редактора Clarion формater экрана, который позволил порождать структуры **SCREEN**. Формater экрана может также считывать структуры **SCREEN**. Как это сделать? Поместите курсор на структуру **SCREEN** и вызовите формater экрана. Он интерпретирует исходный текст и отобразит на экране соответствующие формату данные. Теперь измените что-нибудь в рисунке и выйдите из программы. Формater экрана внесет изменения в исходный текст структуры **SCREEN** и заменит старую версию структуры новой.

Аналогичную структуру я придумал и для отчетов. Структура **REPORT** содержит информационные строки страницы, верхний и нижний колонтитулы. Оператор **PRINT** управляет автоматическим форматированием и переполнением страниц. Также как и для структур **SCREEN** в исходный текст редактора включен формater отчетов для поддержки структур **REPORT**.

## Раскрытие окон

---

Наша реализация пользовательского интерфейса отлично подходила к Microsoft Windows. Программисты под Windows переживали очень трудные времена - кто может упрекнуть их! Пример программы “Hello World”, поставляемый с популярным продуктом C++ был длиной 8 страниц! Подобная циклу АСCEPT в Clarion модель передачи сообщений была отчаянно необходима для Windows. Мы решили реализовать ее под Windows.

Мы превратили структуры **SCREEN** в структуры **WINDOW**, введя необходимую для объявления свойств экранных объектов грамматику. Чтобы соответствовать интерфейсу с несколькими документами, мы ввели возможность организации нескольких процессов выполнения в рамках одной программы. Изменили грамматику структуры **REPORT**, чтобы изображать печатный документ в режиме WYSIWYG, ввели возможность создания “форматки”, вложенные групповые заголовки и итоги.

Оператор АСCEPT стал структурой, определяющей границы цикла обработки событий. Мы разработали компилятор вместе с библиотекой времени выполнения, чтобы скрыть направление обращения к процедурам, используемым при обработке экранных событий. Обращение к обработчику окон генерируется перед циклом АСCEPT. Сам этот цикл генерируется как вложенная процедура.

Обработчик окон создает необходимые экранные объекты, задает общие процедуры обработки событий, для каждого события генерируемого каждым объектом. Этот обработчик управляет перехватом событий, таких как перерисовка окна и обращение к вложенной процедуре для обработки другого события.

Когда окно закрывается, обработчик окон передает управление оператору, следующему за циклом АСCEPT.

Для программиста на Clarion все это совершенно просто. Открыл окно, затем вошел в цикл АСCEPT. Этот цикл выполняется для каждого события, на которое программа должна отреагировать. Закрыл окно и вышел из цикла.

Мы определили набор удобных функций для идентификации событий и связанных с ними объектов. Программа обработки типового диалогового окна выглядит приблизительно так:

OPEN(Window)	!Открыть окно
ACCEPT	!Начать работу в окне
CASE FIELD()	!Какое поле обрабатывается ?
OF ?OK	! кнопка OK ?
CASE EVENT()	! Какое событие произошло ?
OF EVENT:Selected	! кнопка OK нажата ?
:	! обработать кнопку OK
CLOSE(Window)	! закрыть окно
END	! Конец структуры CASE EVENT()
OF ?CANCEL	! кнопка CANCEL ?
CASE EVENT()	! Какое событие произошло ?
OF EVENT:Selected	! кнопка CANCEL нажата ?
:	! обработать кнопку CANCEL
CLOSE(Window)	! закрыть окно
END	! Конец структуры CASE EVENT()
ELSE	! Может быть событие не связанное с полем
CASE EVENT()	! Какое событие произошло ?
OF EVENT:CloseWindow	! Окно закрывается ?
:	! сделать что-то перед закрытием окна
CLOSE(Window)	! закрыть окно
END	! Конец структуры CASE EVENT()
END	! Конец структуры CASE FIELD()
END	!Конец цикла ACCEPT
RETURN	!Передать управление обратно

Объявления структуры экрана, отчетов и файлов очень понятные. Но кроме того, они

имеют и ограничения. Поскольку они компилируются, изменить во время выполнения программы большинство объявлений нельзя. Большинство пожеланий программистов, пишущих на Clarion, о совершенствовании языка связано с тем, чтобы сделать возможным доступ к значениям объявляемых атрибутов и их изменение в самой программе.

В нашей библиотеке времени выполнения под Windows эти структуры являются объектами. И их свойства можно изменять. Стандартное в объектно-ориентированных языках написание *объект.свойство* мы реализовать не могли, поскольку уже использовали точку в качестве терминатора структур и десятичной точки. Поэтому выбрали фигурные скобки, в которые и заключили свойство. С помощью такого написания значение любого объявленного в программе атрибута, например, текста надписи на кнопке можно изменить вот таким оператором:

```
?Button{PROP:Text} = 'My Button'
```

## Проектирование базы данных

Мне хотелось реализовать доступ к баз данных с простым синтаксисом, который позволял бы применять все три стандартных метода доступа к файлам: прямой, последовательный и индексный. Организация файла также должна быть простой: заголовок, за которым идут информационные записи фиксированной длины. Заголовок описывает структуру записи, связанные с ней ключи и поля типа memo, расположенные в отдельных файлах. Эта схема похожа на схему, принятую в dBASE: доступ к записи может быть последовательным или прямым через ключ или относительный номер записи. Чтобы объявлять файлы и их компоненты я ввел структуру **FILE**, аналогичную структуре **FD** языка COBOL:

Detail	File,PRE(Dt1),NAME('C:\LEDGER\DETAIL.DAT')
AcctKey	KEY(Dt1:AcctNo,Dt1:Period,Dt1:Date)
BatchKey	KEY(Dt1:Batch,Dt1:Period),DUP
Comment	MEMO(4096)
	RECORD ! Запись Detail
AcctNo	SHORT ! Номер счета
Period	BYTE ! Отчетный период
Date	DATE ! Дата транзакции
Batch	STRING(12) ! Идентификатор партии
Amount	DECIMAL(12,2) ! Итог (+/- = дебит/кредит)
	END
	END

Я реализовал последовательную обработку с помощью ключевых слов **SET, NEXT, PREVIOUS, SKIP**. Команда **SET** задает последовательность (с помощью ключа или относительного номера записи) и начальную точку обработки для трех других команд, которые задают прямое и обратное считывание и пропуск записей. Эти ключевые слова прекрасно сочетаются с функцией “конец файла” (**EOF**) в цикле чтения:

```
SET(Dt1:AcctKey) ! Установить последовательность
```

```
LOOP UNTIL EOF(Detail)      ! обработки номеров счетов
NEXT(Detail)                 ! Цикл обработки каждой записи
:                             ! Читать следующую запись
END
```

Команда **GET** читает произвольную запись по ключу или относительному номеру. Важно, что **GET** не мешает последовательной обработке, поскольку эта команда не устанавливает следующую для обработки запись. Команды **PUT** и **DELETE** обрабатывают записи, доступ к которым был получен с помощью команд **NEXT**, **PREVIOUS** или **GET**. Команда **ADD** включает новую запись в БД. Как оказалось, такая концепция доступа эффективна, надежна и универсальна - существенная и популярная составная часть нашего продукта.

Однако, по мере роста популярности языка Clarion приходилось вводить новые возможности. Сначала разработчикам потребовался доступ к файлам dBASE, поэтому мы добавили библиотеку процедур по работе с dBASE (библиотеки процедур Clarion'a принято называть "Language Extension Modules" или LEM'ы - модули расширения языка). Потом фирма Novell разработала систему поддержки связи клиента и сервера для Btrieve (с ведением индекса на сервере). Для некоторых крупных прикладных систем на Clarion'e требовалась Btrieve для повышения производительности при обработке транзакций. В результате появились Btrieve LEM'ы, разработанные двумя нашими сторонними партнерами.

А еще оставались DB2 и RDB, и Oracle, и SQL Server, и любые другие системы по управлению базами данных, работающие на ПК или к которым ПК имеет доступ. Мы добавили поддержку прямых вызовов функций на C в следующих версиях языка с таким расчетом, чтобы программа на Clarion'e могла работать с БД, для которой имеется API на языке C. Но мне стало ясно, что это не может решить всех проблем. Нельзя допустить, чтобы в универсальном языке для коммерческих применений использовались разные наборы процедур доступа в зависимости от формата БД. Работа с файлами не должна приводить к необходимости переделки программ. Для языка Clarion нужна была стандартизованная, встроенная поддержка для работы со всеми широко распространенными базами данных.

Нам предложили принять в качестве синтаксиса запросов к базе данных синтаксис SQL. Я отнесся к предложению серьезно и переделал несколько типичных программ на Clarion'e, используя встроенный SQL. Но вскоре мне стало ясно, что это порочная идея. Для написания программ синтаксис SQL слишком многословен и не изящен. На SQL небольшой цикл из четырех строк, приведенный выше, становился таким альбатросом:

```
DECLARE X CURSOR
FOR SELECT *
FROM Detail
ORDER BY Dtl:AcctNo,Dtl:Period,Dtl:Date
```



```

END
END
OPEN X
LOOP
  FETCN X
  IF ReturnCode = 100 TNEN BREAK.
:
END
CLOSE X

```

Концепция курсора в SQL не только не изящна, но и почти бесполезна. Нельзя с помощью курсора пропустить часть информации, например, чтобы вновь вывести предыдущую страницу записей. Нельзя также произвольно изменить его позицию, например, перескочить к слову “Jones” при просмотре записей по алфавиту. Я пришел к выводу, что, если бы мне суждено было заменить синтаксис доступа к БД Clarion’a на синтаксис SQL, меня бы вымазали дегтем, извалили в перьях и выставили из города.

Поэтому мы пошли по пути создания заменяемых драйверов баз данных. Программистам на Clarion’e нравился метод доступа к БД этого языка, но им нужна была поддержка форматов других БД. На основе имеющейся структуры языка, мы расширяли их кругозор и повышали возможности разработанных прикладных систем. С помощью новой технологии драйверов баз данных мы одинаково реализуем доступ ко всем базам данных - нетривиальное преимущество продукта.

## Новая структура VIEW

Для того, чтобы сделать драйвер SQL, мы отображали его синтаксис на нашу грамматику доступа к базам данных. Наш оператор **SET** конструирует предложение **SELECT** языка SQL, которое выполняется при первом выполнении оператора **NEXT** или **PREVIOUS**. Если меняется направление (например **NEXT ... PREVIOUS**), драйвер выдает другой **SELECT** с другим уточнением **ORDER BY**. Наш оператор **GET** порождает **SELECT...FETCH**. **ADD** порождает **INSERT**; **GET...DELETE** порождает **DELETE**; а **GET...PUT** порождает **UPDATE**. Несколько возможностей, таких как относительный доступ к записям (возможно, имеется в виду вложенный **SELECT** - прим. перев.) не поддерживается для SQL баз данных, но в остальном реализация совершенно полная.

Однако с помощью нашей грамматики было невозможно реализовать некоторые очень важные возможности SQL. В программах на Clarion фильтрация записей реализуется посредством последовательного чтения и пропуска неподходящих записей:

```

LOOP UNTIL EOF(Part)
  NEXT(Part)
  IF Prt:OnHand > 0 THEN CYCLE .
:
END

```

Базы данных с SQL языком фильтруют записи на сервере и тем самым экономят

пользователю много времени. В Clarion программах соединение файлов происходит посредством чтения первичной записи по первичному ключу, для того, чтобы, узнав значение вторичного ключа, прочитать вторичную запись. В базах данных с SQL языком первичная и вторичная записи возвращаются в ответ на один запрос. В Clarion программах при каждом обращении в базу данных считываются все поля записи записи. В SQL языке пользователю передаются только требуемые поля.

Конечно, и SQL не может самостоятельно определить, что считывать. Нужно сообщить ему, что он должен сделать. Для этого мы разработали структуру VIEW.

```
View      VIEW(Part),FILTER('PRT:OnHand = 0')
          PROJECT(PRT:Number,PRT:Name,PRT:OnHand,PRT:Usage)
          JOIN(Vendor,PRT:Vendor,VND:Number)
          PROJECT(VND:Name,VND:Address,VND:CityStateZip)
          END
          END
```

Структура VIEW объединяет намерения программиста, и таким образом драйвер может использовать любой сервис, предоставляемый ему ядром соответствующей системы баз данных. Драйвер или сам выполняет фильтрацию (отбор записей), соединение (поиск вторичной записи) и проекцию (выборку полей) или запрашивает выполнение этих операций на сервере базы данных. В обоих случаях оптимизируется производительность.

Также проблемой была реализация под SQL оптимистическая стратегия совместного использования данных. Для того, чтобы обновлять совместно используемый файл, Clarion-программа считывает и запоминает запись. Затем, перед изменением записи в базе, запись блокируется, повторно считывается и сравнивается с запомненной копией. Если они одинаковые, то измененная запись заносится в базу данных. В противном случае запись изменена другой рабочей станцией и об этом оповещается оператор. Такой процесс называется “оптимистическая стратегия совместного использования” и основывается на предположении, что запись за этот промежуток времени обычно не изменяется.

В SQL оптимистическая стратегия совместного использования реализуется предложением WHERE, которое требует чтобы все подлежащие обновлению поля в период времени после первого считывания записи сохранили свои значения. Если изменилось хоть одно из этих полей, SQL возвращает соответствующее сообщение об ошибке. Поскольку в Clarion не было синтаксической конструкции для выполнения такого запроса относительно связанных файлов, мы добавили для этого оператор WATCH. Для того, чтобы инициировать работу по оптимистической стратегии, перед оператором GET, NEXT или PREVIOUS выполняется оператор WATCH. Когда считывается запись, драйвер запоминает ее копию. При выполнении оператора PUT драйвер или повторно считывает запись или выдает запрос UPDATE...WHERE базе данных SQL. Если запись за это время изменилась, оператор PUT возвращает соответствующий код ошибки.

## Первый компилятор

---

В комплект поставки версии 1.0 Clarion'a, отгруженной в мае 1986 г., входили и компилятор, и интерпретатор. Компилятор вырабатывал промежуточный код, который затем интерпретировался утилитой Processor. Промежуточный код был настолько компактен, что большие прикладные системы на Clarion'е помещались в небольшую по объему основную память (256К), которая была у ПК типа IBM PC того времени. Такой сжатый код был результатом того, что компилятор генерировал двоичное описание каждого оператора объявления данных. Далее данные адресовались двубайтовым указателем на двоичное описание. Поэтому требовалось пять байтов, чтобы сложить целую и строковую величины, а результат отформатировать по шаблону (один байт для операции сложения и четыре байта для указателей на описатели целого числа и строкового шаблона). Для каждой операции Processor проверял типы данных операндов и проводил необходимые преобразования типов.

Но сжатый промежуточный код не был основной целью этого проекта. Интерпретируя данные на выходе компилятора Processor мог выполнить прикладную программу на Clarion'е без дополнительного этапа компоновки. И это было существенно. В 1986 г. и еще долго после этого компоновка оставалась одним из самых медленных процессов. Наши клиенты высоко оценивали скоростные характеристики тестовых примеров, но при этом высказывали замечания, что “настоящие” языки программирования приводят к появлению .EXE файлов! В начале следующего года мы выпустили транслятор, который преобразовывал промежуточный код в .OBJ файлы, заменяя коды операций вызовами процедур. Указатели передавались в качестве параметров. Этот подход прослужил нам верой и правдой шесть лет, но и он породил ряд проблем:

- \* Возникли трудности с использованием внешних библиотек. Файлы .OBJ можно было включить в файл .EXE, но выполнить напрямую Processor'ом было нельзя. Мы разработали процесс, который преобразовывал подходящие файлы .OBJ в специальный двоичный формат (LEM), который, в свою очередь, мог быть выполнен процессором и вновь преобразован в .OBJ файл транслятором. Но процесс оказался довольно сложным и применялся только опытными разработчиками.
- \* Простые программы на Clarion'е порождали большие по объему .EXE файлы. Процедуры принятия решений на этапе выполнения составляли списки библиотечных процедур, которые включались в .EXE файлы, но никогда не вызывались. В результате известная программа “Hello World” заняла 141К памяти.
- \* Прикладные программы на Clarion'е работали медленнее, чем программы на C, Pascal'е и Modula-2, потому что программы на Clarion'е проверяли типы данных во время выполнения, другие же языки, делали это на этапе компиляции.
- \* Больше не было необходимости избегать компоновки в цикле тестирования. Новые компоновщики, которые поддерживали библиотеки времени выполнения могли

скомпоновать программу для тестирования со скоростью загрузки утилиты Processor в память.

Но самое главное, нам нужна была технология, которая бы обеспечила разработчикам переход к Windows, OS/2, UNIX, 32-разрядной машине и архитектурам не фирмы Intel.

## Новый партнер

---

В мае 1990 г. мы решили эти и многие другие проблемы, купив лицензию на технологию фирмы Jensen & Partners International (JPI), разработчиков продуктов серии TopSpeed из Великобритании. JPI была основана в 1988 году Нильсом Йенсеном (Niels Jensen), основателем компании Borland International, которую он покинул со всей своей группой разработчиков языков. Они купили свои незаконченные разработки и выпустили серию продуктов TopSpeed. JPI разработала компиляторы языков C, C++, Pascal, Modula-2, в состав которых входил один и тот же оптимизирующий генератор кода. Фирма JPI назвала компиляторы “внешними” компонентами, а генератор кода - “внутренним” компонентом.

Мы сразу же приступили к созданию внешнего компонента Clarion'a. И, конечно, сделать это оказалось труднее, чем мы думали. Потребовалось внести в язык больше изменений, чем мы ожидали. На разработку проекта ушло больше времени и средств, чем предполагалось вначале. Но результаты нас ошеломили.

Мы знали, что внутренний компонент системы TopSpeed - очень хороший продукт, но никак не ожидали, что программа, реализующая на Clarion'е алгоритм “Решето Эратосфена” (поиск простых чисел) будет работать в два раза быстрее, чем аналогичная программа на Turbo C++ фирмы Borland. Мы также закупили лицензию на технологию компоновки TopSpeed, но я еще до конца не понял ее преимущества. Уникальный компонент “Smart Linking” (интеллектуальная компоновка) системы TopSpeed убирал из .EXE файлов все процедуры, на которые не было ссылок, и статические элементы данных. Более того, пока мы работали над нашим внешним компонентом, фирма JPI разработала автоматический загрузчик оверлеев, DLL DOS, DOS Extender и объявила о поддержке 32-разрядной архитектуры. Благодаря этим новшествам мы, наконец, устранили отставание в быстродействии, которое всегда было свойственно языкам высокого уровня, предназначенным для административных и экономических задач.

В сентябре 1991 г. на первой конференции разработчиков программ в среде Clarion мы сделали сообщение о нашем новом продукте. Новые характеристики и взаимосвязь с системой TopSpeed вызвали восторженные отклики в прессе. Воспользовавшись торжеством, мы с Нильсом Йенсеном, основателем JPI, начали переговоры о слиянии компаний. Оно имело бы огромное значение - компиляторы TopSpeed приобретали бы огромный американский рынок программных продуктов, а Clarion - их внутреннюю технологию. После длительных переговоров в апреле 1992 года произошло слияние. Два с половиной года спустя компании полностью объединили свои операции и серии продуктов и объединенная компания была переименована в TopSpeed Corporation. В октябре 1994

года TopSpeed Corporation выпустила на рынок Clarion for Windows первый продукт целиком разработанный объединенной компанией.

## **Что теперь**

---

Эти заметки первоначально представляли собой вступление к Руководству программиста, которое поставлялось с Clarion Database Developer версии 3.0, выпущенному в апреле 1993 года. Версии Clarion для Windows требовались обширные добавления и пересмотр подходов. Таково уж развитие. Разработка программного обеспечения мне представляется как процесс тихого покачивания китайской доски - игрушки, в которой все мраморные шарики должны заполнить ямки. В этой аналогии финал - законченный программный продукт. До выхода в свет Clarion for Windows я чувствовал, что мы далеки еще от цели. Теперь я думаю иначе. Совсем немного шариков еще катается.



---

## **Глава 1 Введение**

---

### **Справочное руководство по языку**

Clarion for Windows представляет собой интегрированную среду для разработки информационных систем и прикладных программ по обработке данных, предназначенных для использования на персональных компьютерах в операционной среде Windows. Основой этой среды является язык программирования Clarion. В данном руководстве этот язык описан кратко, в модульном стиле. И хотя это не учебник, когда нужно узнать точный синтаксис написания какого-либо оператора, объявления или функции следует в первую очередь обращаться к этой книге.

Везде, где это было возможно каждый элемент сопровождался реальными примерами.

---

### **Построение книги**

#### **Глава 1 - Введение**

Глава 1 представляет собой введение в справочное пособие по языку Clarion. В ней содержится краткий обзор содержимого каждой из глав и правила, помогающие читателю понять соглашения, принятые при изложении материала книги.

#### **Глава 2 - Формат исходного текста программы**

В главе 2 описывается общая структура программы для Windows на языке Clarion. В ней приведены пунктуация, специальные символы, зарезервированные слова и детальное описание “строительных блоков”, необходимых для создания модульной, структурированной Clarion-программы.

#### **Глава 3 - Объявление переменных**

В главе 3 описываются типы данных, используемые при объявлении переменных в Clarion-программах. Кроме того, в ней определяются и иллюстрируются форматные маски, называемые “шаблонами”.

#### **Глава 4 - Выражения и присвоение**

В главе 4 определяется синтаксис, требующийся для того, чтобы объединить переменные, функции и константы в числовые, строковые и логические выражения. Кроме того в ней определяется, как значение выражения присваивается переменной.

#### **Глава 5 - Управляющие операторы**

В главе 5 описываются сложные исполняемые операторы, которые управляют последовательностью выполнения программы.

#### **Глава 6 - Структуры для описания окна**

В главе 6 описываются структуры данных APPLICATION и WINDOW и все их атрибуты.

#### **Глава 7 - Оконные элементы управления**

В главе 7 описываются объекты, которые могут размещаться в структуре APPLICATION или WINDOW

#### **Глава 8 - Операторы для работы с окнами**

В главе 8 описываются исполняемые операторы и функции, относящиеся конкретно к структурам APPLICATION и WINDOW.

### **Глава 9 - Печатные отчеты**

В главе 9 описывается структура данных REPORT и все ее компоненты и атрибуты. В ней также приведены исполняемые операторы и функции, специфичные для структуры REPORT.

### **Глава 10 - Графические команды**

В главе 10 описываются исполняемые операторы и функции, с помощью которых в окнах APPLICATION и WINDOW, а также печатных документах рисуются графические примитивы.

### **Глава 11 - Файлы данных**

В главе 11 описываются объявления, исполняемые операторы и функции, при помощи которых осуществляется доступ к файлам данных. В ней также изложены сведения по операторам и функциям, требующимся при разработке многопользовательских систем и систем обработки транзакций.

### **Глава 12 - Структуры для организации виртуального файла**

В главе 12 описывается структура VIEW. В ней также приведены исполняемые операторы и функции, используемых при обращении к данным посредством виртуального файла.

### **Глава 13 - Очереди в памяти**

В главе 13 описывается работа со структурами данных, называемыми QUEUE (очередь), которая используется для эффективной обработки информации в оперативной памяти. Наряду со всеми составляющими этой структуры и ее атрибутами в этой главе также приведено описание исполняемых операторов и функций, которые относятся к использованию структуры QUEUE.

### **Глава 14 - Прочие операторы и функции**

В главе 14 приведено описание операторов и функций, которые не имеют непосредственного отношения к структурам данных или тематическим областям, раскрываемым в главах с 1 по 13.

## **Приложение А - Справочное руководство по библиотекам DDE, OLE и ODX**

В этом приложении описываются операторы и функции, с помощью которых выполняется динамический обмен данными с другими параллельно выполняющимися приложениями Windows (DDE), связывание и внедрение объектов (OLE) и связывание и внедрение пользовательских объектов (ODX).

### **Приложение Б - Коды клавиш**

В приложении Б описываются мнемонические коды соответствия клавиш, которые улучшают читаемость исходных текстов на Clarion.

### **Приложение В - Присвоение значений свойствам**

В этом приложении описывается синтаксис присвоения значений свойствам и все свойства, доступные во время выполнения программы.

### **Приложение Г - Сообщения об ошибках**

В приложении Г описываются сообщения об ошибках времени компиляции и



выполнения

## Приложение Д - Мнемонические имена событий

В этом приложении описываются мнемонические имена событий, которые улучшают читаемость исходных текстов на Clarion.

### Соглашения и обозначения

---

Обозначения используемые в синтаксических диаграммах:

<u>Обозначение</u>	<u>Значение</u>
[ ]	В квадратные скобки заключаются необязательные атрибуты и параметры.
( )	В скобки заключается список параметров.
	Вертикальными линиями ограничивается список параметров, из которых допустим один и только один параметр..

Примеры в этой книге набираются так:

IF NOT SomeDate	!IF и NOT это ключевые слова
SomeDate = TODAY()	!SomeDate - это имя переменной
END	!TODAY и END это ключевые слова

### КЛЮЧЕВЫЕ СЛОВА ЯЗЫКА CLARION

Все слова, набранные "ПРОПИСНЫМИ БУКВАМИ" являются ключевыми словами языка Clarion.

Имена Переменных      Для улучшения читаемости используются и прописные и строчные буквы

Комментарии      В основном строчными буквами

Эти соглашения призваны улучшить читаемость понимание примеров.

### Формат описания элементов языка

---

Каждый элемент языка программирования Clarion, упомянутый в этом руководстве печатается ЗАГЛАВНЫМИ буквами. Информация по компонентам языка приводится в виде синтаксической диаграммы, подробного описания и примера исходного текста.

Элементы описываются логически сгруппированными на основании иерархических связей между ними, поэтому оглавление этой книги не упорядочено по алфавиту. Обычно типы и структуры данных приводятся в начале главы, за ними следуют их атрибуты, а исполняемые операторы и функции в конце.

Формат описания используемый в этом руководстве иллюстрируется синтаксической диаграммой, приведенной ниже.

## КЛЮЧЕВОЕ\_СЛОВО (краткое описание назначения)

[метка] **КЛЮЧЕВОЕ СЛОВО** параметр1[параметр2][АТРИБУТ1() ][АТРИБУТ2() ]  
 список  
 альтернатив

**КЛЮЧЕВОЕ СЛОВО**      Краткое описание того, что делает данное **КЛЮЧЕВОЕ СЛОВО**

параметр1      Полное описание параметра1, вместе с тем как он связан с параметром2 и **КЛЮЧЕВЫМ\_СЛОВОМ**.

параметр2      Полное описание параметра2, вместе с тем как он связан с **КЛЮЧЕВЫМ\_СЛОВОМ**. Квадратные скобки, в которые заключен этот параметр, означают, что он является необязательным и может быть опущен.

список альтернатив      Полное описание альтернативных значений параметра1 вместе с тем, как они связаны с параметром2 и **КЛЮЧЕВЫМ\_СЛОВОМ**.

**АТРИБУТ1()**      Высказывание, описывающее связь **АТРИБУТА1** с **КЛЮЧЕВЫМ\_СЛОВОМ**

**АТРИБУТ2()**      Высказывание, описывающее связь **АТРИБУТА2** с **КЛЮЧЕВЫМ\_СЛОВОМ**

Далее идет короткое описание того, что реализует **КЛЮЧЕВОЕ\_СЛОВО**. Часто **КЛЮЧЕВОЕ\_СЛОВО** представляет собой атрибут **КЛЮЧЕВОГО\_СЛОВА**, которое описывалось в предшествующем тексте. Иногда у **КЛЮЧЕВОГО\_СЛОВА** нет параметров и/или атрибутов.

Генерируемые события: Если для **КЛЮЧЕВОГО\_СЛОВА** генерируются события, то здесь они перечисляются.

Тип возвращаемого значения: Тип возвращаемого значения если данное **КЛЮЧЕВОЕ\_СЛОВО** является функцией.

Выдаваемые сообщения об ошибках: Если при использовании **КЛЮЧЕВОГО\_СЛОВА** возможно возникновение ошибочных ситуаций, которые можно обнаружить с помощью функций **ERROR** и **ERRORCODE**, то эти ситуации перечисляются в этом разделе описания **КЛЮЧЕВОГО\_СЛОВА**.

### Пример:

FieldOne = FieldTwo + FieldThree      !Это пример исходного текста  
 FieldTwo = KEYWORD(FieldOne, FieldThree)      !Комментарий следует после  
 восклицательного знака

**Смотри также:** **АТРИБУТ1**, **АТРИБУТ2** и другие, имеющие отношение к рассматриваемому, ключевые слова, соглашения и обозначения.





---

## **Глава 2. Формат исходного текста программы**

### **Формат оператора**

Язык Clarion является языком “ориентированным на операторы”. Такие языки исходят из того факта, что программы существуют в виде ASCII-файлов с исходным текстом - каждая строка программы соответствует записи файла. Поэтому вместо знаков препинания можно воспользоваться разделителями записей (символами “возврат каретки” и “перевод строки”).

В общем формат оператора в языке Clarion следующий:

метка        ОПЕРАТОР [(параметры)] [,АТРИБУТ[(параметры)]] ...

Атрибуты задают характеристики элемента и используются только в операторах объявления данных. Исполняемые операторы имеют вид стандартного обращения к процедуре, за исключением операторов присваивания ( $A=B$ ) и управляющих структур (таких как IF, CASE и LOOP).

Метки операторов должны начинаться в первой колонке строки исходного текста. Оператор без метки не может начинаться в первой колонке. Оператор заканчивается символом конца строки. Если оператор не помещается на одной строке, то он продолжается на следующую строку с помощью символа “|” (вертикальная черта). Кроме того, для разделения нескольких операторов на одной строке можно использовать “;” (точка с запятой).

Будучи ориентированным на операторы, язык Clarion исключает многие из знаков пунктуации, обязательных в других языках для идентификации меток и разделения операторов. Блок операторов начинается одним обозначением составного оператора, а заканчивается оператором END (или точкой).

---

### **Имена переменных и метки операторов**

Операторы языка в исходном модуле можно разделить на две категории: операторы объявления данных и исполняемые операторы или просто “данные” и “программный код”.

Операторы объявления данных резервируют области памяти, которыми при выполнении программы манипулируют исполняемые операторы. С помощью метки можно идентифицировать или сослаться на любой исполняемый оператор или оператор объявления данных. С помощью меток осуществляются ссылки на все переменные, структуры данных, процедуры, функции и локальные подпрограммы.

Метка указывает конкретное место в программе. Любой исполнимый оператор может иметь метку для того, чтобы использоваться в качестве точки перехода в операторе GOTO.

Даже если на нее нет ссылок, метка увеличивает на десять байт размер исполняемого оператора.

Метка оператора PROCEDURE или FUNCTION является именем процедуры или функции. Использование метки оператора PROCEDURE в качестве исполняемого оператора вызывает выполнение этой процедуры. Метка оператора FUNCTION используется в выражении или списке параметров другой процедуры для того, чтобы представить значение, возвращаемое функцией.

Правила образования допустимых в Clarion меток:

- \* Метка ДОЛЖНА начинаться в первой колонке строки исходного текста.
- \* Метка может содержать буквы (латинские прописные и строчные), цифры от 0 до 9, знаки подчеркивания(\_), и двоеточия(:).
- \* Первым символом должна быть буква или знак подчеркивания.
- \* Регистр букв не имеет значения, т.е. CurRent и CURRENT представляют собой одну и ту же метку.
- \* Резервированное слово не может использоваться в качестве метки.

## Структуры

---

Когда операторы объявления данных вложены в другие операторы объявления данных, то получаются составные структуры данных. В языке Clarion существует много составных структур данных: APPLICATION, WINDOW, REPORT, FILE, RECORD, GROUP, VIEW, QUEUE и т.д. Такие составные структуры данных должны заканчиваться точкой (.) или ключевым словом END. Операторы IF, CASE, EXECUTE, LOOP, BEGIN и ACCEPT представляют собой исполняемые управляющие структуры, которые также должны заканчиваться точкой или оператором END.

## Уточнение имени переменной

---

У переменных, объявленных внутри составных структур данных (GROUP, QUEUE, FILE, RECORD, и т.д. ) могут быть совпадающие имена, если только они не содержатся в одной и той же структуре. Для того, чтобы точно указать поле, имеющее имя, которое совпадает с именем поля в другой структуре, и обеспечить уникальность имени, можно использовать атрибут PRE этой структуры так как он описан в документации (Префикс:ИмяПеременной). Тем не менее использование для этой атрибута PRE необязательно и его можно опустить.

К любой переменной в составной структуре данных можно обратиться, присоединяя к имени переменной спереди имя структуры через точку (ИмяСтруктуры.ИмяПеременной). Такой способ уточнения имени следует использовать для структур, для которых не указан атрибут PRE. Для ссылок на переменные внутри любых структур кроме структур CLASS и поименованных переменных-указателей, вместо точки можно использовать двоеточие (это

служит только для обеспечения совместимости с предыдущими версиями Clarion для Windows).

Для уточнения имени переменной, которая находится во вложенной составной структуре, следует спереди присоединить имя, каждой структуры предыдущего уровня (если она имеет имя). Если же у какой-либо из вложенных структур отсутствует имя, то эта часть из уточнения имени опускается. Это похоже на анонимные структуры UNION в C++. Это означает, что в случае структуры FILE (без атрибута PRE), в которой структура RECORD имеет имя, на отдельные поля записи нужно ссылаться таким образом: ИмяФайла.ИмяСтруктурыRecord.ИмяПоля. Если у структуры RECORD нет имени, то на отдельные ее поля можно ссылаться так: ИмяФайла. ИмяПоля

Такой синтаксис уточнения имен используется также для ссылок на все элементы структур CLASS - и на элементы данных и на методы. Для того, чтобы обратиться методу - элементу структуры CLASS, укажите НазваниеКласса.ИмяМетода в любом месте где допустимо обращение к процедуре или функции.

### Пример:

```
MasterFile  FILE,DRIVER('TopSpeed')
Record      RECORD
AcctNumber  LONG      !На эту переменную ссылка осуществляется так
Masterfile:Record:AcctNumber

..
Detail      FILE,DRIVER('TopSpeed')
            RECORD
AcctNumber  LONG      ! На эту переменную ссылка осуществляется так Detail:AcctNumber

..
Memory      GROUP,PRE(Mem)
Message     STRING(30) !Можно ссылаться как Mem:Message или Memory:Message
            END
SaveQueue   QUEUE
Field1      LONG      ! На эту переменную ссылка осущ. так SaveQueue:Field1
Field2      STRING    ! На эту переменную ссылка осущ. так SaveQueue:Field2
            END
OuterGroup  GROUP
Field1      LONG      ! На эту переменную ссылка осущ. так OuterGroup:Field1
Field2      STRING    ! На эту переменную ссылка осущ. так OuterGroup:Field2
InnerGroup  GROUP
Field1      LONG      ! На эту перем. ссылка. OuterGroup:InnerGroup:Field1
Field2      STRING    ! На эту переменную ссылка OuterGroup:InnerGroup:Field2
            END
            END
```

**Смотри также:** PRE, CLASS, переменные-указатели

## Зарезервированные слова

Приведенные далее ключевые слова являются зарезервированными и не могут использоваться в качестве меток ни в каких случаях.

ACCEPT	AND	BEGIN	BREAK
BY	CASE	CLASS	CHOOSE
COMPILE	CYCLE	DISPOSE	DO
ELSE	ELSIF	END	EXECUTE
EXIT	FUNCTION	GOTO	IF
INCLUDE	LOOP	MEMBER	NEW
NOT	OF	OLE	OMIT
OR	OROF	PROCEDURE	PROGRAM
RETURN	ROUTINE	SECTION	THEN
TIMES	TO	UNTIL	WHILE
XOR			

Ключевые слова, приведенные ниже, могут использоваться в качестве меток структур данных или исполняемых операторов. Но они не могут быть метками операторов PROCEDURE и FUNCTION. Их можно использовать в качестве имен параметров в прототипе, только если также указан тип данных параметра.

APPLICATION	CODE	DETAIL	FILE
FOOTER	FORM	GROUP	HEADER
ITEM	JOIN	MAP	MENU
MENUBAR	MODULE	OPTION	QUEUE
RECORD	REPORT	ROW	SHEET
TAB TABLE	TOOLBAR	VIEW	WINDOW

## Специальные символы

Список специальных символов, используемых в языке Clarion:

### Инициаторы

- ! Восклицательным знаком начинается комментарий
- ? Вопросительным знаком начинается метка соответствия поля или меню
- @ Знаком коммерческое “At” начинается шаблон.
- \* Звездочкой начинается имя параметра, передаваемого “по адресу”, в прототипе процедуры или функции в структуре MAP.

### Терминаторы

- ; Точка с запятой является разделителем исполняемых операторов.
- CR/LF Комбинация символов “возврат каретки”/” перевод строки” тоже является разделителем исполняемых операторов.
- . Точка завершает операторную структуру или структуру данных (заменитель оператора END).



- | Вертикальная черта является символом продолжения оператора
- # Знак фунта в конце имени означает неявное объявление переменной типа LONG.
- \$ Знак доллара в конце имени означает неявное объявление переменной типа REAL.
- “ Двойные кавычки в конце имени означают неявное объявление переменной типа STRING.

### Разделители

- ( ) В круглые скобки заключается список параметров.
- [ ] В квадратные скобки заключается список индексов массива.
- “ ” В одиночные кавычки заключается строковая константа.
- { } В фигурные скобки заключается коэффициент повторения символа в строковой константе.
- < > В угловые скобки заключается код ASCII символа в строковой константе или параметр в прототипе в структуре MAP, который может опускаться при обращении к процедуре или функции.
- : Двоеточие разделяет начало и конец “подстроки”.
- , Запятая разделяет параметры в списке.

### Коннекторы

- . Десятичная точка в числовых константах или точкой соединяется имя составной структуры с именем одного из ее компонентов.
- : Двоеточие соединяет префикс с меткой переменной или метку составной структуры с именем переменной внутри этой структуры.
- \$ Знак доллара соединяет метку окна или структуры REPORT с меткой соответствия объекта в операторе присвоения значения свойству объекта
- \_ Знак подчеркивания соединяет группы символов внутри метки или имени переменной.

### Операции

- + Символ плюс обозначает сложение.
- Символ минус обозначает вычитание.
- \* Звездочка обозначает умножение.
- / Слэш обозначает деление.
- % Знак процента обозначает взятие остатка от деления.
- ^ Операция возведения в степень.
- < Левая угловая скобка обозначает операцию “меньше”.
- > Правая угловая скобка обозначает операцию “больше”.
- = Знак равенства обозначает присвоение или равенство.
- ~ Тильда обозначает булеву операцию “логическое НЕ”.
- & Амперсанд обозначает конкатенацию.

## Формат программы

### PROGRAM (объявить программу)

```

PROGRAM
MAP
    прототипы
    [MODULE( )
        прототипы
    END ]
END
глобальные данные
CODE
    исполняемые операторы
    [RETURN]
    процедуры или функции
  
```

#### PROGRAM

Обязательный первый оператор в программном исходном модуле.

#### MAP

Обязательное объявление глобальных процедур и функций

прототипы

Объявляют процедуры и функции глобальные данные

Объявляются глобальные статические данные, которые могут использоваться всеми процедурами и функциями.

#### CODE

Начинает секцию исполняемых операторов исполняемые операторы

Исполняемые операторы программы

#### RETURN

Завершает выполнение программы. Возвращает управление операционной системе. процедуры или функции

Исходные коды процедур и функций в программном модуле

Оператор PROGRAM является первым оператором объявления в исходном модуле Clarion-программы. Ему могут предшествовать только строки комментария. При компиляции имя файла, содержащего оператор PROGRAM, используется в качестве имен объектного (.OBJ) и исполняемого (.EXE) файлов. Оператор PROGRAM может иметь метку, но компилятором она игнорируется.

Программа, в состав которой входят процедуры и/или функции, должна иметь структуру MAP. В ней объявляются прототипы процедур и/или функций. Любая процедура и/или функция, содержащаяся в отдельном исходном файле, должна быть объявлена в структуре MODULE внутри структуры MAP.

Данные, объявленные в программном модуле между ключевыми словами PROGRAM и CODE, являются глобальными статическими и могут быть доступны любой процедуре или функции в программе.

**Пример:**

```

PROGRAM !Примерное объявление программы
INCLUDE('EQUATES.CLW') !Вставить стандартные метки соответствия
MAP
CalcTemp PROCEDURE !Прототип процедуры
END
CODE
CalcTemp !Обращение к процедуре

CalcTemp PROCEDURE
Fahrenheit REAL(0) !Объявление глобальных данных
Centigrade REAL(0)

Window WINDOW('Temperature Conversion'),CENTER,SYSTEM
STRING('Enter Fahrenheit Temperature: '),AT(34,50,101,10)
ENTRY(@N-04),AT(138,49,60,12),USE(Fahrenheit)
STRING('Centigrade Temperature:'),AT(34,71,80,10),LEFT
ENTRY(@N-04),AT(138,70,60,12),USE(Centigrade),SKIP
BUTTON('Another'),AT(34,92,32,16),USE(?Another)
BUTTON('Exit'),AT(138,92,32,16),USE(?Exit)
END
CODE !Начало раздела исполняемых операторов
OPEN(Window)
ACCEPT
CASE ACCEPTED()
OF ?Fahrenheit
Centigrade = (Fahrenheit - 32) / 1.8
DISPLAY(?Centigrade)
OF ?Another
Fahrenheit = 0
Centigrade = 0
DISPLAY
SELECT(?Fahrenheit)
OF ?Exit
BREAK
END
END
CLOSE(Window)
RETURN

```

**Смотри также:** MAP, MODULE, PROCEDURE, FUNCTION, Объявление данных и распределение памяти.

## MEMBER (идентифицировать дополнительный исходный файл)

**MEMBER**([программа])

[MAP

    прототипы

.]

[метка]      локальные данные

    процедуры или функции

### MEMBER

Первый оператор в модуле исходного текста, который не является программным модулем. Обязательный оператор.

### программа

Текстовая константа, содержащая имя файла с исходной программой (без расширения). Если этот параметр опущен, то данный модуль является “универсальным модулем”, который можно компилировать в любую программу, добавив его в проект.

### MAP

Объявления локальных процедур и функций. К объявленным здесь процедурам и функциям можно обращаться только из процедур и функций данного member-модуля.

### прототипы

Объявления процедуры и функции.

### локальные данные

Объявляются локальные статические данные, на которые можно ссылаться только в процедурах и функциях, чей исходный текст содержится в данном member-модуле

### Процедуры или функции

Исходные тексты процедур и функций в программном модуле.

MEMBER стоит первым оператором в исходном модуле, который не является программным исходным файлом (не содержит оператора PROGRAM). Ему могут предшествовать только строки комментария. Оператор MEMBER требуется в начале любого исходного файла, который содержит процедуры или функции, используемые программой. Он идентифицирует программу, к которой относится исходный модуль.

Member-модуль может содержать локальную структуру MAP. Обращение к определенным в ней процедурам и функциям возможно только из процедур и функций данного member-модуля. Исходный текст процедур и функций, объявленных в структуре MAP, может содержаться в исходном файле member-модуля или другом исходном файле. Если исходный текст процедур и функций, объявленных в структуре MAP, содержится в отдельном файле, то их прототипы должны быть объявлены в структуре MODULE внутри этой структуры MAP. А тот отдельный исходный файл должен также содержать свою собственную структуру MAP, которая объявляет те же самые прототипы тех же процедур и функций. Любая процедура или функция, не объявленная в глобальной (программной) структуре MAP, должна быть объявлена в локальной MAP-структуре member-модуля, который содержит их исходный текст.

Данные, объявленные в member-модуле, между ключевыми словами MEMBER и PRO-

CEDURE или FUNCTION, являются локальными статическими и могут быть доступны процедурам и функциям данного модуля.

### Пример:

!Модуль Source1 содержит:

```
MEMBER('OrderSys') !Модуль входит в программу OrderSys
MAP                !Объявление локальных процедур
Func1              FUNCTION(STRING),STRING!Функция Func1 известна только в этих двух модулях
```

```
MODULE('Source2.clw')
HistOrd2 PROCEDURE !Процедура HistOrd2 известна только в этих двух модулях
END
END
```

```
LocalData STRING(10) !Локальные данные по отношению к member-модулю
HistOrd PROCEDURE    !Объявление процедуры HistOrd
HistData STRING(10)  !Объявляет данные локальные для этой процедуры
```

```
CODE
LocalData = Func1(HistData)
Func1 FUNCTION(RecField)      !Объявляет локальную функцию
CODE
!Исполняемые операторы
```

!Модуль Source2 содержит:

```
MEMBER('OrderSys') !Модуль относится к программе OrderSys
MAP                !Объявление локальных процедур
HistOrd2 PROCEDURE !HistOrd2 известна только в этих двух модулях
MODULE('Source1.clw')
```

```
Func1              FUNCTION(STRING),STRING
!Func1 известна только в этих двух модулях
END
END
```

```
LocalData STRING(10)
!Объявляет данные локальные по отношению к member-модулю
```

```
HistOrd2 PROCEDURE !Объявление процедуры HistOrd2
CODE
LocalData = Func1(LocalData)
```

**Смотри также:** MODULE, PROCEDURE, FUNCTION, Объявление данных и распределение памяти.

## MAP (объявить прототипы процедур и/или функций)

**MAP**

прототипы

**[MODULE()**

прототипы

**END]**

**END**

**MAP**

Содержит прототипы, которые объявляют процедуры, функции и внешние модули исходного текста, используемые в программном или member-модуле.

прототипы

Объявляют процедуры и функции

**MODULE()**

Объявляет member-модуль исходного текста.

Структура MAP содержит прототипы, которые объявляют функции, процедуры и внешние исходные модули используемые в программном или member-модуле, но которые не являются элементами структур CLASS. Структура MAP, находящаяся в программном модуле, объявляет процедуры и функции, доступные в любом месте программы. А структура MAP, находящаяся в member-модуле, объявляет процедуры и функции, которые доступны только в пределах данного member-модуля.

Структура MAP обязательна для любой программы на языке Clarion, потому что в нее компилятор автоматически вставляет модуль исходного текста BUILTINS.CLW. Он содержит прототипы большинства процедур и функций внутренней библиотеки Clarion, которые являются частью языка. Этот файл обязателен потому, что в самом компиляторе (в целях повышения эффективности) нет этих прототипов. Поскольку прототипы в файле BUILTINS.CLW используют те же самые метки соответствия констант, которые описаны в файле EQUATES.CLW этот файл также автоматически включается компилятором в каждую кларионовскую программу.

### Пример:

Один файл содержит:

PROGRAM !Пример программы в файле sample.cla

MAP !Начало объявлений в структуре MAP

LoadIt PROCEDURE !Процедура LoadIt

END !Конец структуры MAP

В отдельном файле содержится:

MEMBER('Sample') !Объявление member-модуля

MAP !Начало объявлений в локальной структуре MAP

ComputeIt PROCEDURE !Прототип процедуры ComputeIt

END !Конец структуры MAP

**Смотри также:** PROGRAM, MEMBER, MODULE, прототипы процедур и функций

**MODULE (указать исходный файл member-модуля)****MODULE**(файл исходного текста)

прототипы процедур

прототипы функций

**END****MODULE**                    Задает member-модуль или внешнюю библиотеку.

файл исходного текста            Строковая константа. Если исходный файл содержит текст на языке Clarion, то здесь указывается имя файла (без расширения), в котором находятся процедуры и/или функции. Если же исходный файл представляет собой внешнюю библиотеку, то эта строка содержит произвольный уникальный идентификатор.

прототипы процедур                Прототипы процедур, содержащихся в исходном файле.

прототипы функций                Прототипы функций, содержащиеся в исходном файле.

Структура MODULE именует member-модуль. Она содержит прототипы процедур и функций, содержащихся в исходном файле. Структура MODULE может объявляться только внутри структуры MAP.

**Пример:**

!Файл "sample.clw" содержит:

```
PROGRAM            !Пример программы в sample.clw
MAP                !Начало объявлений в структуре MAP
MODULE('Loadit')    ! исходный модуль loadit.clw
  LoadIt            ! процедура loadit
END                ! конец модуля
MODULE('Compute')   ! исходный модуль compute.clw
  ComputeIt !        некая процедура вычислений
END                ! конец модуля
END                !конец структуры MAP
```

!Файл "loadit.clw" содержит:

```
MEMBER('sample')    !Объявляет member-модуль
MAP                !Начинает локальную структуру MAP
MODULE('Process')    ! исходный модуль process.cla
ProcessIt PROCEDURE        ! некая процедура обработки
END                ! конец модуля
END                !конец локальной структуры MAP
```

**Смотри также:** MAMBER, MAP, прототипы процедур и функций

## PROCEDURE (определить процедуру)

метка	<b>PROCEDURE</b> (список параметров)
[метка]	локальные данные
	<b>CODE</b>
	операторы
	<b>[RETURN]</b>

<b>PROCEDURE</b>	Начинает секцию исходного текста, которую можно выполнить указав ее имя.
метка	Назначает процедуре имя. При определении метода в структуре CLASS может содержать имя класса присоединенное спереди к имени процедуры.
список параметров	Необязательный список имен, значения которых передаются в процедуру. Эти имена определяют локальные ссылки на передаваемые параметры. При определении метода в структуре CLASS может содержать имя класса как неявный первый параметр.
локальные данные	Объявляются локальные данные, доступные только в рамках этой процедуры.
<b>CODE</b>	Начало исполняемых операторов
операторы	Исполняемые операторы процедуры
<b>RETURN</b>	Завершение выполнения процедуры. Возврат в точку, из которой было обращение к процедуре

Оператор **PROCEDURE** начинает порцию исходного текста, выполнение которой может быть инициировано из любой точки программы. К ней обращаются просто указывая метку оператора **PROCEDURE** (и, если необходимо, параметры) в качестве исполняемого оператора в программной секции программы, процедуры или функции.

В списке параметров следом за именем (обязательным), под которым параметр используется внутри данной процедуры, возможно (но не обязательно) определяется тип данных параметра. Параметры отделяются запятой. Если процедура “перегружаемая” (имеет несколько определений), то наряду с меткой обязателен и тип данных каждого параметра (включая заключенные в угловые скобки, означающие возможность опускания параметра).

При выполнении оператора **RETURN** процедура заканчивается и возвращает управление в точку вызова. После последнего исполняемого оператора в процедуре (в конце программной секции) происходит выполнение неявного оператора **RETURN**. Конец программной секции процедуры определяется по концу файла исходного текста или появлению операторов **FUNCTION**, **ROUTINE** или следующего оператора **PROCEDURE**.

Данные, объявленные в процедуре в промежутке между операторами **PROCEDURE** и **CODE** являются локальными для процедуры, которые доступны только в этой процедуре



(за исключением случая передачи их как параметров в другую процедуру или функцию). Эти данные располагаются в памяти в момент входа в процедуру и освобождаются по ее завершении. Если данные меньше порогового для стека значения (по умолчанию 5К), то они помещаются в стек. В противном случае память для них выделяется из “кучи”.

Процедура должна иметь прототип, объявленный в структуре MAP или CLASS программного или member-модуля. В случае указания прототипа процедуры в программном модуле обращение к ней возможно из любых других процедур и функций данной программы. Если же процедура представлена прототипом в member-модуле, то обращение к ней возможно только из процедур и функций этого модуля.

### Пример:

```

PROGRAM          !Примерный программный код
MAP
OpenFile PROCEDURE(FILE AnyFile)      ! Прототип процедуры с параметрами
ShoTime  PROCEDURE !Procedure prototype without parameter
END
CODE
OpenFile(FileOne)  !Вызвать процедуру, чтобы открыть файл
ShoTime!вызвать процедуру ShoTime
:      !Другие исполняемые операторы
OpenFile PROCEDURE(FILE AnyFile)      ! Открывает любой файл
CODE !Начало программной секции
OPEN(AnyFile)      !Открыть файл
IF ERRORCODE() = 2      !Если файл не найден,
  CREATE(AnyFile) ! создать его
END  !Конец оператора IF
RETURN      !Возврат в вызвавшую процедуру или функцию

ShoTime PROCEDURE !Вывести время
Time  LONG      !Локальная переменная
Window WINDOW,CENTER
  STRING(@T3),USE(Time),AT(34,70)
  BUTTON('Exit'),AT(138,92),USE(?Exit)
END
CODE      !Начало программной секции
TIME = CLOCK() !Получить время от системы
OPEN(Window)
ACCEPT
CASE ACCEPTED()
  OF ?Exit
    BREAK
  END
END
RETURN      !Возврат в вызвавшую процедуру или функцию

```

**Смотри также:** прототипы процедур и функций, объявление данных и распределение памяти, перегрузка функций, CLASS.

## FUNCTION (определить функцию)

метка	<b>FUNCTION</b> (список параметров)
	локальные данные
	<b>CODE</b>
	операторы
	<b>RETURN</b> (значение)

<b>FUNCTION</b>	Начинает секцию исходного текста, которую можно выполнить указав ее имя.
метка	Именуется функция. При определении метода в классе может содержать имя класса присоединенное спереди к имени функции.
список параметров	Необязательный список имен, значения которых передаются в функцию. Эти имена определяют локальные ссылки на передаваемые параметры. При определении метода в структуре CLASS может содержать имя класса как неявный первый параметр.
локальные данные	Объявляются данные, доступные только в рамках этой функции.
<b>CODE</b>	Начало исполняемых операторов.
операторы	Исполняемые операторы функции.
<b>RETURN</b>	Завершение выполнения функции и возврат значения в выражение, в котором функция используется.
значение	Числовая или строковая константа или переменная, которая содержит результат обращения к функции.

Оператор FUNCTION начинает порцию исходного текста, выполнение которой может быть инициировано указанием метки этого оператора со списком параметров (если не передаются параметры, то круглые скобки тем не менее требуются). Выполнение функции заканчивается обязательным оператором RETURN.

В списке параметров следом за именем (обязательным), под которым параметр используется внутри данной функции, возможно (но не обязательно) определяется тип данных параметра. Параметры отделяются запятой. Если функция “перегружаемая” (имеет несколько определений), то наряду с меткой обязателен и тип данных каждого параметра (включая заключенные в угловые скобки, означающие возможность опускания параметра).

Функция может использоваться в выражениях или передаваться как параметр процедуры или другой функции. Функцию можно выполнить, обратившись к ней как к процедуре, если логика программы не подразумевает возврата значения функцией. В этом случае компилятор выдаст предупреждение (если ее прототип не содержит атрибута PROC), которое можно безболезненно проигнорировать.

Данные, объявленные внутри функции, в промежутке между операторами FUNCTION и CODE являются локальными динамическими, поэтому доступны только в этой функции (за исключением случая передачи их как параметров в другую процедуру или функцию). Эти данные располагаются в стеке, а память для них выделяется в момент входа в функцию и освобождается по ее завершении. Если объем данных меньше предельного для стека (по умолчанию 5K), то данные размещаются в стеке, в противном случае, они размещаются в куче (heap).

Функция должна иметь прототип, объявленный в структуре MAP или CLASS, программного или member-модуля. В случае указания прототипа функции в программном модуле обращение к ней возможно из любых других процедур и функций данной программы. Если же функция представлена прототипом в member-модуле, то обращение к ней возможно только из процедур и функций этого модуля.

### Пример:

```

PROGRAM
MAP
FullName  FUNCTION(String Last,String First,<String Init>),String  !Прототип
функции с параметрами
DayString FUNCTION,String          !Прототип функции без параметров
END
CODE
TodayString = DayString()
                !Обращение к функции без параметров для функции требуются скобки ( )
: !Глобальные исполняемые операторы
:
START(NewThread)
!кларноновская функция START вызывается как процедура
                !генерируется сообщение компилятора
                ! но выполняется корректно

FullName FUNCTION(String Last, String First,String Init)          ! Функция FullName
CODE !Начало программной секции
IF OMITTED(3) OR Init = ''          !Если нет среднего инициала
    RETURN(CLIP(First) & ' ' & Last)    !вернуть полное имя
ELSE !в противном случае
    RETURN(CLIP(First) & ' ' & Init & ' ' & Last)
    ! вернуть полное имя
END

DayString FUNCTION          !Функция день недели
CODE !Начало программной секции
Day# = (TODAY() % 7) + 1      !Определить день из системной даты
EXECUTE Day#                !По номеру возратить строкой день недели
RETURN('Sunday')
```

```

RETURN('Monday')
RETURN('Tuesday')
RETURN('Wednesday')
RETURN('Thursday')
RETURN('Friday')
RETURN('Saturday')

```

**Смотри также:** прототипы процедур и функций, объявление данных и распределение памяти, перегрузка функций, CLASS.

## CODE- (начало исполняемых операторов)

### CODE

Оператор **CODE** отделяет в программном модуле, процедуре и функции секцию объявления данных от секции исполняемых операторов (программной секции). За оператором CODE следует первый исполняемый оператор в программном модуле, процедуре или функции.

#### Пример:

```
PROGRAM                                !Здесь идет объявление глобальных данных
```

```
CODE                                  ! Здесь идут исполняемые операторы
```

```
OrdList PROCEDURE                    !Объявить процедуру
                                !Здесь идут операторы объявления локальных данных
CODE                                !Начало программной секции
                                !Здесь идут исполняемые операторы
```

**Смотри также:** операторы PROGRAM, PROCEDURE, FUNCTION

## ROUTINE (объявить локальную подпрограмму)

### метка ROUTINE

**ROUTINE** Объявляет начало локальной подпрограммы.  
метка Имя локальной подпрограммы

Оператор ROUTINE начинает локальную подпрограмму, состоящую из исполняемых операторов. Она локальна по отношению к процедуре или функции, в которой написана и должна находиться в конце ее программной секции. Все переменные, доступные в процедуре или функции доступны и локальной подпрограмме, включая данные локальные для процедуры и модуля и глобальные данные.

Обращение к локальной подпрограмме производится оператором DO, за которым следует метка локальной подпрограммы. После выполнения локальной подпрограммы управление передается оператору, следующему за оператором DO. Локальная подпрограмма

заканчивается с концом исходного файла, или с началом другой локальной подпрограммы, процедуры или функции. Для того, чтобы завершить выполнение локальной подпрограммы, может также использоваться оператор EXIT (подобно оператору RETURN в процедурах).

Внутренне локальная подпрограмма оформляется компилятором как локальная процедура в ассемблере. Поэтому нет вопросов, связанных эффективностью, которые не были бы самоочевидны:

- \* Операторы DO и EXIT очень эффективны.
- \* Обращение к данным, локальным для процедуры, менее эффективно, чем обращение к данным, локальным для модуля.
- \* Неявные переменные, используемые только в подпрограмме, менее эффективны, чем локальные переменные.
- \* Каждый оператор RETURN в локальной подпрограмме реализуется 40 байтами.

### Пример:

```
SomeProc PRDCEDURE
CODE
    !Исполняемые операторы
DO Tally          !Вызвать локальную подпрограмму
    !Исполняемые операторы
Tally ROUTINE     !Конец процедуры, начало локальной подпрограммы
IF CountVar < 55   !Если меньше 55
    CountVar += 1  ! увеличить на единицу
ELSE              !Иначе
    CountVar = 0   ! сбросить счетчик
EXIT              !
END
```

**Смотри также:** операторы EXIT, DO

### END (закончить структуру)

**END**

Оператор END заканчивает объявление структуры данных или составной исполняемый оператор. Функционально эквивалентен точке (.).

Принято оператор END выравнивать на ту же колонку, в которой начинается завершаемая им структура, а операторы структуры пишутся с отступом для улучшения читаемости. Обычно, оператор END используется для завершения структур состоящих из нескольких строк, тогда как точка - для структур занимающих одну строку. Если имеется вложенные исполняемые структуры и все они заканчиваются в одном месте, то вместо нескольких операторов END на нескольких строках используются несколько точек на одной

строке.

### Пример:

```
Customer  FILE,DRIVER('Clarion')!Объявить файл
          RECORD      ! начало объявления записи
Hanle     STRING(20)
Number    LONG
          END          !Конец структуры записи
          END          !Конец объявления файла
Archive   FILE,DRIVER('Clarion')      ! Объявить файл
          RECORD          !Начало объявления структуры записи
Name      STRING(20)
Number    LONG
          ..            !конец объявления и структуры RECORD и структуры FILE
          CODE
          IF Number <> SavNumber
            DO GetNummber
          END
          IF SomeCondition THEN BREAK.    !Заканчивается точкой
          CASE Action
          OF 1
            DO AddRec
            IF Number <> SavNumber      !Начало структуры IF
              DO SomeRoutine
            END                        !Конец структуры IF
          OF 2
            DO ChgRec
          OF 3
            DO DelRec
          END
```

## Последовательность выполнения операторов

---

В секции исполняемых операторов программы на Clarion операторы обычно выполняются по одному в последовательности их расположения в исходном тексте. Для того, чтобы изменить последовательность выполнения операторов используются управляющие операторы и обращения к процедурам и функциям.

Обращение к процедуре изменяет последовательность выполнения программы посредством перехода к вызываемой процедуре и выполнения содержащихся в ней исполняемых операторов. Когда в вызванной процедуре выполняется оператор RETURN или в ней выполнен последний оператор, управление возвращается оператору, следующему за обращением к процедуре.

Обращение к функции изменяет последовательность выполнения программы

посредством перехода к вызываемой функции и выполнения содержащихся в ней исполняемых операторов. Когда в вызванной функции выполняется оператор RETURN, возвращая ее значение, управление передается оператору, содержащему обращение к функции.

Управляющие структуры IF, CASE, LOOP и EXECUTE изменяют последовательность выполнения, основываясь на вычислении выражений. Когда значение выражения вычислено, управляющая структура в зависимости от условий выполняет содержащиеся в ней операторы.

Кроме того, ветвление происходит при выполнении операторов GOTO, DO, CYCLE, BREAK, EXIT, RETURN и RESTART. Выполнение этих операторов немедленно и безусловно изменяет нормальную последовательность выполнения программы. Функция START начинает новый процесс выполнения, безусловно переключая выполнение на этот процесс. Однако, щелкнув мышью в активном окне другого процесса, пользователь может активизировать выполнение этого другого процесса.

### Пример:

```

PROGRAM
MAP
ComputeTime      PROCEDURE(*GROUP)      !Передача группы в качестве параметра
MatchMaster      PROCEDURE              !Не передается никаких параметров
END

ParmGroup GROUP   !Объявить группу
FieldOne  STRING(10)
FieldTwo  LONG
END
CODE           !Начало исполняемых операторов
FieldTwo = CLOCK()      !Выполнить 1-й
ComputeTime(ParmGroup)  !Выполнить 2-й, передав управление в процедуру
MatchMaster           !Выполнить после того, как выполнится процедура
ComputeTime
    
```

### Обращение к процедурам и функциям

```

Имя_процедуры[(параметры)]
переменная = имя_функции[(параметры)]
    
```

Имя\_процедуры      Имя процедуры как оно объявлено в прототипе в структуре MAP.  
 параметры      Необязательный список параметров, передаваемых процедуре или функции. Список параметров может представлять из себя одну или несколько переменных или выражений. Параметры разделяются запятыми и объявляются в прототипе.  
 переменная      Имя переменной, в которую заносится возвращаемое функцией

значение.

имя\_функции           Имя функции как оно объявлено в прототипе в структуре MAP.

Процедура вызывается указанием ее имени (с возможным списком параметров) в качестве оператора в исполняемой части программы, процедуры или функции. Список параметров должен соответствовать списку, объявленному в прототипе процедуры. К процедуре нельзя обращаться в выражении. Если процедура является методом какого-либо класса, то имя процедуры должно состоять из имени класса, за которым через точку следует имя собственно процедуры (имя\_класса.имя\_процедуры).

Функция вызывается указанием ее имени (с возможным списком параметров) в качестве компонента выражения или в списке параметров передаваемых в другую процедуру или функцию. Список параметров должен соответствовать списку, объявленному в прототипе. Кроме того, если, возвращаемое функцией значение не играет роли, то к ней можно обратиться указанием ее имени (с возможным списком параметров) таким же образом как к процедуре. На такое использование функции компилятор выдаст предупреждение, которое можно безболезненно проигнорировать. Если функция является методом какого-либо класса, то имя функции должно состоять из имени класса, за которым через точку следует имя собственно функции (имя\_класса.имя\_функции)

### Пример:

```

PROGRAM
MAP
ComputeTime      PROCEDURE(*GROUP)      !Передача группы в качестве параметра
MatchMaster      FUNCTION,BYTE,PROC      !Функции не передается параметров

END
ParmGroup GROUP      !Объявление группы
FieldOne  STRING(10)
FieldTwo  LONG
END
CODE
FieldTwo = CLOCK()
!В качестве выражения используется встроенная функция
ComputeTime(ParmGroup)      !Вызов процедуры ComputeTime
MatchMaster()                !Обращение к функции как к процедуре

```



## Прототипы процедур и функций

### Синтаксис прототипов

имя **PROCEDURE** [(список параметров)] [,соглашение о передаче параметров] [,**RAW**] [,**NAME( )**] [,**TYPE**] [,**DLL( )**] [,**PRIVATE**] [,**VIRTUAL**]

имя **FUNCTION** [(список параметров)] [,тип возвращаемого значения] [,соглашение о передаче параметров] [,**RAW**] [,**NAME( )**] [,**TYPE**] [,**DLL( )**] [,**PROC**] [,**PRIVATE**] [,**VIRTUAL**]

name[(список параметров)] [,тип возвращаемого значения] [,соглашение о передаче параметров] [,**RAW**] [,**NAME( )**] [,**TYPE**] [,**DLL( )**] [, **PROC**] [, **PRIVATE**]

имя	Метка оператора <b>PROCEDURE</b> или <b>FUNCTION</b> , который определяет начало секции исполняемых операторов.
<b>PROCEDURE</b>	Указывает что никаких данных не возвращается.
<b>FUNCTION</b>	Указывает что ланные возвращаются и требует наличия типа возвращаемого значения.
список параметров	Типы данных параметров, передаваемых в процедуру или функцию. За типом данных каждого параметра может следовать метка, используемая для документирования параметра (и только). Каждый параметр может включать присвоение значения, используемого по умолчанию (константы), которое передается, если параметр опущен.
тип возвращаемого значения	Тип возвращаемого функцией значения.
соглашение о передаче параметров	Задает соглашение о передаче параметров через стек по типу языка <b>C</b> или <b>Паскаль</b> .
<b>RAW</b>	Указывает, что для параметров типа <b>STRING</b> или <b>GROUP</b> передается только их адрес в памяти (без длины передаваемой строки). Кроме того, этот атрибут изменяет поведение параметров со <b>?</b> и <b>*?</b> спереди. Этот атрибут служит только для совместимости с функциями на языках 3-го поколения и не действует для процедур, написанных на языке <b>Clarion</b> .
<b>NAME()</b>	Задает альтернативное, “внешнее” имя процедуры или функции.
<b>TYPE</b>	Указывает, что прототип является определением типа процедуры, передаваемой в качестве параметра.
<b>DLL</b>	Указывает, данная процедура или функция находится во внешней библиотеке <b>DLL</b> .
<b>PROC</b>	Задает, что обращение к данной функции как к процедуре не вызывает предупреждающего сообщения компилятора.
<b>PRIVATE</b>	Указывает, что к данной процедуре или функции можно обращаться

из других процедур и функций, расположенных только в том же самом программном модуле (обычно используется в классах).

**VIRTUAL**                      Указывает, что данная процедура или функция представляет собой виртуальный метод структуры **CLASS**.

Все процедуры и функции в программе должны быть представлены прототипами в структуре **MAP** или **CLASS**. Прототип точно говорит компилятору, какую форму обращения к процедуре или функции ожидать среди исполняемых операторов.

На предыдущей странице перечислены три допустимые формы объявления прототипов. Первые две, использующие ключевые слова **PROCEDURE** и **FUNCTION**, допустимы для использования в любом месте, и являются предпочтительными. Третья форма пока поддерживается только для совместимости с предыдущими версиями Clarion.

Прототип состоит:

- \* из имени процедуры или функции.
- \* ключевого слова **PROCEDURE** или **FUNCTION**, которое может отсутствовать в структуре **MAP**, но которое обязательно в структуре **CLASS**;
- \* необязательного списка параметров, задающего все передаваемые параметры;
- \* типа возвращаемого функцией значения ;
- \* соглашения о передаче параметров, если предстоит компоновка объектов, подразумевающих передачу параметров, через стек (объекты компилированные не TopSpeed компилятором).
- \* и по мере необходимости атрибутов **RAW**, **NAME**, **TYPE**, **DLL**, **PROC**, **PRIVATE**, и **VIRTUAL**.

Дополнительно можно указать для процедуры или функции соглашение о передаче параметров через стек по типу языка **C** (справа налево) или **Паскаль** (слева направо). Это обеспечивает совместимость с библиотеками третьих фирм, написанными на других языках. Если соглашение о передаче параметров не указано, то по умолчанию используется передача параметров в регистрах, принятая в языках семейства TopSpeed.

Независимо от того, передаются ли параметры-значения или параметры-переменные, атрибут **RAW** позволяет передать просто адрес памяти (\*?) для строки или группы в процедуру или функцию не на языке Clarion. Обычно для строк и групп передаются адрес и длина строки или группы. Указание атрибута **RAW** исключает передачу длины. Это наиболее полезно при использовании внешних библиотечных функций, которые принимают только адрес строки.

Атрибут **NAME** обеспечивает процедуре или функции внешнее, альтернативное имя. Это также делается для обеспечения совместимости с написанными на других языках библиотеками третьих фирм. Например, в некоторых компиляторах языка **C**, наряду с



```
Func50 FUNCTION(SREAL),REAL,PASCAL,DLL
      END
END
```

**Смотри также:** MAP, MEMBER, MODULE, NAME, PROCEDURE, FUNCTION, RETURN, Списки параметров в прототипе, Перегрузка функций

## Списки параметров в прототипах

Список параметров представляет собой разделенный запятыми список типов данных, передаваемых в процедуру или функцию. Весь свисок параметров, заключенный в скобки, следует за ключевым словом PROCEDURE или FUNCTION. За любым типом данных через пробел может следовать соответствующая синтаксису языка метка параметра (которая игнорируется компилятором и служит только для документирования параметра). Определение любого числового параметра (передаваемого “по значению”) может, кроме того, содержать присвоение числовой константы обозначению типа данных или метке параметра, если таковая имеется. Этим определяется значение по умолчанию, передаваемое в том случае, когда данный параметр опущен.

Если значение, передаваемое по умолчанию не определено, то параметр, который может опускаться при обращении к процедуре или функции, в списке параметров в прототипе должен быть заключен в угловые скобки (<>). Определить во время выполнения программы факт опускания параметра можно с помощью функции OMITTED (за исключением тех опущенных параметров, для которых определено значение, передаваемое по умолчанию).

### Параметры-значения

Параметры-значения являются “передаваемыми по значению”. В вызываемой процедуре или функции используются копии переменных, передаваемые вызывающей процедурой или функцией в списке параметров. Вызываемая процедура или функция не может изменить значение переданной ей переменной в вызывающей процедуре или функции. К параметрам-значениям применяются обычные правила преобразования данных. Реально передаваемые параметры-значения преобразуются к типу данных, указанному в прототипе данной процедуры или функции. Допустимые типы параметров-значений:

BYTE SHORT USHORT LONG ULONG SREAL REAL DATE TIME STRING

### **Пример:**

```
MAP
MODULE('Test')
MyProc1 PROCEDURE(LONG)           !Параметр-значение типа LONG
MyProc2 PROCEDURE(<LONG>)         ! Параметр-значение типа LONG, который может
опускаться
MyProc3 PROCEDURE(LONG=23)        !Если параметр опущен, то передать 23

MyProc4 PROCEDURE(LONG Count, REAL Sum) !Параматр типа LONG имеет имя Count, а
! параметр типа REAL имеет имя Sum
```

```

MyProc5 PROCEDURE(LONG Count=1, REAL Sum=0)
    !Значение по умолчанию параметра Count равно 1а параметра Sum равно 0
END
MODULE('Party3.Obj')
Func48 FUNCTION(REAL),REAL,PASCAL !передача параметров как в Паскале
Func49 FUNCTION(SREAL),REAL,C,NAME('_func49')
    !передача параметров как в C и внешнее имя
END
MODULE('STDFuncs.DLL') ! DLL-библиотека стандартных функций
Func50 FUNCTION(SREAL),REAL,PASCAL,DLL
END
END

```

### Параметры-переменные

Параметры-переменные являются “передаваемыми посредством адреса”. Переменные, передаваемые посредством адреса имеют в памяти только одно местоположение. Изменение значения переменной в вызываемой процедуре или функции изменяет его также и в вызывающей. Типы данных параметров-переменных приводятся в списке параметров прототипа процедуры или функции в структуре MAP со звездочкой (\*) перед названием типа. Допустимые параметры-переменные:

```

*BYTE *SHORT *USHORT *LONG *ULONG *SREAL *REAL *BFLOAT4 *BFLOAT8
*DECIMAL *PDECIMAL *DATE *TIME *STRING *PSTRING *CSTRING *GROUP

```

### Пример:

```

MAP
MODULE('Test')
MyProc2 PROCEDURE(<*LONG>) !Параметр-переменная типа LONG
MyFunc1 FUNCTION(*SREAL),REAL,C !Параметр-переменная типа SREAL,
    ! возвращается REAL,
    !передача параметров как в C
END
MODULE('Party3.Obj')
Func4 FUNCTION(*CSTRING),REAL,C,RAW !В функцию на C передается
    !только адрес строки
    !типа CSTRING
Func47 FUNCTION(*CSTRING),CSTRING,C,RAW !возвращает указатель на
CSTRING
END
END

```

### Передача массивов в качестве параметров

Чтобы передать в качестве параметра массив, в прототипе должен быть объявлен тип данных массива как параметр-переменная (передаваемая посредством адреса), с пустым

списком индексов. Если массив более чем одномерный, то чтобы обозначить число измерений в нем, в списке индексов запятыми разделяются позиции. При обращении в процедуру или функцию должен передаваться целый массив, а не один элемент.

### Пример:

```

PROGRAM
MAP
MainProc  PROCEDURE
  AddCount PROCEDURE(*LONG[,],*LONG[,])
!Передача двух двумерных массивов типа LONG
  END
CODE
MainProc  !Вызов первой процедуры

MainProc PROCEDURE
TotalCount LONG,DIM(10,10)
CurrentCnt  LONG,DIM(10,10)
CODE
  AddCount(TotalCount,CurrentCnt) !Обратиться к процедуре передав массив
AddCount  PROCEDURE(*LONG[,] Total,*LONG[,] Current)
!Процедура ожидает передачи двух массивов
CODE
  LOOP I# = 1 TO MAXIMUM(Total,1)      !Цикл по первому индексу
    LOOP J# = 1 TO MAXIMUM(Total,2)    !Цикл по второму индексу
      Total[I#,J#] += Cur[I#,J#]      ! увеличить TotalCount
    .
  .
  CLEAR(Cur)!Очистить массив CurrentCnt

```

**Смотри также:** прототипы процедур и функций, MAXIMUM

### Передача параметров с неопределенным типом данных

Используя нетипизированные параметры-значения и нетипизированные параметры-переменные, можно писать обобщенные функции, которые выполняют действия над переданными им параметрами в ситуации, когда точный тип параметра может изменяться от одного обращения к функции к другому. Это полиморфные параметры; в зависимости от типа передаваемых в процедуру данных они могут приобретать любой другой простой тип данных.

Нетипизированные параметры-значения представляются в прототипе процедуры или функции знаком вопроса (?). При выполнении процедуры параметр динамически обретает тип и действует как объект данных базового типа передаваемой переменной (LONG, DECIMAL, STRING, or REAL) или переменной, которой он последний раз был присвоен. Это означает, что этот “подразумеваемый” тип данных параметра можно изменить внутри

процедуры или функции, что позволяет его рассматривать как параметр любого типа данных.

Нетипизированные параметры-значения передаются в процедуру или функцию “по значению”, а его “подразумеваемый” тип данных подчиняется правилам преобразования данных, принятым в Clarion.

Как нетипизированные параметры-значения могут передаваться переменные следующих типов данных:

BYTE SHORT USHORT LONG ULONG SREAL REAL BFLOAT4 BFLOAT8  
DECIMAL PDECIMAL DATE TIME STRING PSTRING CSTRING GROUP

(рассматриваемая как строка) нетипизированный параметр-значение (?) нетипизированный параметр-переменная (\*?)

Если нетипизированный параметр-значение передается во внешнюю библиотечную функцию, написанную не на языке Clarion, то можно использовать для него атрибут RAW. Это вызовет преобразование данных к типу LONG и затем передачу данных как параметр “void \*” в С или С++ (что исключает появление предупреждения о “несовместимости типов”).

Нетипизированные параметры-переменные представляются в прототипе процедуры или функции звездочкой и знаком вопроса (\*?). Внутри процедуры параметр с типом переменной действует как объект такого же типа данных, что и переменная в вызывающей процедуре, чье значение передается данным параметром. Это означает, что во время выполнения процедуры или функции тип данных параметра не изменяется.

Нетипизированные параметры-переменные передаются в процедуру или функцию “посредством адреса”. Таким образом любые изменения, выполненные над переданным параметром, производятся непосредственно над переменной из вызывающей процедуры, использовавшейся в качестве параметра. Параметры с неопределенным типом позволяют писать по-настоящему полиморфные процедуры и функции.

Внутри процедуры или функции, которая получает параметры с типом переменной, делать какие-либо предположения о типе данных принятого параметра небезопасно. Опасность таких предположения заключается в том, что существует вероятность присвоения значения, выходящего за границы допустимого для переданного параметра диапазона. Если это происходит, то результат может существенно отличаться от ожидаемого.

В качестве нетипизированных параметров-переменных могут передаваться данные следующих типов:

BYTE SHORT USHORT LONG ULONG SREAL REAL BFLOAT4 BFLOAT8 DECIMAL PDECIMAL DATE TIME STRING PSTRING CSTRING нетипизированный параметр-переменная (\*?)

Если нетипизированный параметр-переменная (\*?) передается во внешнюю библиотечную функцию, написанную не на языке Clarion, то можно указать атрибут RAW. В таком случае это эквивалентно передаче параметра “void\*” в С или С++.

Массив нельзя передавать ни в одной из форм параметров с неопределенным типом.

### Пример:

```

PROGRAM
MAP
Proc1      PROCEDURE(?)      !Нетипизированный параметр-значение
Proc2      PROCEDURE(*?)     !Нетипизированный параметр-переменная
Proc3      PROCEDURE(*?)     !Нетипизированный параметр-переменная
Max        FUNCTION(?,?,?)   !Функция, возвращающая нетипизированный
                                !параметр-переменную
END
GlobalVar1 BYTE(3)           !Начальное значение = 3
GlobalVar2 DECIMAL(8,2)
GlobalVar3 DECIMAL(8,1)
MaxInteger LONG
MaxString  STRING(255)
MaxFloat   REAL
CODE
PROC1(GlobalVar1)             !Передать байт равный 3
PROC2(GlobalVar2)             !Передать в процедуру DECIMAL(8,2) и она напечатает 3.33
PROC2(GlobalVar3)             !Передать в процедуру DECIMAL(8,1) и она напечатает 3.3
PROC3(GlobalVar1)             !Передать в процедуру байт и наблюдать как она зациклится
MaxInteger = Max(1,5)          !Функция Max возвращает 5
MaxString = Max('Z','A')       !Функция Max возвращает 'Z'
MaxFloat = Max(1.3,1.25)       !Функция Max возвращает 1.3
Proc1      PROCEDURE(ValueParm)
CODE          !ValueParm начинается с 3 и является
ValueParm = ValueParm & ValueParm ! теперь содержит 33 (ValueParm - строка)
ValueParm = ValueParm / 10 ! теперь содержит 3.3 (ValueParm - REAL)
Proc2      PROCEDURE(VariableParm)
CODE
VariableParm = 10/3             !Присвоить 3.333333... переданной переменной
Proc3      PROCEDURE(VariableParm)
CODE
LOOP
  IF VariableParm > 250 THEN BREAK. !Если передана переменная типа BYTE, то цикл
бесконечный
  VariableParm += 10
END
Max        FUNCTION(Val1,Val1) !Найти большее из двух переданных значений
CODE
  IF Val1 > Val2                 !Сравнить первое значение со вторым
  RETURN(Val1)                  ! и вернуть его, если оно больше

```



```
ELSE      !иначе  
RETURN(Val2)    ! вернуть второе  
END
```

### Параметры-объекты

Параметры-объекты передают имя структуры данных в вызываемую процедуру или функцию. Передача объекта позволяет вызываемой процедуре или функции использовать те команды Clarion, которые требуют указания метки структуры в качестве параметра. В прототипе процедуры или функции в структуре MAP параметры-объекты указываются именем объекта. Параметры-объекты всегда “передаются по адресу”. Допустимые параметры-объекты:

FILE VIEW KEY INDEX QUEUE WINDOW REPORT BLOB

#### **Пример:**

```
MAP  
MODULE('Test')  
MyFunc2    FUNCTION(FILE),STRING    !Параметр-объект FILE, возвращается строка  
ProcType    PROCEDURE(FILE),TYPE    !Определение типа параметра-процедуры  
MyFunc4    FUNCTION(FILE),STRING,PROC  
!Можно обращаться как к процедуре (не будет предупр.)  
MyProc6    PROCEDURE(FILE),PRIVATE  
!Может вызываться другими процедурами только модуля TEST.CLW  
END  
END
```

### Параметры-процедуры

Параметры-процедуры передают в вызываемую процедуру или функцию имя другой процедуры или функции. В прототипе процедуры или функции в структуре MAP параметры-процедуры представляются именем ранее объявленного прототипа такого же типа, который может иметь атрибут TYPE, а может и не иметь. При обращении в программе к вызываемой процедуре ей должно передаваться имя процедуры или функции, которая имеет точно такой прототип какой имеет процедура, указанная в прототипе вызываемой процедуры.

За каждым параметром в списке может следовать допустимая в языке Clarion метка, которая совершенно игнорируется компилятором. Эта метка используется исключительно

в целях документирования параметров, чтобы сделать прототип более удобочитаемым.

Каждое определение передаваемого параметра может также включать присвоение константного значения типу данных (или документирующей метке, если она есть), которое задает значение, используемое по умолчанию, которое передается, когда параметр опущен.

**Пример:**

```
MAP
  MODULE('Test')
ProcType  PROCEDURE(FILE),TYPE
! Определение типа параметра-процедуры
MyFunc3   FUNCTION(ProcType),STRING
!Процедура-параметр типа ProcType, возвращает STRING,
!должна передаваться процедура, принимающая объект FILE
!в качестве параметра
      END
END
```

**Передача “поименованных” структур GROUP, QUEUE и CLASS**

Передача структуры GROUP как параметра-переменной или QUEUE как параметра-объекта в процедуру или функцию не позволяет в вызванной процедуре обращаться к полям - компонентам этих структур. Однако, чтобы передавать адрес параметра и обеспечить доступ к полям - компонентам этих структур, можно указать в списке параметров прототипа метку структуры GROUP или QUEUE. Таким же образом можно “поименовать” класс, чтобы обеспечить принимающей процедуре возможность доступа к элементам данных и методам класса.

Для того, чтобы обращаться к составляющим структуру полям поместите метку структуры GROUP, QUEUE, или CLASS в список параметров прототипа. Таким образом параметр будет передаваться “посредством адреса” и в вызванной процедуре можно будет обращаться к отдельным полям. Структура передаваемых данных всегда должна совпадать с определением так же как и типы данных составляющих полей. Передаваемая группа или очередь может быть “надмножеством” поименованного параметра, если только идущие спереди поля совпадают соответствующими полями поименованной структуры GROUP или QUEUE. Реально передаваемый класс также может быть производным от поименованного класса.

Для структур GROUP, QUEUE или CLASS, указанных именем в списке параметров атрибут TYPE не нужен и их не нужно объявлять раньше чем структуру MAP. Однако они должны быть объявлены раньше, чем процедура или функция будет принимать их в качестве параметров при вызове. Это единственный случай в языке Clarion, когда разрешается такая “ссылка вперед”.

В операторе PROCEDURE или FUNCTION может объявляться локальное имя передаваемой группы с префиксом (Pre:LocalName), предупреждающим совпадение имен, однако это является устаревшим подходом, поскольку более предпочтительно использование синтаксиса уточнения имени переменной (LocalName.MemberName). К компонентам структуры обращаются по именам, присвоенным им в “поименованной” группе - а не по именам полей в передаваемой структуре. Это позволяет написать обобщенную процедуру, независимую от того, какая структура данных реально будет в нее передаваться.

### Пример:

```

        PROGRAM
        MAP
MyProc   PROCEDURE
AddQue   PROCEDURE(PassGroup,NameQue)

!Принимает GROUP описанную так же как PassGroup и QUEUE
! определенную так же как NameQue
        END
PassGroup GROUP,PRE(Pas),TYPE
!Определение типа: группа с 2-мя полями STRING(20)
F1        STRING(20)
F2        STRING(20)
        END
NameGroup GROUP,PRE(Nme)      !Группа Name
First     STRING(20)
Last      STRING(20)
Company   STRING(30)
        END
NameQue   QUEUE,PRE(Que)      !Name Queue
First     STRING(20)
Last      STRING(20)
        END
        CODE
        MyProc

MyProc   PROCEDURE
LocalQue QUEUE      !Локальная структура Name Queue
FirstName STRING(20)
LastName  STRING(20)
        END
        CODE
NameGroup.First = 'Fred'
NameGroup.Last = 'Flintstone'
AddQue(NameGroup,LocalQue)  !Передать NameGroup и LocalQue в AddQue

```

```

        NameGroup.First = 'Barney'
        NameGroup.Last = 'Rubble'
        AddQue(NameGroup,LocalQue)
        NameGroup.First = 'George'
        NameGroup.Last = 'O''Jungle'
        AddQue(NameGroup,LocalQue)
        LOOP X# = 1 TO RECORDS(LocalQue)           !Посмотреть, что теперь в
LocalQue
            GET(LocalQue,X#)
            MESSAGE(CLIP(LocalQue.FirstName) & ' ' & LocalQue.LastName)
        END

AddQue    PROCEDURE(PassedGroup,PassedQue)
CODE
    PassedQue.First = PassedGroup.F1
    PassedQue.Last  = PassedGroup.F2
    ADD(PassedQue)
!Добавить элемент в очередь PassedQue (LocalQue)
    IF ERRORCODE() THEN STOP(ERROR()).

```

## Типы значений, возвращаемых функциями

Функция должна возвращать значение. Тип возвращаемого значения приводится после необязательного списка параметров, отделенный от него запятой. Допустимые типы возвращаемых значений:

BYTE SHORT USHORT LONG ULONG SREAL REAL DATE TIME STRING  
 CSTRING \*BYTE \*SHORT \*USHORT \*LONG \*ULONG \*SREAL \*REAL \*DATE  
 \*TIME нетипизированное возвращаемое значение (?)

Нетипизированное возвращаемое значение (?) означает, что тип данных возвращаемого функцией значения неизвестен. Нетипизированное возвращаемое значение работает точно таким же образом как и нетипизированный параметр-значение. Когда возвращаемое функцией значение подпадает под стандартные правила преобразования типов, то не имеет значения какого типа данные вернула функция.

В прототипе функций, которые возвращают указатель (адрес данных), типу возвращаемого значения (за исключением CSTRING) должна предшествовать звездочка (\*). Это сделано только для совместимости с внешними библиотечными функциями (написанными на других языках), которые возвращают только адрес данных. Во время выполнения программы необходимые действия с возвращенным указателем выполняются автоматически. Функции, прототипированные таким способом, действуют точно так же как определенные в программе переменные - когда в тексте Clarion-программы используется такая функция, автоматически используются данные, адрес которых возвращен функцией.

Эти данные можно присвоить другим переменным, передать в качестве параметров процедурам и функциям, или получить их адрес с помощью функции ADDRESS.

В качестве примера предположим, что функция XYZ() возвращает \*CSTRING (указатель на CSTRING), переменная CStringVar имеет тип CSTRING, а переменная LongVar имеет тип LONG. Простой оператор присваивания языка Clarion: CStringVar = XYZ() поместит данные, указанные адресом, который возвратит функция XYZ() в переменную CStringVar. Оператор LongVar = ADDRESS(XYZ()), поместит адрес данных в переменную LongVar.

Тип данных CSTRING является исключением, так как все другие типы данных имеют фиксированную длину, а CSTRING - нет. Поэтому, любую функцию на С, возвращающую указатель на CSTRING, можно прототипировать в С как "char \*", но компилятор копирует элемент данных в стек. Поэтому, точно также как в случае других значений, возвращаемых посредством указателей, при использовании функции в тексте программы на языке Clarion данные, на которые указывает возвращенный функцией указатель используются автоматически (указатель разыменовывается).

### Пример:

MAP

MODULE('Party3.Obj') !Библиотека сторонней фирмы

Func46 FUNCTION(\*CSTRING),REAL,C,RAW !В функцию на С передается только адрес данных,

! возвращается Real

Func47 FUNCTION(\*CSTRING),\*CSTRING,C,RAW !Возвращается указатель на CSTRING

Func48 FUNCTION(REAL),REAL,PASCAL

!Передача параметров по типу PASCAL, возвращается Real

Func49 FUNCTION(SREAL),REAL,C,NAME('\_func49')

!Передача параметров по типу С и внешнее имя функции, возвращается Real

END

END

**Смотри также:** MAP, MEMBER, MODULE, NAME, FUNCTION, RETURN

## С, PASCAL (соглашения о связях)

С

PASCAL

Атрибуты С и PASCAL в прототипе процедуры или функции указывают, что параметры всегда передается через стек. В языке С параметры помещаются в стек справа налево по списку параметров, в то время как в Паскале наоборот, слева направо по списку. Паскаль также полностью совместим с соглашениями о связях интерфейса прикладного программирования (API) Windows и для 16-ти разрядных и 32-х разрядных приложений,

эти соглашения стали соглашениями используемыми по умолчанию операционной системой. Соглашения о связях языка С и Паскаль обеспечивают совместимость с библиотеками сторонних фирм, написанных не на языке Clarion (если они компилировались компиляторами TopSpeed). Если в прототипе не указать тип соглашений о связях, то будет использоваться внутреннее, принятое в компиляторах семейства TopSpeed соглашение, основанное на передаче параметров в регистрах.

### Пример:

MAP

MODULE('Party3.Obj') !Библиотека сторонней фирмы

Func46 FUNCTION(\*CSTRING),REAL,C,RAW !Передавать в функцию написанную на С только адрес CSTRING

Func49 FUNCTION(\*CSTRING,\*REAL),REAL,PASCAL,RAW ! Передавать CSTRING, затем REAL, только адрес

..

**Смотри также:** Прототипы процедур и функций, Списки параметров в прототипах

## RAW (передавать только адрес)

### RAW

Атрибут RAW задает в прототипе процедуры или функции для передаваемых в качестве параметров групп и строк передачу только их адреса памяти. Он позволяет для параметров типа \*?, STRING, или GROUP передавать в процедуру или функцию, написанную не на языке Clarion, только адрес памяти, независимо от того передаются ли параметры адресом или значением. Обычно, для параметров тип STRING, или GROUP передается адрес и длина строки. Указание атрибута RAW исключает передачу длины. В случае прототипа с параметром неопределенного типа параметр воспринимается как LONG, но передается как "void \*", что исключает появление предупреждений компоновщика. Этот атрибут введен для того, чтобы обеспечить совместимость с внешними библиотечными функциями, которые при вызове ожидают передачи только адреса строки.

### Пример:

MAP

MODULE('Party3.Obj') ! Библиотека сторонней фирмы

Func46 FUNCTION(\*CSTRING),REAL,C,RAW

! Передавать в функцию написанную на С только адрес CSTRING

..

**Смотри также:** Прототипы процедур и функций, Списки параметров в прототипах

## NAME (установить внешнее имя для прототипируемой процедуры)

### NAME( константа )

**NAME**            Указать компоновщику "внешнее" имя

константа      Строковая константа. Регистр букв имеет значение.

Атрибут NAME задает внешнее имя, используемое компоновщиком. Этот атрибут может указываться в прототипе процедуры или функции. Параметр константа и содержит внешнее имя, используемое компоновщиком для того, чтобы подключить процедуру или функцию из внешней библиотеки.

**Пример:**

```
PROGRAM
MAP
MODULE('External.Obj')
AddCount FUNCTION(LONG),LONG,C,NAME('_AddCount')    !Функция на C называющаяся
'_AddCount'
..
```

**Смотри также:**      Прототипы процедур и функций

## TYPE (Задать определение типа процедуры или функции)

### TYPE

Атрибут TYPE задает прототип, который не является ссылкой на реальную процедуру или функцию. Вместо этого, он определяет имя прототипа, которое должно использоваться в других прототипах для обозначения типа процедуры, передаваемой в другую процедуру или функцию в качестве параметра.

Когда имя “типизированного” прототипа используется в списке параметров другого прототипа, процедура, которую этот другой прототип представляет будет принимать в качестве параметра метку оператора PROCEDURE или FUNCTION, который имеет точно такой же список параметров (и такое же возвращаемое значение, если это функция), что и “типизированный” прототип.

**Пример:**

```
MAP
ProcType PROCEDURE(FILE),TYPE    !Определение типа процедуры-параметра
MyFunc3 FUNCTION(ProcType),STRING    !Параметр - процедура, возвращается
строка
! должна передаваться метка процедуры, которая принимает
! в качестве параметра файл
```

END

**Смотри также:**      Прототипы процедур и функций, Списки параметров в прототипах

## DLL (процедура определена внешне, в библиотеке DLL)

### DLL( [ флаг ] )

**DLL**      Объявляет процедуру или функцию, определенную внешне, в библиотеке DLL.

**флаг** Числовая константа, метка соответствия, или определение системы поддержки проекта, которое задает активен ли данный атрибут. Если флаг установлен в 0, то этот атрибут неактивен, как если бы его вообще не было. Если флаг имеет отличное от нуля значение, то атрибут активен. Флаг может быть неопределенной меткой, и в этом случае атрибут считается активным.

Атрибут DLL указывает, что процедура или функция, в прототипе которой имеется этот атрибут, определена в библиотеке с динамическими связями (DLL). Для 32-битовых приложений атрибут DLL обязателен, так как такие библиотеки являются настраиваемыми (перемещаемыми) в 32-битовом адресном пространстве, которое требует от компилятора еще одного дополнительного разыменовывания (преобразования адреса) при обращении к процедуре.

### Пример:

```
MAP
MODULE('STDFuncs.DLL')      ! DLL - библиотека стандартных функций
Func50 FUNCTION(SREAL),REAL,PASCAL,DLL
END
END
```

## PROC (нет предупреждения, что функция вызывается как процедура)

### PROC

Атрибут PROC указывает, что к функции, в прототипе которой он помещен, можно обращаться как к процедуре, и компилятор при этом не генерирует предупреждающего сообщения. Этот атрибут позволяет в тех случаях, когда возвращаемое функцией значение не играет роли, использовать функцию как процедуру.

### Пример:

```
MAP
MODULE('STDFuncs.DLL')      ! DLL - библиотека стандартных функций
Func50 FUNCTION(SREAL),REAL,PASCAL,PROC
END
END
```

## PRIVATE (использование процедуры ограничено классом или модулем)

### PRIVATE

Атрибут PRIVATE указывает, что к процедуре или функции, в прототипе которой он помещен, можно обращаться из процедур и функций этого же самого исходного модуля. Тем самым она скрывается от процедур и функций других модулей. Обычно этот атрибут указывается в прототипах методов в структурах CLASS, и поэтому к методу можно



обращаться из других методов класса в данном модуле.

### Пример:

```
MAP
  MODULE('STDFuncs.DLL')      ! DLL - библиотека стандартных функций
Func49 FUNCTION(SREAL),REAL,PASCAL,PROC
Proc50 PROCEDURE(SREAL),PRIVATE  !К этой процедуре можно обращаться только из
Func49
  END
END

OneClass CLASS,MODULE('OneClass.CLW'),TYPE
BaseProc  PROCEDURE(REAL Parm)    !Общий метод
Proc      PROCEDURE(REAL Parm),PRIVATE  !Объявить личный метод
  END
TwoClass OneClass    !Экземпляр класса OneClass

CODE
TwoClass.BaseProc(1) !Правильное обращение к BaseProc
TwoClass.Proc(2)     !Неправильное обращение к Proc

!В файле OneClass.CLW:
  MEMBER()

OneClass.BaseProc PROCEDURE(REAL Parm)
  CODE
SELF.Proc(Parm)          ! Правильное обращение к Proc

OneClass.Proc          PROCEDURE(REAL Parm)
  CODE
  RETURN(Parm)
```

## **VIRTUAL (установить, что метод виртуальный)**

### **VIRTUAL**

Атрибут **VIRTUAL** указывает, что процедура или функция, в прототипе которой он задан, является виртуальным методом структуры **CLASS**, содержащий этот прототип. Этот атрибут позволяет методам базового класса обращаться к методам производного класса. Атрибут **VIRTUAL** нужно поместить и в прототип метода базового класса, и в прототип метода производного класса.

### Пример:

```
OneClass CLASS    !Базовый класс
BaseProc  PROCEDURE(REAL Parm)  !Не виртуальный метод
```

```

Proc      PROCEDURE(REAL Parm),VIRTUAL      !Объявить виртуальный метод
END

TwoClass  CLASS(OneClass)      !Производный от OneClass класс
Proc      PROCEDURE(REAL Parm),VIRTUAL      ! Объявить виртуальный метод
END

ClassThree OneClass  !Другой экземпляр объекта класса OneClass
ClassFour  TwoClass  !Другой экземпляр объекта класса TwoClass

CODE
OneClass.BaseProc(1) !BaseProc обращается к OneClass.Proc
TwoClass.BaseProc(2) !BaseProc обращается к TwoClass.Proc
ClassThree.BaseProc(3)      !BaseProc обращается к OneClass.Proc
ClassFour.BaseProc(4) !BaseProc обращается к TwoClass.Proc

OneClass.BaseProc PROCEDURE(REAL Parm)
CODE
SELF.Proc(Parm)      !Обращение к виртуальному методу, или OneClass.Proc
                     ! или TwoClass.Proc, в зависимости от того какой
                     ! экземпляр объекта выполняется

```

## ***Перегрузка функций***

Перегрузка функций означает возможность в нескольких определениях процедур и функций использовать одинаковое имя процедуры или функции. Это одна из форм полиморфизма. Для того, чтобы сделать возможными одинаковые имена, каждая такая процедура или функция должна получать различные параметры, так, чтобы, основываясь на списке параметров, компилятор мог определить, к какой процедуре или функции осуществляется обращение.

Идея состоит в том, чтобы разрешить использование одного и того же имени для нескольких процедур, но имеющих разные прототипы, так что можно выполнять особые (но обычно похожие) действия над данными разных типов.

В языке Clarion также имеются полиморфные функции, для которых используются параметры “?” и “\*?”, однако перегрузка функций расширяет этот полиморфизм, чтобы еще включать параметры-объекты и параметры “поименованные группы”.

Одним из примеров перегрузки функций может служить оператор OPEN языка Clarion, который инициализирует объект для последующего использования в программе. В зависимости от того, объект какого типа ему передан (файл, окно, структура VIEW и т.д.) он выполняет похожие, но физически различные действия.

## Правила перегрузки функций

В языке Clarion имеется встроенное преобразование типов данных, что может затруднить для компилятора реализацию перегрузки. Поэтому есть правила, обуславливающие реализацию компилятором перегрузки функций. Эти правила применяются в следующем порядке:

1. Параметры-объекты приводятся к объектам FILE, KEY, WINDOW, и QUEUE. Если уже после этого можно выбрать прототип, то компилятор это и делает (под это правило попадают большинство функций Clarion). Заметим, что KEY и VIEW неявно происходят от FILE, в то время как APPLICATION и REPORT - от WINDOW.
2. Все параметры “поименованные группы” должны соответствовать друг другу по своей структуре. Параметры-процедуры соответствуют по структуре. Классы должны соответствовать по именам, а не просто по структуре.
3. Прототипы должны соответствовать по числу и порядку следования обязательных параметров. Это третий (а не первый) фактор, по которому обычно компилятор может решить, какой прототип пользователь имел ввиду или выдать наиболее осмысленное сообщение об ошибке.
4. Если соответствующий прототип не найден, то допускается соответствие производных объектов. В этот момент допускается, что KEY может соответствовать FILE, а группа которая предполагается производной, должна соответствовать одному из базовых классов. Если одно предположение о производности объекта не срабатывает, делается следующее и так до трех раз. Теперь все QUEUE соответствуют очередям и группам и т.п. Прежде остальных типов параметров устанавливается происхождение классов.
5. Параметры- переменные (непоименованные) должны точно соответствовать типам реально передаваемых данных. Для параметров-значений делается попытка установить их точное соответствие.
6. Все параметры значения считаются имеющими один и тот же тип.

### Пример:

MAP

Func	PROCEDURE(WINDOW)	! 1
Func	PROCEDURE(FILE)	! 2
Func	PROCEDURE(KEY)	! 3
Func	PROCEDURE(FILE,KEY)	! 4
Func	PROCEDURE(G1)	! 5
Func	PROCEDURE(G0)	! 6
Func	PROCEDURE(KEY,G0)	! 7
Func	PROCEDURE(FILE,G1)	! 8
Func	PROCEDURE(SHORT = 10)	! 9
Func	PROCEDURE(LONG)	! 10
Func	PROCEDURE()	! Неправильно, неотличимо от 9
Func	PROCEDURE(*SHORT)	! 11
Func1	PROCEDURE(*SHORT)	
Func1a	PROCEDURE(*SHORT)	

```

Func2      PROCEDURE(*LONG)
Func       PROCEDURE(Func1) ! 12
Func       PROCEDURE(Func1a) ! Неправильно, то же, что и 12
Func       PROCEDURE(Func2) ! 13
END

G0         GROUP
END
G1         GROUP(G0)
END

CODE
Func(A:Window)      ! Обращение к 1 по правилу 1
Func(A:File) ! Обращение к 2 по правилу 1
Func(A:Key)         ! Обращение к 3 по правилу 1
Func(A:View)        ! Обращение к 2 по правилу 4
Func(A:Key,A:Key)   ! Обращение к 4 по правилу 4 (следовало бы обратиться
! к функции с параметрами key,key, если бы лна была)
Func(A:G0) ! Обращение к 6 по правилу 2
Func(A:G1) ! Обращение к 5 по правилу 2
Func(A:Func2)       ! Обращение к 13 по правилу 2
Func(A:Key,A:G1)   ! Ошибка - неопределенность. Если использовать правило 4,
! то подходят и 7 и 8
Func(A:Short)      ! Ошибка - неопределенность. Обращение к 9 или 11
Func(A:Real)       ! Обращение к 9 по правилу 6
Func              ! Обращение к 9 по правилу 3

```

## Образование имен и совместимость с C++

Каждая перегружаемая функция будет иметь для компоновки имя, составленное из имени процедуры и “искромсанного” списка аргументов. Образование имен разработано так, чтобы обеспечить некоторую степень “перекрестных” обращений между C++ и Clarion. Со стороны C++ необходимо указать:

```
#pragma name(prefix=>"")
```

Имена состоящие из прописных букв. Со стороны Clarion нужно иметь структуру MODULE со строкой нулевой длины в качестве параметра:

```
MODULE("")
END
```

Перекрестные обращения возможны только между теми функциями, прототипы которых содержат только данные приведенные в следующем списке. Клароновские параметры-переменные (передаваемые посредством адреса) соответствуют параметрам ссылок в C, если не указано что они могут опускаться, в этом случае они соответствуют указателям.

```
Clarion  C++
BYTE    unsigned char
```

```

USHORT    unsigned short
SHORT     short
LONG      long
ULONG     unsigned long
SREAL     float
REAL      double
*CSTRING  (with RAW)    char&
<*CSTRING> (with RAW)  char*
<*GROUP>  (with RAW)    void*

```

Отметим, что для совместимости с C++ тип возвращаемых функцией данных не включен в составное имя. Естественным следствием этого является то, функции нельзя различить по типу возвращаемого значения.

### Пример:

// прототипы C++:

```

#pragma name(prefix=>")
void HADD(short,short);
void HADD(long*,unsigned char);
void HADD(short unsigned &);
void HADD(char *,void *);

```

! прототипы Clarion:

```

MODULE("")
    hADD(short,short)
    HaDD(<*long>,byte)
HaD(*ushort)
    HADd(<*CSTRING>,<*GROUP>),RAW
END

```

## Директивы компилятора

Директивы компилятора представляют собой операторы, которые во время компиляции предписывают ему предпринять некоторые действия. В объектный код программы, который генерирует компилятор, эти операторы не включаются, не привносятся, таким образом, никаких издержек в исполняемый модуль.

### BEGIN (определить операторную структуру)

```

BEGIN
    операторы
.

```

**BEGIN**            Объявляет структуру из операторов, рассматриваемую как единый оператор.  
операторы        Исполняемые операторы программы

Директива BEGIN предписывает компилятору обрабатывать группу операторов как

единую структуру. Структура BEGIN должна заканчиваться точкой или оператором END.

В управляющей структуре EXECUTE директива BEGIN обычно используется для того, чтобы несколько операторов рассматривались один.

**Пример:**

```
EXECUTE Value
Proc1  !Выполнить, если Value=1
BEGIN  !Выполнить, если Value=2
  Value += 1
  Proc2
END
Proc3  !Выполнить, если Value=3
END
```

**Смотри также:** оператор EXECUTE

## COMPILE (указать исходный текст для компиляции)

**COMPILE**(терминатор[,выражение])

**COMPILE**      Задаёт порцию строк исходного текста, которые должны быть включены в процесс компиляции.

терминатор      Строковая константа, которая отмечает последнюю строку исходного текста, подлежащего компиляции.

выражение      Выражение, позволяющее осуществить условную компиляцию. Это выражение вида: **МЕТКА СООТВЕТСТВИЯ** = целочисленная константа.

Директива COMPILE указывает блок исходного текста, который должен быть включен в процесс компиляции. Этот включаемый блок начинается с директивы COMPILE и заканчивается строкой, которая содержит строковую константу, указанную в качестве терминатора. Строка, содержащая терминатор, включается в компилируемый блок целиком.

Необязательный параметр выражение обеспечивает условную компиляцию. Вид его фиксирован: метка соответствия или условный переключатель, установленный в системе поддержки проекта, затем знак равенства (=), за которым следует целочисленная константа. Программный текст между директивой COMPILE и терминатором компилируется, только если выражение истинно. Хотя выражение является необязательным, директива без этого параметра не нужна, потому, что если нет явно указанных исключений, то компилируется весь исходный текст. Директивы COMPILE и OMIT противоположны по значению и не могут быть вложены одна в другую.

**Пример:**

```
Demo  EQUATE(1) !Задается значение метки соответствия Demo
CODE
  COMPILE('EndDemoChk', Demo = 1) !Компилировать, если Demo включено
  DO DemoCheck!Проверить ограничения для демо-версии
EndDemoChk      !Конец участка условной компиляции
```

**Смотри также:** директивы OMIT, EQUATE.

**INCLUDE (компилировать текст другого файла)****INCLUDE**(имя файла[,секция])

**INCLUDE** Указывает что, следует компилировать исходный текст, который находится в отдельном файле, не являющемся member-модулем.

имя файла Строковая константа, которая содержит спецификацию файла исходного текста. Если опущено расширение, то подразумевается .CLW

Директива INCLUDE задает исходный текст, который должен компилироваться, и который находится в отдельном файле. Начиная со строки, содержащей директиву INCLUDE, указанный ею файл исходного текста или секция того файла компилируется, как если бы они находились в этом месте компилируемого исходного модуля. Вложенность вставляемых файлов может равняться 3-м, то есть можно вставить оператором INCLUDE файл, в который оператором INCLUDE вставляется файл, в который оператором INCLUDE вставляется файл, но в этот последний файл уже нельзя вставить оператором INCLUDE ничего ...

Для того, чтобы найти файл, компилятор использует redirection-файл (CW.RED), просматривая каталоги, указанные для этого типа файлов (обычно по расширению). Это снимает необходимость указывать полный путь к включаемому в компиляцию файлу. Redirection-файл рассматривается в Руководстве программиста.

**Пример:**

```
GenLedgeR PROCEDURE      !Объявление процедуры
  INCLUDE('filedefs.cla')  !Здесь включить объявление файлов
  CODE                    !Начало программной секции
  INCLUDE('Setups','ChkErr') !Включить проверку ошибок из файла setups.cla
```

**OMIT (указать текст, который не должен компилироваться)****OMIT**(терминатор[,выражение])

**OMIT** Задает порцию строк исходного текста, которые во время компиляции должны пропускаться.

терминатор Строковая константа, которая отмечает последнюю строку блока исходного текста.

выражение Выражение, позволяющее осуществить условное выполнение директивы OMIT. Выражение должно быть типа МЕТКА СООТВЕТСТВИЯ = целочисленная константа.

Директива OMIT задает блок строк исходного текста, которые должны быть пропущены при компиляции. Эти строки могут содержать комментарии к программе или часть программы, которая пропускается при тестировании. Пропускаемый блок начинается директивой OMIT и заканчивается строкой, которая содержит константу, указанную в качестве терминатора. Последняя строка целиком включается в пропускаемый блок.

Необязательный параметр выражение обеспечивает условный пропуск блока. Формат выражения фиксированный: метка соответствия или условный переключатель,

установленный в системе поддержки проекта, затем знак равенства (=), за которым следует целочисленная константа. Директива OMIT выполняется только если выражение истинно.

Директивы COMPILE и OMIT противоположны по значению и не могут быть вложены одна в другую.

### Пример:

```
OMIT(**END**)    !Безусловный OMIT
*****
*  Главный цикл программы
*****
**END**
Demo    EQUATE(0) !Установить значение метки Demo
CODE
    OMIT('EndDemoChk',Demo = 0)    !Пропустить только если Demo выключено
    DO DemoCheck                  !Проверить ограничения для демо-версии
EndDemoChk    !Конец пропускаемого текста
```

Смотри также: директивы COMPILE, EQUATE

## SECTION (указать секцию исходного текста)

### SECTION(строка)

**SECTION** Идентифицирует начало некоторой порции исполняемых операторов.  
строка Строковая константа, представляющая собой имя секции

Директива компилятора SECTION идентифицирует начало блока исполняемых операторов или операторов объявления данных. Имя блока используется в качестве необязательного параметра в директиве INCLUDE для того, чтобы осуществить включение в компиляцию указанного блока исходного текста. Блок заканчивается следующей директивой SECTION или по концу файла исходного текста.

### Пример:

```
SECTION('FirstSection')    !Начало блока
FieldOne STRING(20)
FieldTwo LONG
SECTION('SecondSection')  !Конец предыдущего блока,
    !начало нового
    IF Number <> SavNumber
    DO GetNumber
    END
SECTION('ThirdSection')    !Конец предыдущего блока,
    !начало нового
```



```
CASE Action
OF 1
  DO AddRec
    OF 2
OF 2
  DO ChgRec
OF 3
  DO DelRec
END !Третий блок заканчивается по концу файла
```

**Смотри также:**     директива INCLUDE.



## Глава 3 Объявление переменных

### Операторы объявления

#### BYTE (целочисленная переменная без знака длиной в один байт)

метка	<b>BYTE</b> ([начальное значение])	[,DIM()]	[,OVER()]	[,NAME()]	[,EXTERNAL]	[,STATIC]	[,THRED]	[,AUTO]	[,DLL]	[,PRIVATE]
-------	------------------------------------	----------	-----------	-----------	-------------	-----------	----------	---------	--------	------------

<b>BYTE</b>	объявляет целочисленную переменную без знака длиной в один байт.
Формат:	переменная в один байт
Биты:	7 0
Диапазон:	от 0 до 255
начальное значение	Числовая константа. Если параметр опущен, то начальное значение устанавливается в 0.
DIM()	Размерность, если это массив переменных такого типа.
OVER()	Использование памяти, выделенной другой переменной.
NAME()	Указывает альтернативное внешнее имя для переменной.
EXTERNAL	Указывает на то, что переменная объявляется, а память для нее выделяется во внешней библиотеке. Не допустим в объявлениях внутри структур FILE, QUEUE или GROUP.
DLL	Указывает, что переменная определена в библиотеке DLL. В дополнение к этому атрибуту обязателен атрибут EXTERNAL.
STATIC	Указывает, что память для переменной резервируется во время компиляции.
THRED	Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедуры неявно добавляется также атрибут STATIC.
AUTO	Указывает, что переменная не имеет начального значения.
PRIVATE	Указывает, что переменная “не видна” вне границ модуля, содержащего методы класса. Этот атрибут допустим только внутри структуры CLASS.

#### Пример:

```
Count1 BYTE !Объявить целочисленную со знаком
Count2 BYTE,OVER(Count1) !Объявить “поверх” нее еще одну
Count3 BYTE,DIM(4)!Объявить массив из 4-х однобайтовых целых
Count4 BYTE(5) !Объявить и присвоить начальное значение
Count5 BYTE,EXTERNAL !Объявить как внешнюю переменную
Count6 BYTE,EXTERNAL,DLL !Объявить как внешнюю переменную в библиотеке DLL
Count7 XXXX,NAME('SixCount') !Объявить с внешним именем
ExampleFile FILE,DRIVER('Clarion')!Объявить файл
Record RECORD
Count8 BYTE,NAME('Counter') !Объявить с внешним именем
..
```

## SHORT (целочисленная переменная со знаком длиной в два байта)

метка **SHORT**([начальное значение]) [,DIM()] [,OVER()] [,NAME()] [,EXTERNAL] [,STATIC] [,THRED] [,AUTO] [,DLL] [,PRIVATE]

Формат:        знак    величина

Биты:            15    14        0

Диапазон:        от -32768 до 32767

начальное значение    Числовая константа. Если параметр опущен, то начальное значение устанавливается в 0.

DIM()            Размерность, если это массив переменных такого типа.

OVER()            Использование памяти, выделенной другой переменной.

NAME()            Указывает альтернативное внешнее имя для переменной.

EXTERNAL        Указывает на то, что переменная объявляется, а память для нее выделяется во внешней библиотеке. Не допустим в объявлениях внутри структур FILE, QUEUE или GROUP.

DLL                Указывает, что переменная определена в библиотеке DLL. В дополнение к этому атрибуту обязателен атрибут EXTERNAL.

STATIC            Указывает, что память для переменной резервируется во время компиляции.

THRED            Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедуры неявно добавляется также атрибут STATIC.

AUTO                Указывает, что переменная не имеет начального значения.

PRIVATE            Указывает, что переменная “не видна” вне границ модуля, содержащего методы класса. Этот атрибут допустим только внутри структуры CLASS.

Оператор SHORT объявляет двухбайтовую целочисленную переменную со знаком в формате слова Intel 8086. Старший бит в этом формате представляет собой знак (0 - положительный, 1 - отрицательный). Отрицательные значения представляются в стандартном дополнительном (до 2-х) коде.

### Пример:

Count1    SHORT !Объявить двубайтовую целочисленную со знаком

Count2    SHORT,OVER(Count1)    !Объявить “поверх” нее еще одну

Count3    SHORT,DIM(4)        !Объявить массив из 4-х двубайтовых целых

Count4    SHORT(5)        !Объявить и присвоить начальное значение

Count5    SHORT,EXTERNAL    !Объявить как внешнюю переменную

Count6    SHORT,EXTERNAL,DLL    !Объявить как внешнюю переменную в библиотеке DLL

Count7    SHORT,NAME('SixCount') !Объявить с внешним именем

ExampleFile FILE,DRIVER('Clarion') !Объявить файл

Record    RECORD

Count8    !SHORT,NAME('Counter')    ! Объявить с внешним именем

..

**USHORT (целочисленная переменная без знака длиной в два байта)**

метка      **USHORT**([начальное значение]) [,**DIM**()] [,**OVER**()] [,**NAME**()] [,**EXTERNAL**] [,**STATIC**] [,**THRED**] [,**AUTO**] [,**DLL**] [,**PRIVATE**]

Формат:      величина

Биты:          15                      0

Диапазон:      от 0 до 65535

начальное значение      Числовая константа. Если параметр опущен, то начальное значение устанавливается в 0.

**DIM**()          Размерность, если это массив переменных такого типа.

**OVER**()        Использование памяти, выделенной другой переменной.

**NAME**()        Указывает альтернативное внешнее имя для переменной.

**DLL**            Указывает, что переменная определена в библиотеке DLL. В дополнение к этому атрибуту обязателен атрибут **EXTERNAL**.

**EXTERNAL**    Указывает на то, что переменная объявляется, а память для нее выделяется во внешней библиотеке. Не допустим в объявлениях внутри структур **FILE**, **QUEUE** или **GROUP**.

**STATIC**        Указывает, что память для переменной резервируется во время компиляции.

**THRED**        Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедуры неявно добавляется также атрибут **STATIC**.

**AUTO**          Указывает, что переменная не имеет начального значения.

**PRIVATE**      Указывает, что переменная “не видна” вне границ модуля, содержащего методы класса. Этот атрибут допустим только внутри структуры **CLASS**.

Оператор **USHORT** объявляет двухбайтовую целочисленную переменную без знака в формате слова Intel 8086. Знакового разряда нет.

**Пример:**

```
Count1  USHORT      !Объявить двубайтовую целочисленную со знаком
Count2  USHORT,OVER(Count1) !Объявить “поверх” нее еще одну
Count3  USHORT,DIM(4)   !Объявить массив из 4-х двубайтовых целых
Count4  USHORT(5)      !Объявить и присвоить начальное значение
Count5  USHORT,EXTERNAL !Объявить как внешнюю переменную
Count6  USHORT,EXTERNAL,DLL !Объявить как внешнюю переменную в библиотеке DLL
Count7  USHORT,NAME('SixCount') !Объявить с внешним именем
ExampleFile FILE,DRIVER('Clarion') !Объявить файл
Record  RECORD
Count8  USHORT,NAME('Counter') ! Объявить с внешним именем
```

..

**LONG (целочисленная переменная длиной четыре байта со знаком)**

метка	<b>LONG</b> ([начальное значение]) [,DIM()] [,OVER()] [,NAME()] [,EXTERNAL] [,STATIC] [,THRED] [,AUTO] [,DLL] [,PRIVATE]
Формат:	знак      величина
Биты:	31                      0
Диапазон:	от -2147483648 до 2147483647
начальное значение	Числовая константа. Если параметр опущен, то начальное значение устанавливается в 0.
<b>DIM()</b>	Размерность, если это массив переменных такого типа.
<b>OVER()</b>	Использование памяти, выделенной другой переменной.
<b>NAME()</b>	Указывает альтернативное внешнее имя для переменной.
<b>DLL</b>	Указывает, что переменная определена в библиотеке DLL. В дополнение к этому атрибуту обязателен атрибут EXTERNAL.
<b>EXTERNAL</b>	Указывает на то, что переменная объявляется, а память для нее выделяется во внешней библиотеке. Не допустим в объявлениях внутри структур FILE, QUEUE или GROUP.
<b>STATIC</b>	Указывает, что память для переменной резервируется во время компиляции.
<b>THRED</b>	Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедуры неявно добавляется также атрибут STATIC.
<b>AUTO</b>	Указывает, что переменная не имеет начального значения.
<b>PRIVATE</b>	Указывает, что переменная “не видна” вне границ модуля, содержащего методы класса. Этот атрибут допустим только внутри структуры CLASS.

Оператор **LONG** объявляет четырехбайтовую целочисленную переменную со знаком в формате двойного слова Intel 8086. Старший бит представляет собой знак (0 - положительный, 1 - отрицательный). Отрицательные значения представляются в стандартном дополнительном (до 2-х) коде.

**Пример:**

```
Count1  LONG  !Объявить двубайтовую целочисленную со знаком
Count2  LONG,OVER(Count1)    !Объявить “поверх” нее еще одну
Count3  LONG,DIM(4) !Объявить массив из 4-х длинных целых
Count4  LONG(5)    !Объявить и присвоить начальное значение
Count5  LONG,EXTERNAL  !Объявить как внешнюю переменную
Count6  LONG,EXTERNAL,DLL  !Объявить как внешнюю переменную в библиотеке DLL
Count7  LONG,NAME('SixCount') !Объявить с внешним именем
ExampleFile FILE,DRIVER('Clarion') !Объявить файл
Record  RECORD
Count8  LONG,NAME('Counter') ! Объявить с внешним именем
```

..

**ULONG (целочисленная переменная без знака длиной четыре байта)**

метка	<b>ULONG</b> ([начальное значение])[ <b>DIM</b> ()] [ <b>OVER</b> ()] [ <b>NAME</b> ()] [ <b>EXTERNAL</b> ][ <b>STATIC</b> ] [ <b>THRED</b> ] [ <b>AUTO</b> ] [ <b>DLL</b> ] [ <b>PRIVATE</b> ]
Формат:	величина
Биты:	31                      0
Диапазон:	от 0 до 4294967295
начальное значение	Числовая константа. Если параметр опущен, то начальное значение устанавливается в 0.
<b>DIM</b> ()	Размерность, если это массив переменных такого типа.
<b>OVER</b> ()	Использование памяти, выделенной другой переменной.
<b>NAME</b> ()	Указывает альтернативное внешнее имя для переменной.
<b>DLL</b>	Указывает, что переменная определена в библиотеке DLL. В дополнение к этому атрибуту обязателен атрибут <b>EXTERNAL</b> .
<b>EXTERNAL</b>	Указывает на то, что переменная объявляется, а память для нее выделяется во внешней библиотеке. Не допустим в объявлениях внутри структур <b>FILE</b> , <b>QUEUE</b> или <b>GROUP</b> .
<b>STATIC</b>	Указывает, что память для переменной резервируется во время компиляции.
<b>THRED</b>	Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедуры неявно добавляется также атрибут <b>STATIC</b> .
<b>AUTO</b>	Указывает, что переменная не имеет начального значения.
<b>PRIVATE</b>	Указывает, что переменная “не видна” вне границ модуля, содержащего методы класса. Этот атрибут допустим только внутри структуры <b>CLASS</b> .

Оператор **ULONG** объявляет четырехбайтовую целочисленную переменную без знака в формате двойного слова Intel 8086. Знакового разряда нет.

**Пример:**

```
Count1  ULONG !Объявить двубайтовую целочисленную со знаком
Count2  ULONG,OVER(Count1) !Объявить “поверх” нее еще одну
Count3  ULONG,DIM(4)      !Объявить массив из 4-х длинных целых без знака
Count4  ULONG(5)          !Объявить и присвоить начальное значение
Count5  ULONG,EXTERNAL !Объявить как внешнюю переменную
Count6  ULONG,EXTERNAL,DLL !Объявить как внешнюю переменную в библиотеке DLL
Count7  ULONG,NAME('SixCount')!Объявить с внешним именем
ExampleFile FILE,DRIVER('Clarion') !Объявить файл
Record   RECORD
Count8   ULONG,NAME('Counter')      ! Объявить с внешним именем
..
```

## **SREAL (переменная с плавающей точкой длиной четыре байта со знаком)**

метка	<b>SREAL</b> ([начальное значение]) [,DIM()] [,OVER()] [,NAME()] [,EXTERNAL] [,DLL] [,STATIC] [,THRED] [,AUTO] [,PRIVATE]
-------	--

Формат:                      знак    порядок    мантисса

Биты:                      31    30    23    0

Диапазон:            0, +/- 1.175494 E-38 .. +/- 3.402823 E+38 (6 значащих цифр)

начальное значение    Числовая константа. Если параметр опущен, то начальное значение устанавливается в 0.

**DIM()**                      Размерность, если это массив переменных такого типа.

**OVER()**                    Использование памяти, выделенной другой переменной.

**NAME()**                    Указывает альтернативное внешнее имя для переменной.

**DLL**                        Указывает, что переменная определена в библиотеке DLL. В дополнение к этому атрибуту обязателен атрибут EXTERNAL.

**EXTERNAL**                Указывает на то, что переменная объявляется, а память для нее выделяется во внешней библиотеке. Не допустим в объявлениях внутри структур FILE, QUEUE или GROUP.

**STATIC**                    Указывает, что память для переменной резервируется во время компиляции.

**THRED**                    Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедуры неявно добавляется также атрибут STATIC.

**AUTO**                      Указывает, что переменная не имеет начального значения.

**PRIVATE**                  Указывает, что переменная “не видна” вне границ модуля, содержащего методы класса. Этот атрибут допустим только внутри структуры CLASS.

Оператор **SREAL** объявляет четырехбайтовую переменную со знаком в формате с плавающей точкой Intel 8087 (одинарной точности).

### **Пример:**

Count1    **SREAL**    !Объявить переменную с плавающей точкой 4 байта со знаком

Count2    **SREAL,OVER**(Count1) !Объявить “поверх” нее еще одну

Count3    **SREAL,DIM**(4)    !Объявить массив из 4-х переменных

Count4    **SREAL**(5) !Объявить и присвоить начальное значение

Count5    **SREAL,EXTERNAL**    !Объявить как внешнюю переменную

Count6    **SREAL,EXTERNAL,DLL**    !Объявить как внешнюю переменную в библиотеке

DLL

Count7    **SREAL,NAME**(‘SixCount’)    !Объявить с внешним именем

ExampleFile **FILE,DRIVER**(‘Btrieve’)    !Объявить файл

Record    **RECORD**

Count8    **SREAL,NAME**(‘Counter’)    ! Объявить с внешним именем

..



## REAL (переменная с плавающей точкой длиной восемь байт со знаком)

метка	REAL([начальное значение]) [,DIM()] [,OVER()] [,NAME()] [,EXTERNAL] [,DLL] [,STATIC] [,THRED] [,AUTO][,PRIVATE]
Формат:	знак порядок мантисса
Биты:	63 62 52 0
Диапазон:	0, +/- 2.225073858507201 E-308 .. +/- 1.79769313496231 E+308 (15 значащих цифр)
начальное значение	Числовая константа. Если параметр опущен, то начальное значение устанавливается в 0.
<b>DIM()</b>	Размерность, если это массив переменных такого типа.
<b>OVER()</b>	Использование памяти, выделенной другой переменной.
<b>NAME()</b>	Указывает альтернативное внешнее имя для переменной.
<b>DLL</b>	Указывает, что переменная определена в библиотеке DLL. В дополнение к этому атрибуту обязателен атрибут EXTERNAL.
<b>EXTERNAL</b>	Указывает на то, что переменная объявляется, а память для нее выделяется во внешней библиотеке. Не допустим в объявлениях внутри структур FILE, QUEUE или GROUP.
<b>STATIC</b>	Указывает, что память для переменной резервируется во время компиляции.
<b>THRED</b>	Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедуры неявно добавляется также атрибут STATIC.
<b>AUTO</b>	Указывает, что переменная не имеет начального значения.
<b>PRIVATE</b>	Указывает, что переменная “не видна” вне границ модуля, содержащего методы класса. Этот атрибут допустим только внутри структуры CLASS.

Оператор **REAL** объявляет восьмибайтовую переменную со знаком в формате с плавающей точкой Intel 8087 (двойной точности).

### Пример:

```
Count1 REAL      !Объявить переменную с плавающей точкой 8 байт со знаком
Count2 REAL,OVER(Count1) !Объявить “поверх” нее еще одну
Count3 REAL,DIM(4)   !Объявить массив из 4-х переменных
Count4 REAL(5)      !Объявить и присвоить начальное значение
Count5 REAL,EXTERNAL !Объявить как внешнюю переменную
Count6 REAL,EXTERNAL,DLL !Объявить как внешнюю переменную в библиотеке DLL
Count7 REAL,NAME('SixCount') !Объявить с внешним именем
ExampleFile FILE,DRIVER('Btrieve') !Объявить файл
Record RECORD
Count8 REAL,NAME('Counter')      ! Объявить с внешним именем
```

..

метка	<b>BFLOAT4</b> ([начальное значение]) [ <b>DIM</b> ()] [ <b>OVER</b> ()] [ <b>NAME</b> ()] [ <b>EXTERNAL</b> ] [ <b>DLL</b> ] [ <b>STATIC</b> ] [ <b>THRED</b> ] [ <b>AUTO</b> ][ <b>PRIVATE</b> ]
-------	---

начальное значение Числовая константа. Если параметр опущен, то начальное значение устанавливается в 0.

Оператор **BFLOAT4** объявляет восьмибайтовую переменную со знаком в формате с плавающей точкой Microsoft BASIC (одинарной точности). Этот тип данных обычно используется для совместимости с существующими данными, поскольку перед всеми арифметическими операциями он преобразуется в **REAL**.

• •

**BFLOAT8 (с плавающей точкой длиной восемь байт со знаком)**

метка	<b>BFLOAT8</b> ([начальное значение]) [, <b>DIM</b> ()] [, <b>OVER</b> ()] [, <b>NAME</b> ()] [, <b>EXTERNAL</b> ] [, <b>DLL</b> ] [, <b>STATIC</b> ] [, <b>THRED</b> ] [, <b>AUTO</b> ][, <b>PRIVATE</b> ]
Формат:	порядок    знак    мантисса
Биты:	63    56    55    0
Диапазон:	0, +/- 5.877471754 E-39 .. +/- 1.7014118346 E+38 (15 значащих цифр)
начальное значение	Числовая константа. Если параметр опущен, то начальное значение устанавливается в 0.
<b>DIM</b> ()	Размерность, если это массив переменных такого типа.
<b>OVER</b> ()	Использование памяти, выделенной другой переменной.
<b>NAME</b> ()	Указывает альтернативное внешнее имя для переменной.
<b>DLL</b>	Указывает, что переменная определена в библиотеке DLL. В дополнение к этому атрибуту обязателен атрибут <b>EXTERNAL</b> .
<b>EXTERNAL</b>	Указывает на то, что переменная объявляется, а память для нее выделяется во внешней библиотеке. Не допустим в объявлениях внутри структур <b>FILE</b> , <b>QUEUE</b> или <b>GROUP</b> .
<b>STATIC</b>	Указывает, что память для переменной резервируется во время компиляции.
<b>THRED</b>	Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедуры неявно добавляется также атрибут <b>STATIC</b> .
<b>AUTO</b>	Указывает, что переменная не имеет начального значения.

Оператор **BFLOAT** объявляет восьмибайтовую переменную со знаком в формате с плавающей точкой Microsoft BASIC (двойной точности). Этот тип данных обычно используется для совместимости с существующими данными, поскольку перед всеми арифметическими операциями он преобразуется в **REAL**.

**Пример:**

```
Count1  BFLOAT8      !Объявить переменную с плавающей точкой 8 байт со знаком
Count2  BFLOAT8,OVER(Count1) !Объявить "поверх" нее еще одну
Count3  BFLOAT8,DIM(4)   !Объявить массив из 4-х переменных
Count4  BFLOAT8(5)      !Объявить и присвоить начальное значение
Count5  BFLOAT8,EXTERNAL !Объявить как внешнюю переменную
Count6  BFLOAT8,EXTERNAL,DLL !Объявить как внешнюю переменную в библиотеке DLL
Count7  BFLOAT8,NAME('SixCount') !Объявить с внешним именем
ExampleFile FILE,DRIVER('Btrieve') !Объявить файл
Record  RECORD
Count8  BFLOAT8,NAME('Counter')   ! Объявить с внешним именем
..
```

**DECIMAL (упакованная десятичная переменная длины со знаком)**

метка	<b>DECIMAL</b> (длина[,длина дробной части] [начальное значение]) [, <b>DIM</b> ()][, <b>OVER</b> ()] [, <b>NAME</b> ()][, <b>EXTERNAL</b> ] [, <b>STATIC</b> ] [, <b>THRED</b> ] [, <b>AUTO</b> ] [, <b>DLL</b> ][, <b>PRIVATE</b> ]
Формат:	d d d . . d d d d d d d d d d
Биты:	s 31 30 29 . . 10 9 8 7 6 5 4 3 2 1
Диапазон:	от -99999999999999999999999999999999 до 99999999999999999999999999999999
длина	Обязательно требующаяся числовая константа, содержащая общее число десятичных цифр (целой и дробной частей вместе без учета запятой). Максимальная длина 31 цифра.
длина дробной части	Числовая константа, которая устанавливает число десятичных цифр в дробной части переменной (справа от запятой). Она должна быть меньше или равна параметру длина. Если этот параметр опущен, то объявляется целочисленная переменная в таком формате.
начальное значение	Числовая константа. Если параметр опущен, то начальное значение устанавливается в 0.
<b>DIM</b> ()	Размерность, если это массив переменных такого типа.
<b>DLL</b>	Указывает, что переменная определена в библиотеке DLL. В дополнение к этому атрибуту обязателен атрибут <b>EXTERNAL</b> .
<b>OVER</b> ()	Использование памяти, выделенной другой переменной.
<b>NAME</b> ()	Указывает альтернативное внешнее имя для переменной.
<b>EXTERNAL</b>	Указывает на то, что переменная объявляется, а память для нее выделяется во внешней библиотеке. Не допустим в объявлениях внутри структур <b>FILE</b> , <b>QUEUE</b> или <b>GROUP</b> .
<b>STATIC</b>	Указывает, что память для переменной резервируется во время компиляции.
<b>THRED</b>	Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедуры неявно добавляется также атрибут <b>STATIC</b> .
<b>AUTO</b>	Указывает, что переменная не имеет начального значения.
<b>PRIVATE</b>	Указывает, что переменная “не видна” вне границ модуля, содержащего методы класса. Этот атрибут допустим только внутри структуры <b>CLASS</b> .

Оператор **DECIMAL** объявляет переменной длины упакованную десятичную переменную со знаком. Каждый байт **DECIMAL** переменной содержит две десятичные цифры (по 4 бита на цифру). Самый левый байт содержит знак переменной в старших 4-х разрядах (0 - положительный, любая другая комбинация - отрицательный) и одну десятичную цифру. Таким образом десятичная переменная всегда содержит нечетное число цифр (и **DECIMAL**(10), и **DECIMAL**(11) занимают 6 байт).

**Пример:**

```
Count1 DECIMAL(5,0)      !Объявить десятичную упакованную переменную в 3 байта
Count2 DECIMAL(5,0),OVER(Count1) !Объявить “поверх” нее еще одну
```

```
Count3 DECIMAL(5,0),DIM(4) !Объявить массив из 4-х десятичных переменных
Count4 DECIMAL(5,0)(5) !Объявить и присвоить начальное значение
Count5 DECIMAL(5,0),EXTERNAL !Объявить как внешнюю переменную
Count6 DECIMAL,EXTERNAL,DLL !Объявить как внешнюю переменную в библиотеке
DLL
Count7 DECIMAL,NAME('SixCount') !Объявить с внешним именем
ExampleFile FILE,DRIVER('Btrieve') !Объявить файл
Record RECORD
Count8 DECIMAL,NAME('Counter') !Объявить с внешним именем
..
```

### PDECIMAL (упакованная десятичная переменная длины со знаком)

метка	<b>PDECIMAL</b> (длина[,длина дробной части][начальное значение])[, <b>DIM</b> ()][, <b>DLL</b> ][, <b>OVER</b> ()][, <b>NAME</b> ()][, <b>EXTERNAL</b> ][, <b>STATIC</b> ][, <b>THRED</b> ][, <b>AUTO</b> ][, <b>PRIVATE</b> ]
-------	---

d d d ... d d d d d d d d d d

Формат: 31 30 29 ... 10 9 8 7 6 5 4 3 2 1 s

Биты: 127 5 0

Диапазон: от -99999999999999999999999999999999 до 99999999999999999999999999999999

длина Обязательно требующаяся числовая константа, содержащая общее число десятичных цифр (целой и дробной частей вместе, без учета запятой). Максимальная длина 31 цифра.

длина дробной части Числовая константа, которая устанавливает число десятичных цифр в дробной части переменной (справа от запятой). Она должна быть меньше или равна параметру длина. Если этот параметр опущен, то объявляется целочисленная переменная в таком формате.

начальное значение Числовая константа. Если параметр опущен, то начальное значение устанавливается в 0.

**DIM()** Размерность, если это массив переменных такого типа.

**OVER()** Использование памяти, выделенной другой переменной.

**NAME()** Указывает альтернативное внешнее имя для переменной.

**DLL** Указывает, что переменная определена в библиотеке DLL. В дополнение к этому атрибуту обязателен атрибут EXTERNAL.

**EXTERNAL** Указывает на то, что переменная объявляется, а память для нее выделяется во внешней библиотеке. Не допустим в объявлениях внутри структур FILE, QUEUE или GROUP.

**STATIC** Указывает, что память для переменной резервируется во время компиляции.

**THRED** Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедуры неявно добавляется также атрибут STATIC.

**AUTO** Указывает, что переменная не имеет начального значения.

**PRIVATE** Указывает, что переменная “не видна” вне границ модуля,

содержащего методы класса. Этот атрибут допустим только внутри структуры CLASS.

Оператор **PDECIMAL** объявляет переменной длины упакованную десятичную переменную со знаком в формате Btrieve и IBM/EBCDIC. Каждый байт PDECIMAL переменной содержит две десятичные цифры (по 4 бита на цифру). Самый правый байт содержит знак переменной в младших 4-х разрядах (0Fh или 0Ch - положительный, 0Dh - отрицательный) и одну десятичную цифру. Таким образом десятичная переменная всегда содержит нечетное число цифр (и DECIMAL(10), и DECIMAL(11) занимают 6 байт).

#### Пример:

```
Count1 PDECIMAL(5,0)    !Объявить десятичную упакованную переменную в 3 байта
Count2 PDECIMAL(5,0),OVER(Count1) !Объявить "поверх" нее еще одну
Count3 PDECIMAL(5,0),DIM(4) !Объявить массив из 4-х десятичных переменных
Count4 PDECIMAL(5,0)(5) !Объявить и присвоить начальное значение
Count5 PDECIMAL(5,0),EXTERNAL !Объявить как внешнюю переменную
Count6 PDECIMAL,EXTERNAL,DLL !Объявить как внешнюю переменную в библиотеке
DLL
Count7 PDECIMAL,NAME('SixCount') !Объявить с внешним именем
ExampleFile FILE,DRIVER('Btrieve') !Объявить файл
Record RECORD
Count8 PDECIMAL,NAME('Counter') !Объявить с внешним именем
..
```

## STRING (строка фиксированной длины)

метка	STRING(строковая константа/длина/шаблон) [,DIM()] [,OVER()] [,NAME()] [,DLL] [,EXTERNAL] [,STATIC] [,THRED] [,AUTO][,PRIVATE]
-------	---

Формат:	фиксированное число байтов
Размер:	от 1 до 65520 байтов в 16-ти разрядных приложениях или до 4-х Мбт в 32-х разрядных
длина	Числовая константа, которая определяет число байт в строке. При использовании этого параметра начальное значение переменной - пробелы.
строковая константа	Устанавливает начальное значение строки. Длина строки в байтах устанавливается равной длине этой строковой константы.
шаблон	Используется для форматирования значения, присваиваемого строке. Длина переменной в этом случае устанавливается равной числу байтов, требующихся, чтобы вместить форматированную строку
<b>DIM()</b>	Размерность, если это массив переменных такого типа.
<b>OVER()</b>	Использование памяти, выделенной другой переменной.
<b>NAME()</b>	Указывает альтернативное внешнее имя для переменной.
<b>DLL</b>	Указывает, что переменная определена в библиотеке DLL. В дополнение к этому атрибуту обязателен атрибут EXTERNAL.
<b>EXTERNAL</b>	Указывает на то, что переменная объявляется, а память для нее выделяется во внешней библиотеке. Не допустим в объявлениях внутри структур FILE, QUEUE или GROUP.

<b>STATIC</b>	Указывает, что память для переменной резервируется во время компиляции.
<b>THRED</b>	Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедуры неявно добавляется также атрибут <b>STATIC</b> .
<b>AUTO</b>	Указывает, что переменная не имеет начального значения.
<b>PRIVATE</b>	Указывает, что переменная “не видна” вне границ модуля, содержащего методы класса. Этот атрибут допустим только внутри структуры <b>CLASS</b> .

Оператор **STRING** объявляет строку символов фиксированной длины. Если не указан атрибут **AUTO**, память, выделяемая для переменной типа **STRING**, заполняется пробелами.

Дополнительно к явному объявлению все символьные строки неявно объявляются как **STRING(1)**, **DIM**(длина строки). Это позволяет адресовать каждый символ строки как элемент массива. В случае, если оператор **STRING** уже имеет атрибут **DIM**, то это неявное объявление массива представляет собой самый последний (необязательный) уровень индексов (справа от объявленных явно)

Кроме того, можно непосредственно адресоваться к нескольким символам внутри строки, используя технологию “частей строки”. Эта технология выполняет действия подобные функции **SUB**, только гораздо более гибкая и эффективная. Более гибкая потому, что “часть строки” может использоваться в операции присвоения с обеих сторон от знака равно (=), а функция **SUB** может использоваться только в качестве источника данных. А более эффективна потому, что требует меньших затрат памяти, чем присвоение отдельных символов или функция **SUB**.

Для того, чтобы взять “часть” строки, номера начального и конечного символов в этой части разделяются двоеточием и помещаются в квадратных скобках как индексы неявно объявленного массива. Номера символов могут быть целочисленными константами, переменными или выражениями. Если используются переменные, то между именами переменных и двоеточием должен быть по крайней мере один пробел, чтобы избежать путаницы с префиксами.

### Пример:

```
Name      STRING(20)      !Объявить 20-ти байтовое поле названия
ArrayString STRING(5),DIM(20) !Объявить массив
Company    STRING('Clarion Software, Inc.') !Программистская компания - 22 байта
Phone      STRING(@P(###)###-####P) !Поле телефонного номера - 13 байт
ExampleFile FILE,DRIVER('Clarion') !Объявить файл
Record     RECORD
NameField   STRING(20),NAME('Name') !Объявить с внешним именем
```

### CODE

```
NameField = 'Tammi' !Присвоить значение
NameField[5] = 'y' ! изменить пятый символ
```

NameField[5:6] = 'ie' ! и изменить часть строки

! пятый и шестой символы

ArrayString[1] = 'First' !Присвоить значение первому элементу

ArrayString[1,2] = 'u' !Изменить второй символ в первой строке

ArrayString[1,2:3] = NameField[5:6] !Присвоить часть строки - части другой строки

## CSTRING (строка, заканчивающаяся двоичным нулем)

метка **CSTRING** (строковая константа/длина/шаблон) [,DIM()) [,OVER()) [,NAME())  
[,EXTERNAL] [,DLL] [,STATIC] [,THRED] [,AUTO][,PRIVATE]

Формат: фиксированное число байтов

Размер: от 1 до 65520 байтов в 16-ти разрядных приложениях или до 4-х Мбт в 32-х разрядных

длина Числовая константа, которая определяет число байт, которое будет храниться в строке. В него должен включаться байт для нулевого завершающего символа.

строковая константа Строковая константа, содержащая начальное значение строки. Длина строки устанавливается равной длине этой строковой константы плюс завершающий нулевой символ.

шаблон Шаблон, использующийся для форматирования значения, присваиваемого строке. Длина переменной в этом случае устанавливается равной числу байтов, требующихся, чтобы вместить форматированную строку плюс завершающий нулевой символ.

**DIM()** Размерность, если это массив переменных такого типа.

**OVER()** Использование памяти, выделенной другой переменной.

**NAME()** Указывает альтернативное внешнее имя для переменной.

**DLL** Указывает, что переменная определена в библиотеке DLL. В дополнение к этому атрибуту обязателен атрибут EXTERNAL.

**EXTERNAL** Указывает на то, что переменная объявляется, а память для нее выделяется во внешней библиотеке. Не допустим в объявлениях внутри структур FILE, QUEUE или GROUP.

**STATIC** Указывает, что память для переменной резервируется во время компиляции.

**THRED** Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедуры неявно добавляется также атрибут STATIC.

**AUTO** Указывает, что переменная не имеет начального значения.

**PRIVATE** Указывает, что переменная “не видна” вне границ модуля, содержащего методы класса. Этот атрибут допустим только внутри структуры CLASS.

Оператор **CSTRING** объявляет строку символов, заканчивающуюся нулевым символом (ASCII код равный 0). Если не указан атрибут AUTO, переменная типа CSTRING



инициализируется строкой нулевой длины.

Эта строка соответствует типу, используемому в языке C и типу поля ZSTRING в Btrieve Record Manager. Хотя память для строки выделяется в соответствии с объявленной длиной, завершающий нулевой символ помещается сразу после введенных данных. Тип данных CSTRING следует использовать для достижения совместимости с внешними файлами или процедурами.

Дополнительно к явному объявлению все символьные строки неявно объявляются как CSTRING(1),DIM(длина строки). Это позволяет адресовать каждый символ строки как элемент массива. В случае, если оператор CSTRING уже имеет атрибут DIM, то это неявное объявление массива представляет собой самый последний (необязательный) уровень индексов (справа от объявленных явно)

Кроме того, можно непосредственно адресоваться к нескольким символам внутри строки, используя технологию “частей строки”. Эта технология выполняет действия подобные функции SUB, только гораздо более гибкая и эффективная. Более гибкая потому, что “часть строки” может использоваться в операции присвоения с обеих сторон от знака равно (=), а функция SUB может использоваться только в качестве источника данных. А более эффективна потому, что требует меньших затрат памяти, чем присвоение отдельных символов или функция SUB.

Для того, чтобы взять “часть” строки, номера начального и конечного символов в этой части разделяются двоеточием и помещаются в квадратных скобках как индексы неявно объявленного массива. Номера символов могут быть целочисленными константами, переменными или выражениями. Если используются переменные, то между именами переменных и двоеточием должен быть по крайней мере один пробел, чтобы избежать путаницы с префиксами.

Поскольку строка типа CSTRING должна заканчиваться нулевым кодом ASCII, то в случае обращения к символам строки только как к элементам массива (а не как строке в целом) ответственность за то, что после всех данных в строку помещен нулевой байт, лежит на программисте. Кроме того, в строке типа CSTRING после нулевого символа может иметься остаток предыдущего значения. Из-за этого внутри групп такая строка может работать неправильно.

### Пример:

```
Name      CSTRING(21)      !Объявить поле названия в 21 байт - 20 байт данных
OtherName  CSTRING(21),OVER(Name) !Объявить “поверх” него массив
Contact    CSTRING(21),DIM(4) !Массив полей по 21 байт - 80 байт данных
Company     CSTRING('Clarion Software, Inc.') !Строка в 23 байта - 22 байта данных
Phone      CSTRING(@P(###)###-####P) !Объявить 14 байт - 13 байт данных
ExampleFile FILE,DRIVER('Btrieve') !Объявить файл
```

Record RECORD

NameField CSTRING(20),NAME('Name')!Объявить с внешним именем

CODE

Name = 'Tammi' !Присвоить значение

Name[5] = 'y' ! изменить пятый символ

Name[5:6] = 'ie' ! и изменить часть строки  
! пятый и шестой символы

Contact[1] = 'First' !Присвоить значение первому элементу

Contact[1,2] = 'u' !Изменить второй символ в первой строке

Contact[1,2:3] = NameField[5:6] !Присвоить часть строки - части другой строки

## PSTRING (строка, включающая байт длины)

метка **PSTRING**(строковая константа/шаблон/длина) [**DIM**()] [**DLL**] [**OVER**()]  
[**NAME**()] [**EXTERNAL**] [**STATIC**] [**THRED**] [**AUTO**][**PRIVATE**]

Формат: фиксированное число байтов

Размер: от 2 до 255 байтов

длина Числовая константа, которая определяет число байт, которое будет содержаться в строке. В нее должен включаться байт в начале строки для хранения длины.

строковая константа Строковая константа, содержащая начальное значение строки. Длина строки устанавливается равной длине этой строковой константы плюс байт длины в начале строки.

шаблон Шаблон, использующийся для форматирования значения, присваиваемого строке. Длина переменной в этом случае устанавливается равной числу байтов, требующихся, чтобы вместить форматированную строку плюс байт длины в начале строки.

**DIM**() Размерность, если это массив переменных такого типа.

**OVER**() Использование памяти, выделенной другой переменной.

**NAME**() Указывает альтернативное внешнее имя для переменной.

**DLL** Указывает, что переменная определена в библиотеке DLL. В дополнение к этому атрибуту обязателен атрибут **EXTERNAL**.

**EXTERNAL** Указывает на то, что переменная объявляется, а память для нее выделяется во внешней библиотеке. Не допустим в объявлениях внутри структур **FILE**, **QUEUE** или **GROUP**.

**STATIC** Указывает, что память для переменной резервируется во время компиляции.

**THRED** Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедуры неявно добавляется также атрибут **STATIC**.

**AUTO** Указывает, что переменная не имеет начального значения.

**PRIVATE** Указывает, что переменная "не видна" вне границ модуля, содержащего

методы класса. Этот атрибут допустим только внутри структуры CLASS.

Оператор **PSTRING** объявляет строку символов, состоящую из байта длины и идущих следом байтов данных. Если не указан атрибут AUTO, переменная типа CSTRING инициализируется строкой нулевой длины.

Эта строка соответствует типу, используемому в языке Паскаль и типу поля LSTRING в Btrieve Record Manager. Хотя память для строки выделяется в соответствии с объявленной длиной, байт длины во время выполнения будет содержать действительную длину строки. Для использования во время выполнения программы PSTRING неявно преобразуется в промежуточное значение типа STRING. Тип данных PSTRING следует использовать для достижения совместимости с внешними файлами или процедурами.

Дополнительно к явному объявлению все символьные строки неявно объявляются как PSTRING(1),DIM(длина строки). Это позволяет адресовать каждый символ строки как элемент массива.

В случае, если оператор PSTRING уже имеет атрибут DIM, то это неявное объявление массива представляет собой самый последний (необязательный) уровень индексов (справа от объявленных явно)

Кроме того, можно непосредственно адресоваться к нескольким символам внутри строки, используя технологию “частей строки”. Эта технология выполняет действия подобные функции SUB, только гораздо более гибкая и эффективная. Более гибкая потому, что “часть строки” может использоваться в операции присвоения с обеих сторон от знака равно (=), а функция SUB может использоваться только в качестве источника данных. А более эффективна потому, что требует меньших затрат памяти, чем присвоение отдельных символов или функция SUB.

Для того, чтобы взять “часть” строки, номера начального и конечного символов в этой части разделяются двоеточием и помещаются в квадратные скобки как индексы неявно объявленного массива. Номера символов могут быть целочисленными константами, переменными или выражениями. Если используются переменные, то между именами переменных и двоеточием должен быть по крайней мере один пробел, чтобы избежать путаницы с префиксами.

Поскольку строка типа PSTRING должна начинаться байтом длины, то в случае обращения к символам строки как к элементам массива (а не как строке в целом) ответственность за то, что байт длины содержит корректное значение, лежит на программисте. Байт длины строки PSTRING адресуется как нулевой элемент массива (единственный случай в Clarion, когда массив имеет нулевой элемент). Поэтому, для строки PSTRING(30) допустимый диапазон индексов массива от 0 до 29. Кроме того, в строке типа CSTRING после нулевого символа может иметься остатки предыдущего значения.

Из-за этого внутри групп такая строка может работать неправильно.

### Пример:

```
Name      PSTRING(21)      !Объявить поле названия в 21 байт - 20 байт данных
OtherName PSTRING(21),OVER(Name) !Объявить "поверх" него массив
Contact   PSTRING(21),DIM(4) !Массив полей по 21 байт - 80 байт данных
Company   PSTRING('Clarion Software, Inc.') !Строка в 23 байта - 22 байта данных
Phone     PSTRING(@P(###)###-####P) !Объявить 14 байт - 13 байт данных
ExampleFile FILE,DRIVER('Btrieve') !Объявить файл
Record    RECORD
NameField PSTRING(20),NAME('Name') !Объявить с внешним именем

..

CODE
Name = 'Tammi'      !Присвоить значение
Name[5] = 'y'       ! изменить пятый символ
Name[5:6] = 'ie'     ! и изменить часть строки - пятый и шестой символы
Contact[1] = 'First' !Присвоить значение первому элементу
Contact[1,2] = 'u'   !Изменить второй символ в первой строке
Contact[1,2:3] = NameField[5:6] !Присвоить часть строки - части другой строки
```

## DATE (дата длиной четыре байта)

метка	DATE [,DIM()][,OVER()][,NAME()][,EXTERNAL] [,DLL][,STATIC] [,THRED] [,AUTO][,PRIVATE]
Формат:	год      мм      дд
Диапазон:	год      от 1 до 9999 месяц    от 1 до 12 день     от 1 до 31
<b>DIM()</b>	Размерность, если это массив переменных такого типа.
<b>OVER()</b>	Использование памяти, выделенной другой переменной.
<b>NAME()</b>	Указывает альтернативное внешнее имя для переменной.
<b>DLL</b>	Указывает, что переменная определена в библиотеке DLL. В дополнение к этому атрибуту обязателен атрибут EXTERNAL.
<b>EXTERNAL</b>	Указывает на то, что переменная объявляется, а память для нее выделяется во внешней библиотеке. Не допустим в объявлениях внутри структур FILE, QUEUE или GROUP.
<b>STATIC</b>	Указывает, что память для переменной резервируется во время компиляции.
<b>THRED</b>	Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедуры неявно добавляется также атрибут STATIC.
<b>AUTO</b>	Указывает, что переменная не имеет начального значения.
<b>PRIVATE</b>	Указывает, что переменная "не видна" вне границ модуля, содержащего методы класса. Этот атрибут допустим только внутри структуры CLASS.

Оператор **DATE** объявляет переменную длиной 4 байта для даты. Этот формат соответствует типу поля DATE, используемому в Btrieve Record Manager. Переменная типа DATE, используемая в выражениях, преобразуется в число дней, прошедших с 28 декабря 1800 (стандартная дата Clarion, обычно хранимая как LONG). Допустимый диапазон дат в стандартном формате Clarion с 1 января 1801 года по 31 декабря 2099 года. При присвоении значения, выходящего за границы допустимого диапазона, результат непредсказуем. Тип данных DATE следует использовать для достижения совместимости с внешними файлами или процедурами.

### Пример:

```
DueDate  DATE      !Объявить переменную даты
Otherdate DATE,OVER(DueDate) !Объявить "поверх" нее еще одну
ContactDate  DATE,DIM(4)    !Объявить массив дат
ExampleFile  FILE,DRIVER('Btrieve') !Объявить файл
Record      RECORD
DateRecd    DATE,NAME('DateField') !Объявить с внешним именем
```

Смотри также: стандартная дата

## TIME (переменная для времени длиной четыре байта)

метка	TIME [,DIM()][,OVER()][,NAME()][,EXTERNAL] [,DLL] [,STATIC] [,THRED][,AUTO][,PRIVATE]
Формат:	чч мм сс сд
Диапазон:	часы от 0 до 23 минуты от 0 до 59 секунды от 0 до 59 секунды/100 от 0 до 99
<b>DIM()</b>	Размерность, если это массив переменных такого типа.
<b>OVER()</b>	Использование памяти, выделенной другой переменной.
<b>NAME()</b>	Указывает альтернативное внешнее имя для переменной.
<b>DLL</b>	Указывает, что переменная определена в библиотеке DLL. В дополнение к этому атрибуту обязателен атрибут EXTERNAL.
<b>EXTERNAL</b>	Указывает на то, что переменная объявляется, а память для нее выделяется во внешней библиотеке. Не допустим в объявлениях внутри структур FILE, QUEUE или GROUP.
<b>STATIC</b>	Указывает, что память для переменной резервируется во время компиляции.
<b>THRED</b>	Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедуры неявно добавляется также атрибут STATIC.
<b>AUTO</b>	Указывает, что переменная не имеет начального значения.

**PRIVATE** Указывает, что переменная “не видна” вне границ модуля, содержащего методы класса. Этот атрибут допустим только внутри структуры CLASS.

Оператор **TIME** объявляет переменную длиной 4 байта для времени. Этот формат соответствует типу поля TIME, используемому в Btrieve Record Manager. Переменная типа TIME, используемая в выражениях, преобразуется в число сотых долей секунды, прошедших с полуночи (стандартный формат времени в Clarion, обычно хранимый как LONG). Тип данных TIME следует использовать для достижения совместимости с внешними файлами или процедурами.

Пример:

```
ChkoutTime      TIME !Объявить переменную времени
OtherTIME TIME,OVER(ChkoutTime)      !Объявить “поверх” нее еще одну
ContactTIME     TIME,DIM(4)          !Объявить массив дат
ExampleFile     FILE,DRIVER('Btrieve') !Объявить файл
Record          RECORD
TimeRecd        TIME,NAME('TimeField') !Объявить с внешним именем
..
```

**Смотри также:** стандартный формат времени

## GROUP (составная структура данных)

метка	<b>GROUP</b> ( [ группа ] ) [ ,PRE() ] [ ,DIM() ] [ ,OVER() ] [ ,NAME() ] [ ,EXTERNAL ] [ ,STATIC ] [ ,THRED ] [ ,BINDABLE ] [ ,TYPE ] [ ,DLL ] [ ,PRIVATE ]
	объявления данных
	<b>END</b>

<b>GROUP</b>	Объявляет составную структуру данных.
группа	Метка ранее объявленной группы или очереди, от которой данная группа наследует структуру. Это может быть GROUP или QUEUE с атрибутом TYPE.
объявления	Несколько последовательных объявлений переменных
<b>PRE()</b>	Объявление префикса для переменных из этой структуры. Недопустим для группы внутри структуры RECORD.
<b>DIM()</b>	Размерность, если это массив переменных такого типа.
<b>OVER()</b>	Использование памяти, выделенной другой переменной.
<b>NAME()</b>	Указывает альтернативное внешнее имя для переменной.
<b>EXTERNAL</b>	Указывает на то, что переменная объявляется, а память для нее выделяется во внешней библиотеке. Не допустим в объявлениях внутри структур FILE, QUEUE или GROUP.
<b>DLL</b>	Указывает, что переменная определена в библиотеке DLL. В дополнение к этому атрибуту обязателен атрибут EXTERNAL.

<b>STATIC</b>	Указывает, что память для переменной резервируется во время компиляции.
<b>THRED</b>	Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедуры неявно добавляется также атрибут <b>STATIC</b> .
<b>BINDABLE</b>	Указывает, что все переменные этой группы можно использовать в динамических выражениях.
<b>TYPE</b>	Указывает, данная группа является объявлением типа для групп, передаваемых в качестве параметров.
<b>PRIVATE</b>	Указывает, что переменная “не видна” вне границ модуля, содержащего методы класса. Этот атрибут допустим только внутри структуры <b>CLASS</b> .

Структура **GROUP** позволяет ссылаться на несколько переменных по одному имени. Таким способом удобно организовать массив для набора переменных, присваивать значения или сравнивать наборы переменных в одном операторе. В больших сложных программах структура **GROUP** полезна для хранения данных, имеющих нечто объединяющее. Объявление группы должно заканчиваться точкой или оператором **END**. Начало структуры группы, объявляемой с параметром группа, точно совпадает со структурой этой группы; объявляемая группа наследует структуру полей группы, указанной параметром группа. Кроме них в объявляемой группе могут быть свои собственные объявления данных, которые идут следом за наследуемыми полями. Если параметр группа указывает на структуру **QUEUE** или **RECORD**, то наследуется только состав полей, но не функциональное назначение этих структур.

При использовании в операторе или выражении группа рассматривается как строковая переменная, составленная из всех переменных в этой структуре. Структура **GROUP** может быть вложена в другую структуру данных, как например **RECORD** или другая группа.

Когда группа рассматривается как строка, числовые переменные, объявленные в ней, из-за своего двоичного формата сравниваются неправильно (кроме **DECIMAL**). По этой причине построение ключа по группе, которая содержит числовые переменные, может приводить к появлению неожиданной последовательности значений в ключе.

Переменные из группы с атрибутом **BINDABLE** можно использовать в динамических выражениях. Значение атрибута **NAME** всех этих переменных является логическим именем, используемым в динамических выражениях. Если атрибут **NAME** не указан, то используется имя переменной (включая префикс). Для имен всех переменных в **EXE**-модуле резервируется место. Таким образом создается программа большего размера, которая использует больше памяти. Поэтому атрибут **BINDABLE** в группе следует использовать только в том случае, если большую часть составляющих ее полей собираетесь использовать в динамических выражениях.

Для группы с атрибутом **TYPE** памяти не распределяется вообще; это только определение типа для групп передаваемых в качестве параметров в процедуры и функции. Такой способ

позволяет принимающей параметр процедуре адресоваться непосредственно к полям в переданной группе. Объявление параметра в операторе PROCEDURE или FUNCTION может устанавливать для передаваемой группы локальный префикс, поскольку оно указывает имя, используемое в данной процедуре, для передаваемой группы, однако, если используется синтаксис уточнения имен, то в объявлении префикса нет необходимости. Например, в операторе PROCEDURE(LOC:PassedGroup) объявляется, что для непосредственного обращения к полям - компонентам переданной в качестве параметра группы, в данной процедуре используется префикс LOC: (наряду с именами полей, использованными в определении типа).

### Пример:

```

PROGRAM
PassGroup GROUP,TYPE      !Определение типа для передаваемой группы
F1      STRING(20)        ! первое поле
F2      STRING(1)! среднее поле
F3      STRING(20)        ! последнее поле
END
MAP
    MyProc1(PassGroup)  !передается группа, описанная точно как PassGroup
END
NameGroup GROUP      !Группа ФИО
First      STRING(20)  !Имя
Middle     STRING(1)!Инициал
Last       STRING(20)  !Фамилия
.          !Конец объявления группы
NameGroup2 GROUP(PassGroup)  !Группа, наследующая структуру полей группы
PassGroup
    ! что приводит к тому, что поля NameGroup2.F1,
    ! NameGroup2.F2 и NameGroup2.F3
END !объявляются в данной группе
DateTimeGrp GROUP,DIM(10) !Массив даты/время
Date      LONG      !
Time      LONG      !
.          !Конец объявления группы
DateTimeGrp GROUP,DIM(10) !Массив дат.времени
Date      LONG
Time      LONG
END !Конец объявления группы

FileNames GROUP,BINDABLE!Группа, поля в которой можно использовать в
    ! динамических выражениях
FileName  STRING(8),NAME('FILE')  !Динамическое имя: FILE
Dot       STRING('.')              !Динамическое имя: Dot
Extension STRING(3),NAME('EXT')    !Динамическое имя: EXT
END

```



```
CODE
MyProc1(NameGroup) !Обратиться к процедуре, передав параметром
! группы NameGroup
MyProc1(NameGroup2) ! Обратиться к процедуре, передав параметром
! группы NameGroup2
```

```
MyProc1  PROCEDURE(PassedGroup) !Процедура, принимающая параметром группу
LocalVar STRING(20)
CODE
LocalVar = PassedGroup.F1 !Присвоить переменной LocalVar
! значение первого поля из переданного параметра
```

**Смотри также:** Синтаксис уточнения имен

## CLASS (объявление объекта)

метка	<b>CLASS</b> ( [ базовый_класс ] ) [ ,EXTERNAL ] [ ,DLL ] [ ,STATIC ] [ ,THREAD ] [ ,BINDABLE ] [ ,MODULE( ) ] [ , TYPE ] [ данные и методы ]
	<b>END</b>

- CLASS** Объект, содержащий объявления данных и методы, которые манипулируют данными
- базовый\_класс** Метка ранее объявленного класса, данные и методы которого наследует новая структура CLASS. Это может быть структура CLASS с атрибутом TYPE.
- EXTERNAL** Указывает, что объект определен во внешней библиотеке и там ему выделена память.
- DLL** Указывает, что объект определен в библиотеке DLL. В этом случае обязательно указание атрибута EXTERNAL .
- STATIC** Указывает, что память для переменной резервируется во время компиляции.
- THREAD** Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедур неявно добавляется также атрибут STATIC.
- BINDABLE** Указывает, что все переменные принадлежащие к этому классу можно использовать в динамических выражениях.
- MODULE** Указывает модуль исходного текста, содержащий определения процедур и функций - методов данного класса. Этот атрибут служит для того же, что и структура MODULE в структуре MAP.
- TYPE** Указывает, что данная структура CLASS представляет собой определение типа для наследования другими классами или использования в списках аргументов в прототипах.
- данные и методы** Объявления данных и прототипы процедур и/или функций. Элементы данных могут быть только те, которые могут объявляться в структуре

## GROUP.

Структура CLASS объявляет объект, который содержит данные и методы, которые манипулируют этими данными. Структура CLASS должна заканчиваться точкой или оператором END. Структуру CLASS, содержащую методы нельзя объявлять в процедуре или функции.

Структура, объявленная с параметром базовый\_класс создает производный класс, который наследует все данные и методы, принадлежащие базовому классу. Производный класс может дополнительно содержать свои собственные данные и методы. Для всех данных, явно объявленных в производном классе, создаются новые переменные - даже если они объявляются с теми же самыми именами, что и данные в базовом классе. Любой метод, прототип которого указан в производном классе, замещает наследуемый метод, если они имеют одинаковый список параметров. Если же два метода имеют различные списки параметров, то в производном классе создаются полиморфные функции, которые должны подчиняться правилам перегрузки функций.

Данные, принадлежащие классу доступны всем методам этого класса. Данным структуры CLASS, имеющим атрибут TYPE, память не выделяется. Такая структура представляет собой лишь определение класса, которое наследуется другими классами, или которое передается в качестве параметра в процедуры или функции. К методам структуры CLASS с атрибутом TYPE, нельзя обращаться непосредственно, а только через методы классов наследующих их.

Определение процедур и функций методов (не прототипов, а их исполняемых кодов) является внешним по отношению к классам. В определении метода или к имени процедуры или функции спереди должно присоединяться имя класса, или первым (неявным) параметром в списке параметров, передаваемых в процедуру или функцию, является имя класса с меткой SELF. Помните, что в операторе определения процедуры или функции, передаваемые параметры снабжаются именами, поэтому, поскольку имя класса является типом данных неявного первого параметра, ему помимо имени класса требуется имя SELF.

Ссылки на данные принадлежащие какому-либо классу, можно осуществлять используя Синтаксис уточнения имен, присоединяя спереди к имени данных имя класса, имя экземпляра класса или имя SELF, означающее текущий исполняемый экземпляр класса. Используя Синтаксис уточнения имен, можно и обращаться к методам классов (присоединяя имя класса спереди к имени метода), или передавая имя класса в качестве первого, (неявного) параметра в списке, передаваемых в процедуру или функцию параметров.

Если имеется метод представленный в данном классе прототипом с тем же самым именем, что и метод с атрибутом VIRTUAL в базовом классе, то в прототипе этого метода в производном классе также должен быть атрибут VIRTUAL. Атрибут VIRTUAL в обоих

этих прототипах создает виртуальные методы, которые позволяют методам базового класса обращаться к одноименным методам в производном классе для выполнения функций характерных для производного класса, о которых в базовом классе ничего неизвестно.

Каждый экземпляр класса, будь то базовый класс, производный класс или объявленный экземпляр любого из них содержит свой собственный набор, распределенный в памяти определенной для данного экземпляра. Тем не менее имеется только по одной копии наследуемых методов (расположенные в том классе, в котором они объявлены), к которым происходит обращение из всех экземпляров данного класса и всех производных классов.

Внутри методов класса ссылки на элементы данных можно осуществлять добавлением спереди к имени переменной слова SELF вместо имени класса. Это позволяет методам обобщенно ссылаться на данные текущего выполняемого экземпляра класса, не принимая во внимание, выполняется ли базовый класс, производный или иди какой-либо экземпляр любого из них. Это также позволяет в базовом классе обращаться к виртуальным методам производного класса.

Существует два способа объявить экземпляр класса: используя оператор LIKE, или просто указав класс в качестве типа данных создаваемого экземпляра. В любом случае новый экземпляр наследует все данные и методы класса для которого создавался экземпляр. В объявлении экземпляра допустимы все атрибуты класса за исключением атрибутов MODULE и TYPE

### Пример:

!Файл ClassPrg.CLW содержит:

PROGRAM

MAP. !Структура MAP требуется только для BUILTINS.CLW

```
OneClass  CLASS      !Базовый класс
NameGroup GROUP      !Ссылаться как к OneClass.NameGroup
First     STRING(20)  ! ссылаться как к OneClass.NameGroup.First
Last      STRING(20)  ! ссылаться как к OneClass.NameGroup.Last
END
BaseProc  PROCEDURE(REAL Parm) !Объявление прототипа метода
Func      FUNCTION(REAL Parm),STRING,VIRTUAL !Объявление прототипа
виртуального метода
Proc      PROCEDURE(REAL Parm),VIRTUAL ! Объявление прототипа виртуального
метода
END !Конец объявления класса

TwoClass  CLASS(OneClass),MODULE('TwoClass.CLW') !Производный от
OneClass
Func      FUNCTION(LONG Parm),STRING ! замещает OneClass.Func
Proc      PROCEDURE(STRING Msg, LONG Parm) !функционально перегружаемая
END
```

```

ClassThree CLASS(TwoClass),MODULE('Class3.CLW')      !Производный от TwoClass
Func      FUNCTION(<STRING Msg>,LONG Parm),STRING,VIRTUAL
Proc      PROCEDURE(REAL Parm),VIRTUAL
          END

```

```

ClassFour LIKE(ClassThree) !Объявить экземпляр класса ClassThree
ClassFive ClassThree !Объявить экземпляр класса ClassThree

```

```

          CODE
          OneClass.NameGroup = '|OneClass Method'      !Присвоить значения каждому
экземпляру NameGroup
          TwoClass.NameGroup = '|TwoClass Method'
          ClassThree.NameGroup = '|ClassThree Method'
          ClassFour.NameGroup = '|ClassFour Method'
          MESSAGE(OneClass.NameGroup & OneClass.Func(1.0)) !Обращение к
OneClass.Func
          MESSAGE(TwoClass.NameGroup & TwoClass.Func(2)) !Обращения к
TwoClass.Func
          MESSAGE(ClassThree.NameGroup & ClassThree.Func('|Call ClassThree.Func',3.0))
! Обращения к
          ! ClassThree.Func
          MESSAGE(ClassFour.NameGroup & ClassFour.Func('|Call ClassFour.Func',4.0))
          !Также обращение ClassThree.Func
          OneClass.BaseProc(5) !BaseProc обращается к OneClass.Proc и Func
          BaseProc(TwoClass,6) !BaseProc также обращается к OneClass.Proc и Func
          TwoClass.Proc('Second Class',7) !обращается к TwoClass.Proc (перегруженной)
          ClassThree.BaseProc(8) !BaseProc обращается к ClassThree.Proc и Func
          ClassFour.BaseProc(9) !BaseProc также обращается к ClassThree.Proc и Func
          Proc(ClassFour,'Fourth Class',10) !обращение к TwoClass.Proc (перегруженной)

```

```

OneClass.BaseProc PROCEDURE(REAL Parm)      !определение OneClass.BaseProc
          CODE
          MESSAGE(Parm & SELF.NameGroup & '|BaseProc executing|calling SELF.Proc
Virtual method')
          SELF.Proc(Parm) !обращение к виртуальному методу
          MESSAGE(Parm & SELF.NameGroup & '|BaseProc executing|calling SELF.Func
Virtual method')
          MESSAGE(SELF.NameGroup & SELF.Func(Parm)) ! обращение к виртуальному
методу

```

```

OneClass.Func FUNCTION(REAL Parm) !Определение OneClass.Func
          CODE
          RETURN('|Executing OneClass.Func - ' & Parm)

```

```

Proc      PROCEDURE(OneClass SELF,REAL Parm)      ! Определение
OneClass.Proc

```

```
CODE
MESSAGE(SELF.NameGroup & ' |Executing OneClass.Proc - ' & Parm)
```

```
!Файл TwoClass.CLW содержит:
MEMBER('ClassPrg')
```

```
Func      FUNCTION(TwoClass SELF, LONG Parm)      ! Определение TwoClass.Func
CODE
RETURN(' |Executing TwoClass.Func - ' & Parm)
```

```
TwoClass.Proc  PROCEDURE(STRING Msg, LONG Parm)      ! Определение
TwoClass.Proc
CODE
MESSAGE(Msg & ' |Executing TwoClass.Proc - ' & Parm)
```

```
! Файл Class3.CLW содержит:
MEMBER('ClassPrg')
```

```
ClassThree.Func  FUNCTION(<STRING Msg>, LONG Parm)      ! Определение
ClassThree.Func
CODE
SELF.Proc(Msg, Parm)  !Обращение к TwoClass.Proc (перегруженной)
RETURN(Msg & ' |Executing ClassThree.Func - ' & Parm)
```

```
ClassThree.Proc  PROCEDURE(REAL Parm) !Определение ClassThree.Proc
CODE
SELF.Proc('Called from ClassThree.Proc', Parm)  !обращение к TwoClass.Proc
MESSAGE(SELF.NameGroup & ' |Executing ClassThree.Proc - ' & Parm)
```

**Смотри также:** Уточнение имен, прототипы процедур и функций в MODULE,  
Перегрузка функций

**LIKE (наследуемый тип данных)**

новая метка	<b>LIKE</b> (существующая метка)	<b>[,DIM()]</b>	<b>[,OVER()]</b>	<b>[,PRE()]</b>
	<b>[,NAME()]</b>	<b>[,BINDABLE]</b>	<b>[,EXTERNAL]</b>	<b>[,DLL]</b>
		<b>[,STATIC]</b>	<b>[,THREAD]</b>	

<b>LIKE</b>	Объявить переменную, тип данных которой наследуется от другой переменной.
новая метка	Метка нового объявления элемента данных
существующая метка	Метка элемента данных, чье объявление будет использоваться
<b>DIM()</b>	Размерность, если это массив переменных такого типа.
<b>OVER()</b>	Использование памяти, выделенной другой переменной.
<b>PRE</b>	Объявляет префикс для переменных в новой структуре (если <b>LIKE</b> объявляет составную структуру данных). Префикс необязателен, поскольку при непосредственном обращении к компонентам новой структуры можно использовать новое имя в контексте синтаксиса уточнения имен.
<b>NAME()</b>	Указывает альтернативное внешнее имя для переменной.
<b>EXTERNAL</b>	Указывает на то, что переменная объявляется, а память для нее выделяется во внешней библиотеке. Не допустим в объявлениях внутри структур <b>FILE</b> , <b>QUEUE</b> или <b>GROUP</b> .
<b>DLL</b>	Указывает, что переменная определена в библиотеке <b>DLL</b> . В дополнение к этому атрибуту обязателен атрибут <b>EXTERNAL</b> .
<b>STATIC</b>	Указывает, что память для переменной резервируется во время компиляции.
<b>THRED</b>	Указывает, что память для переменной выделяется отдельно для каждого исполняемого процесса. Для данных локальных для процедуры неявно добавляется также атрибут <b>STATIC</b> .
<b>BINDABLE</b>	Указывает, что все переменные этой группы можно использовать в динамических выражениях.

Оператор **LIKE** предписывает компилятору определить новую метку, используя то же самое определение как для существующей метки, включая все атрибуты. Если изменяется описание существующей метки, то изменяются и свойства новой метки.

Для новой метки дополнительно можно использовать атрибуты **DIM** и **OVER**. Однако, если существующая метка имеет атрибут **DIM**, то новая метка становится массивом автоматически. Если в операторе **LIKE** добавляется дополнительный атрибут **DIM**, то заданная им размерность массива расширяет уже имеющуюся размерность.

При необходимости могут использоваться атрибуты **PRE** и **NAME**. Если существующая метка уже имеет эти атрибуты, то новая метка их унаследует, что вызовет появление ошибок при компиляции. Для того, чтобы избежать этого, укажите новые атрибуты, чтобы заместить наследуемые.

Если существующая метка представляет структуру **QUEUE**, то оператор **LIKE** не создает новую структуру **QUEUE**, поскольку новая метка рассматривается просто как структура

GROUP. Это справедливо и для случая, когда существующая метка представляет структуру RECORD.

Оператор LIKE можно использовать для того, чтобы создать новый экземпляр класса. Однако, простое объявление нового экземпляра, указывая класс в качестве типа данных, работает как неявный оператор LIKE. В обоих способах объявления нового экземпляра недопустимо использование атрибутов DIM, OVER, PRE, и NAME, все же другие атрибуты в объявлении нового экземпляра класса можно использовать.

### Пример:

```

Amount    REAL !Объявить поле
QTDAmount    LIKE(Amount)    !Использовать то же самое объявление
YTDAmount    LIKE(QTDAmount)    !Снова использовать то же самое объявление
MonthlyAmts    LIKE(Amount),DIM(12) !Использовать то же объявление для массива
AmtPrPerson    LIKE(MonthlyAmts),DIM(10)
                !Использовать то же объявление для массива в 120 элементов (12,10)
ConStruct GROUP,PRE(Con) !Объявить группу
Field1      LIKE(Amount)    ! Construct.field1 - real
Field2      STRING(10)      ! Construct.field2 - string(10)
                END
NewGroup LIKE(Construct)    !Объявить новую группу
                !      NewGroup.field1 - real
                !      NewGroup.field2 - string(10)
MyQue      QUEUE    !Определить очередь
Field1      STRING(10)
Field2      STRING(10)
                END

MyGroup    LIKE(MyQue)    !Определить новую GROUP, такую как QUEUE
AmountFile FILE,DRIVER('Clarion'),PRE(Amt)
Record     RECORD
Amount     REAL    !Объявить поле
QTDAmount    LIKE(Amount)    !Использовать то же самое объявление
Animal    CLASS
Feed      PROCEDURE(short amount),VIRTUAL
Die       PROCEDURE
Age       LONG
Weight    LONG
                END

Cat    LIKE(Animal)    !Новый экземпляр класса Animal

Bird    Animal    ! Новый экземпляр класса Animal (неявный LIKE)

```

**Смотри также:** атрибуты DIM, OVER, PRE, NAME, Синтаксис уточнения имен.

## Неявно объявляемые переменные

Некоторые переменные не объявляются. Они создаются компилятором при первом их упоминании в тексте программы. Неявно объявленные переменные автоматически инициализируются пробельным или нулевым значением; нет необходимости присваивать значение перед использованием. Можно смело полагать, что перед тем как неявным переменным в программе впервые присваиваются значения, они содержат пробелы или нуль.

Любым неявно объявленным переменным, использованным в секции объявления глобальных данных (между ключевыми словами PROGRAM и CODE) назначается статическая память. Любые неявно объявленные переменные, использованные между ключевыми словами MEMBER и PROCEDURE (или FUNCTION) становятся данными модуля тоже статическими. Остальные неявно объявленные переменные являются локальными данными, которым выделяется динамическая память в стеке программы.

Поскольку компилятор создает неявно объявленные переменные по мере их упоминания в программе, существует опасность, что могут возникать проблемы, которые трудно отследить. Проблемы, связанные с отсутствием сообщения об ошибке во время компиляции и проверки типа неявно объявленной переменной. Например, вы набрали неверно имя ранее использованной неявной переменной и компилятор не сообщит об этом а просто создаст новую с новым именем. Когда в вашей программе будет проверяться значение первой переменной, оно будет неверным. Поэтому неявно объявленные переменные следует использовать внимательно и осторожно, и только в ограниченном контексте (или не использовать вообще).

Неявно объявленные переменные обычно используются для индексов массивов, логических переключателей (истина/ложь), промежуточных переменных в сложных вычислениях, управляющих переменных цикла и т.д. В языке Clarion имеется три типа неявно объявленных переменных:

- #     Метка, заканчивающаяся знаком фунта, именует переменную типа LONG.
- \$     Метка, заканчивающаяся знаком доллара, именует переменную типа REAL.
- “     Метка, заканчивающаяся двойной кавычкой, именует переменную типа STRING.

### Пример:

```

LOOP Counter# = 1 TO 10   !Неявная переменная типа LONG
  ArrayField[Counter#] = Counter# * 2   !Инициализация массива
.
Address" = CLIP(City) & ' ' & State & ' ' & Zip           !Неявная строка
SHOW(12,16,Address")    !Использование для просмотра временного значения
Percent$ = ROUND((Quota / Sales),.1) * 100 !Неявная REAL
SHOW(15,22,Percent$,@P%<<<.#P) !Использование для просмотра временного
значения

```



**Смотри также:** объявление данных и распределение памяти

## Переменные-указатели

Переменные-указатели содержат ссылку на объявление других данных (целевых данных или цель). Объявляется переменная-указатель добавлением спереди амперсанда (&) к типу соответствующей ей цели (например: &BYTE, &FILE, &LONG, и т.д.). В зависимости от типа данных цели указатель может содержать адрес памяти или более сложную внутреннюю структуру (описывающую расположение и тип целевых данных).

Допустимые переменные-указатели:

&BYTE,	&SHORT,	&USHORT,	&LONG,
&ULONG,	&DATE	&TIME	&REAL,
&SREAL,	&BFLOAT8,	&BFLOAT4,	&DECIMAL,
&PDECIMAL,	&STRING,	&CSTRING,	&PSTRING,
&QUEUE,	&FILE,	&BLOB,	&VIEW,
&WINDOW.			

Дополнительно к приведенному выше списку можно получить указатели на “поименованные” группу и очередь (&GroupName and &QueueName) и на “поименованные” классы (&ClassName). Это позволяет использовать указатели для передачи “поименованных групп” в качестве параметров.

Нельзя объявлять переменные-указатели внутри структур FILE и VIEW, но можно внутри структур GROUP, QUEUE, и CLASS.

Для объявления указателей типа &STRING, &CSTRING, &PSTRING, &DECIMAL, и &PDECIMAL не требуется указание параметра длина, поскольку необходимая информация о целевых данных содержится в ссылке. Это значит, что переменная-указатель &STRING может содержать ссылку на переменную типа STRING произвольной длины. Переменная-указатель, объявленная как &WINDOW, может содержать ссылку или на структуру типа APPLICATION, WINDOW, или на структуру REPORT.

Оператор &= выполняет операцию присвоения указателя. Он присваивает переменной-указателю значение указателя на объект в правой части оператора.

Использование переменной-указателя синтаксически допустимо в любом контексте, в котором допустимо использование целевой переменной или структуры. При использовании в исполняемом операторе переменная-указатель автоматически разыменовывается, представляя в оператор значение целевого объекта. Единственным исключением является операция присвоения указателей, когда указателю присваивается ссылка на данные на которые он указывает. Например,

RefVar1 &= Var1 !RefVar1 теперь указывает на Var1

RefVar2 &= RefVar1 !RefVar2 теперь также указывает на Var1

Указатели не ограничены рамками исполняемого процесса, и таким образом могут использоваться для ссылок на элементы данных в другом исполняемом процессе.

### Пример:

```
App1      APPLICATION('Hello')
          END
```

```
App2      APPLICATION('Buenos Dias')
          END
```

```
AppRef    &WINDOW !Ссылка на APPLICATION WINDOW или REPORT
```

```
Animal    CLASS
```

```
Feed      PROCEDURE(short amount),VIRTUAL
```

```
Die        PROCEDURE
```

```
Age        LONG
```

```
Weight     LONG
```

```
END
```

```
Carnivore CLASS(Animal),TYPE
```

```
Feed      PROCEDURE(Animal)
```

```
END
```

```
Cat        CLASS(Carnivore)
```

```
Feed      PROCEDURE(short amount),VIRTUAL
```

```
Potty      BYTE
```

```
END
```

```
Bird        Animal      !Экземпляр класса Animal
```

```
AnimalRef   &Animal      !ссылка на класс Animal
```

```
CODE
```

```
IF CTL:Language = 'Spanish' !Если пользователь испаноязычный
```

```
  AppRef &= App1           !установить ссылку на испаноязычное окно
```

```
ELSE
```

```
  AppRef &= App2           ! иначе установить ссылку на англоязычное окно
```

```
END
```

```
OPEN(AppRef)              !Открыть указанное окно
```

```
IF SomeCondition
```

```
  AnimalRef &= Cat         !Указатель на Cat
```

```
ELSE
```

```
  AnimalRef &= Bird        ! Указатель на Bird
```

```
END
```

```
AnimalRef.Feed(10)
```

**Смотри также:** операторы присвоения указателя, TREAD, CLASS

## Атрибуты переменных

### PRE (объявить префикс)

PRE( [ префикс] )

**PRE** Для составных структур данных задать префикс.  
 префикс Допустимы буквы, цифры от 0 до 9 и знак подчеркивания.  
 Начинаться префикс должен с буквы. По традиции префикс  
 состоит из 1-3 символов, хотя может быть и длиннее.

Этот атрибут обеспечивает префикс для составных структур данных. Он используется для того, чтобы различать одноименные переменные в различных структурах. При использовании в исполняемых операторах, операторах присваивания и в списках параметров префикс присоединяется к имени переменной с помощью двоеточия (префикс:имя). Из рассматриваемых в этой главе структур атрибут PRE используется со структурами GROUP и LIKE.

Для идентификации одноименных переменных, которые могут быть объявлены в структурах, не имеющих атрибута PRE, используется другой, более гибкий метод - синтаксис уточнения имен. При упоминании в исполняемых операторах, присвоениях и списках параметров к имени переменной через точку спереди присоединяется имя содержащей данное поле структуры (GroupName.Label).

#### Пример:

```
G1      GROUP,PRE(Mem)      !Объявить некие переменные
Message  STRING(30)      ! с префиксом Mem
Page     LONG
Line     LONG
Device   STRING(30)
END
G2      LIKE(G1),PRE(Me2)    !Объявить вторую группу, подобную первой
        ! содержащую переменные с префиксом Me2
CODE
Mem:Message = 'Variable in original group'      !Используя префикс
G1.Message = 'Same Variable in original group'  !Используя синтаксис уточнения
имен
Me2.Message = 'Variable in LIKE group'
G2.Message = 'Same Variable in LIKE group'
```

**Смотри также:** Зарезервированные слова, Синтаксис уточнения имен

## DIM (установить размерность массива)

**DIM**(размерность,...,размерность)

**DIM**                                   Объявить переменную-массив.  
размерность                   Числовая константа, которая указывает число элементов в этом измерении массива.

Атрибут DIM объявляет переменную как массив. Переменная повторяется столько раз, сколько указывает параметр размерность. Многомерные массивы можно рассматривать как вложенные. Каждое измерение массива имеет свой индекс. Таким образом ссылка на элемент трехмерного массива требует указания трех индексов. На число измерений ограничения нет, однако общий размер массива в 16-ти разрядных приложениях не может превышать 65520 байт (в 32-х разрядных ограничений нет).

Индексы уточняют, к какому элементу массива происходит обращение. Список индексов содержит индекс для каждого измерения массива. Каждый индекс отделяется запятой, а весь список заключается в квадратные скобки ([]). Индекс может быть представлен числовой константой, выражением или функцией. К целому массиву можно обратиться по имени без указания списка индексов.

Структура GROUP представляет собой особый случай. Каждый уровень вложенности добавляет индекс группе и переменным внутри группы. К данным, объявленным внутри группы можно обращаться точно также как самой группе.

### Пример:

Scr	GROUP	!Символы на экране
Row	GROUP,DIM(25)	! двадцать пять строк
Pos	GROUP,DIM(80)	! две тысячи позиций
Attr	BYTE	!Байт атрибутов
Char	BYTE	!Байт символа
	...	!Завершение структур GROUP

В приведенной выше группе:

Scr	группа из 4000 байт
Attr[1.1]	байт
Char[1.1]	байт

Month	STRING(10),DIM(12)	!Двенадцать месяцев
	CODE	
	CLEAR(Month)	!Очистить весь массив
	Month[1] = 'Январь'	!Присвоить значения элементам массива
	Month[2] = 'Февраль'	
	Month[3] = 'Март'	

**Смотри также:** функция MAXIMUM

**EXTERNAL (переменная определена во внешней библиотеке)****EXTERNAL****EXTERNAL**

Указывает, что переменная определена во внешней библиотеке.

Атрибут EXTERNAL указывает что переменная, к которой он относится, определена во внешней библиотеке. Поэтому переменная с атрибутом EXTERNAL объявляется и может использоваться в Clarion-программе, но память для нее не выделяется. Память для такой переменной выделяется во внешней библиотеке. Этот атрибут позволяет Clarion-программе получить доступ к переменным, объявленным во внешней библиотеке как “public” - общие.

Атрибут EXTERNAL допустим только для переменных, объявленных вне структур FILE, QUEUE или GROUP.

Объявления переменных во всех библиотеках (или .EXE - модулях), которые ссылаются на общие переменные, должны быть в точности одинаковыми (с соответствующим добавлением атрибута EXTERNAL). Если объявления отличаются, то может произойти разрушение данных. Ответственность за соблюдение идентичности объявлений лежит на программисте, поскольку ни компилятор не компоновщик не могут определить несоответствия объявлений в различных программах и библиотеках.

При использовании атрибута EXTERNAL для объявления переменной совместно используемой несколькими библиотеками (.OBJ, .LIB, .DLL и .EXE) только в одной из них переменная должна объявляться без атрибута EXTERNAL. Во всех других библиотеках и программах следует объявлять эту переменную с атрибутом EXTERNAL. Это обеспечит уверенность в том, что для переменной распределена только одна область памяти и во всех библиотеках и программах при обращении к ней будут ссылки на одну и ту же область памяти.

Можно посоветовать при разработке больших систем, использующих много библиотек DLL и/или EXE модулей, которые совместно используют одни и те же переменные, собирать реальные описания разделяемых переменных и файлов в одну библиотеку DLL. Таким образом создается одна “главная” библиотека DLL, в которой происходит отслеживание всех действительных объявлений переменных. Эта главная библиотека связывается со всеми программами, которые используют общие переменные. Во всех других библиотеках и программах в этой системе общие переменные должны объявляться с атрибутами EXTERNAL и DLL.

**Пример:**

TotalCount LONG,EXTERNAL !Переменная, объявленная во внешней библиотеке

**Смотри также:** атрибут NAME, DLL.

## DLL (переменная определена в библиотеке .DLL)

### DLL( [ флаг ] )

**DLL**                    Указывает, что переменная объявлена в библиотеке .DLL.  
**флаг**                   Числовая константа, метка соответствия, или определение системы поддержки проекта, которое задает активен ли данный атрибут. Если флаг установлен в 0, то этот атрибут неактивен, как если бы его вообще не было. Если флаг имеет отличное от нуля значение, то атрибут активен.

Атрибут DLL указывает, что переменная, к которой он относится объявлена в библиотеке .DLL. Переменная с атрибутом DLL должна иметь еще и атрибут EXTERNAL. Для 32-битовых приложений атрибут DLL обязателен, так как такие библиотеки являются настраиваемыми (перемещаемыми) в 32-битовом адресном пространстве, которое требует от компилятора еще одного дополнительного разыменовывания (преобразования адреса) при обращении к переменной. Атрибут DLL допустим только для переменных, объявленных вне структур FILE, QUEUE или GROUP.

Объявления переменных во всех библиотеках (или .EXE - модулях), которые ссылаются на общие переменные, должны быть в точности одинаковыми (с соответствующим добавлением атрибута EXTERNAL и DLL). Если объявления отличаются, то может произойти разрушение данных. Ответственность за соблюдение идентичности объявлений лежит на программисте, поскольку ни компилятор не компоновщик не могут определить несоответствия объявлений в различных программах и библиотеках.

При использовании атрибутов EXTERNAL и DLL для объявления переменной совместно используемой несколькими библиотеками (.OBJ, .LIB, .DLL и .EXE) только в одной из них переменная должна объявляться без этих атрибутов. Во всех других библиотеках и программах следует объявлять эту переменную с атрибутами DLL и EXTERNAL. Это обеспечит уверенность в том, что для переменной распределена только одна область памяти и во всех библиотеках и программах при обращении к ней будут ссылки на одну и ту же область памяти.

Можно посоветовать при разработке больших систем, использующих много библиотек DLL и/или EXE модулей, которые совместно используют одни и те же переменные, собирать реальные описания разделяемых переменных и файлов в одну библиотеку DLL. Таким образом создается одна “главная” библиотека DLL, в которой происходит отслеживание всех действительных объявлений переменных. Эта главная библиотека связывается со всеми программами, которые используют общие переменные. Во всех других библиотеках и программах в этой системе общие переменные должны объявляться с атрибутами EXTERNAL и DLL.

### Пример:

TotalCount LONG,EXTERNAL,DLL(dll\_mode) ! Переменная, объявленная во внешней

библиотеке

Смотри также: EXTERNAL

### NAME (указать внешнее имя)

NAME([ константа])  
переменная

NAME	Указывает внешнее имя, используемое компоновщиком или файловым драйвером.
константа	Строковая константа.
переменная	Метка строковой переменной, объявленной в секции глобальных данных или секции объявления данных member-модуля.

Атрибут NAME задает “внешнее” имя переменной, используемое компоновщиком или файловым драйвером.. Он полностью независим от атрибута EXTERNAL - между ними нет никакой связи, хотя оба эти атрибуты могут использоваться с одной переменной.

Атрибут NAME может находиться в прототипе процедуры или функции, в операторах FILE, KEY, INDEX, MEMO, в объявлении любого поля в структуре FILE, в объявлении любого поля в структуре QUEUE или в объявлении любой отдельной переменной. Этот атрибут имеет различное толкование в зависимости от того, где он применяется.

В прототипах процедур и функций может использоваться форма NAME(константа). Константа представляет собой внешнее имя, используемое компоновщиком для того, чтобы идентифицировать процедуру или функцию из внешней библиотеки.

Атрибут NAME в любой из двух форм, относящийся к объявлению файла, указывает DOS-спецификацию файла. Если константа или переменная не содержит обозначения диска и каталога, то подразумеваются текущий диск и каталог. Если опущено расширение, то подразумевается используемое по умолчанию для данного файлового драйвера. Для некоторых файловых драйверов предполагается, что ключи, индексы или MEMO поля находятся в отдельных файлах. Поэтому атрибут NAME может также употребляться в операторах KEY, INDEX или MEMO. Атрибут NAME без параметров предполагает в качестве имени метку оператора, в котором он используется (включая любой указанный префикс).

При объявлении любого поля в структуре RECORD может использоваться атрибут в форме NAME(константа). В этом случае он обеспечивает файловый драйвер именем поля, как это, возможно, принято в файловой системе данного драйвера.

Эта же форма - NAME(константа) может использоваться для любого поля, объявленного в структуре QUEUE, что обеспечивает возможность динамических сортировок во время

выполнения программы.

Атрибут в форме NAME(константа) может использоваться и для любой переменной, объявленной вне какой-либо структуры. В этом случае атрибут обеспечивает внешнее имя, используемое компоновщиком для того, чтобы идентифицировать переменную, объявленную во внешней библиотеке. Если переменная к тому же имеет атрибут EXTERNAL, то она объявляется как общая (public) переменная во внешней библиотеке с соответствующим механизмом выделения памяти. Без атрибута EXTERNAL память переменной выделяется в Clarion-программе, а сама она объявляется как внешняя переменная во внешней библиотеке.

### Пример:

```

PROGRAM
MAP
MODULE('External.Obj')
AddCount  FUNCTION(LONG),LONG,C,NAME('_Addcount') !Функция на С именем
'_Addcount'
..

Cust      FILE,PRE(Cus),NAME(CustName) !Имя файла в переменной CustName
CustKey   KEY('Name'),NAME('c:\data\cust.idx')    !Объявить ключ cust.idx
Record    RECORD
Name      STRING(20),NAME    !По умолчанию внешнее имя 'Cus:Name'
..        !Конец объявлений

SortQue   QUEUE
Field1    STRING(10),NAME('FirstField') !Имя поля сортировки таблицы
Field2    LONG,NAME('SecondField') !Имя поля сортировки таблицы
.

CurrentCnt  LONG,EXTERNAL,NAME('Cur')    !Переменная объявлена как public во
внешней библиотеке
TotalCnt    LONG,NAME('Tot')              !Переменная объявлена как внешняя

```

**Смотри также:** прототипы процедур и функций, операторы FILE, KEY, INDEX, QUEUE, атрибут EXTERNAL.

## OVER (совместное использование памяти)

### OVER(переменная)

**OVER** Допустить обращение к одной области памяти двумя различными способами.  
 переменная Метка переменной, которая уже занимает ту память, которая должна совместно использоваться.

Атрибут OVER позволяет к одному участку памяти адресоваться двумя различными



способами. Переменная, объявленная с атрибутом OVER должна быть не больше той, поверх которой она объявляется (меньше - может быть).

Можно объявить переменную поверх переменной, которая является частью списка параметров, переданного в процедуру или функцию.

Переменная внутри структуры GROUP не может объявляться поверх переменной не входящей в эту структуру.

#### Пример:

```
CustNote  FILE,PRE(Csn)    !Объявить файл CustNote
Notes     MEMO(2000)      ! поле MEMO
Record    RECORD
CustID    LONG
```

```
..
CsnMemoRow  STRING(10),DIM(200),OVER(Csn:Notes) !К полю Csn:Notes можно
обращаться как к целому
            ! или к кусочкам по 10 байт
```

**Смотри также:** атрибут DIM

## STATIC (статическая локальная переменная)

### STATIC

Атрибут **STATIC** говорит о том, что переменной, объявляемой в процедуре или функции, должна распределяться статическая память, а не память в стеке. Этот атрибут позволяет любому значению, содержащемуся в такой переменной возобновляться при переходе от одной копии процедуры к другой.

#### Пример:

```
SomeProc  PROCEDURE
AcctFile  STRING(64),STATIC    !Атрибут STATIC нужен для того,
                               ! чтобы использовать эту переменную в атрибуте NAME
Transactions FILE,DRIVER('Clarion'),PRE(TRA),NAME(AcctFile)
AccountKey  KEY(TRA:Account),OPT,DUP
Record     RECORD
Account    SHORT
Date       LONG    !Дата транзакции
Amount     DECIMAL(14.2)    !Количество
```

**Смотри также:** объявление данных и распределение памяти

## THREAD (статическая переменная зависит от исполняемого процесса)

### THREAD

Атрибут THREAD указывает, что статической переменной память распределяется отдельно для каждого исполняемого процесса в программе. Таким образом значение этой переменной зависит от того, какой процесс выполняется. Как только начат новый исполняемый процесс, для него создается и инициализируется пробелами или нулем новая копия этой переменной (если только не указан атрибут AUTO).

Переменной с атрибутом THREAD должна выделяться статическая память, поэтому локальные данные с таким атрибутом автоматически рассматриваются как статические.

Этот атрибут влечет за собой дополнительные издержки времени выполнения, особенно для глобальных данных и данных модуля. Поэтому его следует использовать только когда это абсолютно необходимо.

#### Пример:

GlobalVar LONG,THREAD !Каждый исполняемый процесс получает свою копию

SomeProc PROCEDURE

LocalVar LONG,THREAD !Локальная переменная отдельная для каждого процесса

**Смотри также :** START, объявление данных, распределение памяти, STATIC, AUTO

## BINDABLE (переменная, используемая в динамических выражениях)

### BINDABLE

Атрибут BINDABLE объявляет структуры GROUP, QUEUE, FILE и VIEW, составляющие переменные которых можно использовать в динамических выражениях во время выполнения программы. Значение атрибута NAME каждой из переменных является логическим именем, используемым в динамических выражениях. Если атрибут NAME отсутствует, то используется имя переменной (включая префикс). Для имен всех переменных структуры в исполняемом модуле выделяется память. Таким образом программа становится больше и использует больше памяти чем обычно. Поэтому атрибут BINDABLE следует использовать, только если большая часть переменных структуры используется в динамических выражениях.

#### Пример:

FileNames GROUP,BINDABLE!Группа переменные которой могут использоваться  
! в динамических выражениях

FileName STRING(8),NAME('FILE') !Динамическое имя FILE

Dot STRING('.') !Динамическое имя: Dot

Extension STRING(3),NAME('EXT') !Динамическое имя: EXT

END

**Смотри также:** BIND, UNBIND, EVALUATE

### **AUTO (локальная переменная без начального значения)**

#### **AUTO**

Атрибут AUTO разрешает распределение переменной, объявленной внутри процедуры или функции, неинициализированной памяти в стеке. При распределении памяти во время выполнения программы, числовой переменной без атрибута AUTO присваивается начальное значение 0, а строковая переменная инициализируется пробелами.

Атрибут AUTO используется когда не нужно быть уверенным в нулевом или пробельном начальном значении переменной, поскольку вы намереваетесь присвоить ей некое отличное от нуля или пробелов значение. Этот атрибут экономит небольшое количество памяти во время выполнения, исключая фрагмент программного кода, необходимый для автоматической инициализации переменной.

#### **Пример:**

```
SomeProc PROCEDURE
SaveCustID LONG,AUTO !Неинициализированная локальная переменная
```

### **TYPE (определение типа группы или класса)**

#### **TYPE**

Атрибут TYPE создает для группы или класса “определение типа”. Определение типа может использоваться в операторах LIKE для описания других таких же групп или классов.

Группе с атрибутом TYPE память не распределяется. Данным для класса или группы с атрибутом TYPE память тоже не выделяется.

#### **Пример:**

```
PassGroup GROUP,TYPE !Определение типа для группы, передаваемой как параметр
F1 STRING(20) ! первое поле
F2 STRING(1)! второе поле
F3 STRING(20) ! последнее поле
END
NameGroupLIKE(PassGroup),PRE(Nme) !Группа фамилия
```

## PRIVATE (переменные класса доступны только в пределах модуля)

### PRIVATE

Атрибут PRIVATE указывает, что переменная для которой он задан, доступна только для процедур и функций определенных внутри исходного модуля, содержащего методы этой структуры CLASS. Таким образом данные инкапсулируются от других классов.

#### Пример:

```
OneClass CLASS,MODULE('OneClass.CLW'),TYPE
PublicVar LONG !Объявить общую переменную
PrivateVar LONG,PRIVATE ! Объявить "личную" переменную
BaseProc PROCEDURE(REAL Parm) ! Объявить общий метод
END
TwoClass OneClass !Экземпляр класса OneClass
CODE
TwoClass.PublicVar = 1 !Правильное присвоение
TwoClass.PrivateVar = 1 !Неправильное присвоение
```

!Файл OneClass.CLW содержит:

```
MEMBER()
MAP
SomeLocalProc PROCEDURE
END

OneClass.BaseProc PROCEDURE(REAL Parm)
CODE
SELF.PrivateVar = Parm ! Правильное присвоение

SomeLocalProc PROCEDURE
CODE
TwoClass.PrivateVar = 1 ! Правильное присвоение
```

## MODULE (указать модуль исходного текста для методов класса)

### MODULE(исходный\_файл)

**MODULE** Указать MEMBER-модуль или файл внешней библиотеки.  
**исходный\_файл** Строковая константа. Если исходный\_файл содержит исходный текст на языке Clarion, то эта константа содержит имя файла (без расширения), в котором находятся процедуры и/или функции. Если исходный\_файл представляет собой внешнюю библиотеку, то эта строка должна содержать произвольный уникальный идентификатор.

Атрибут MODULE структуры CLASS указывает имя MEMBER-модуля или внешней библиотеки, в которой содержатся определение процедур и функций для методов данного

класса. Атрибут MODULE допустим только для структуры CLASS.

**Пример:**

```
OneClass CLASS,MODULE('OneClass') !Определение методов находится в OneClass.CLW
LoadIt    PROCEDURE    !Прототип процедуры LoadIt
ComputeIt PROCEDURE    !Прототип процедуры ComputeIt
END
```

**See Also:** CLASS, MEMBER, FUNCTION, прототипы процедур и функций

## **Объявление данных и распределение памяти**

### **Глобальные, локальные, статические и динамические данные**

---

Операторы объявления данных автоматически распределяют память для хранения значений данных. Термины “глобальные, локальные, статические и динамические данные” описывают тип выделяемой памяти.

Термины глобальные и локальные данные говорят об области распространения данных.

- \* Глобальные данные доступны в любой процедуре программы.
- \* Локальные имеют ограниченную доступность. Их область распространения может ограничиваться рамками одной процедуры или функции или пределами нескольких процедур и функций (в одном исходном модуле)

Термины статические и динамические данные указывают на момент времени, когда для данных выделяется место в памяти.

- \* “Статический” означает, что данным распределяется память, которая освобождается только после завершения выполнения всей программы в целом.
- \* “Динамический” - означает, что данным память выделяется в программном стеке. Память в стеке освобождается, когда процедура или функция распределившая стек возвращает управление в точку из которой к ней произошло обращение.

### **Разделы объявления данных**

---

В программе на языке Clarion существует три области, в которых можно объявить данные:

- \* В программном модуле после ключевого слова PROGRAMM и до оператора CODE. Это секция глобальных данных.
- \* В member-модуле после ключевого слова MEMBER и до первого оператора PROCEDURE или FUNCTION. Это секция данных модуля.

\* В процедуре или функции после ключевого слова PROCEDURE (или FUNCTION) и до оператора CODE. Это секция локальных данных.

Глобальные данные доступны для использования в исполняемых операторах и выражениях в любой процедуре и функции этой программы. Глобальные данные распределяются в статической памяти.

Данные модуля доступны для использования в исполняемых операторах и выражениях только в процедурах и функциях, содержащихся в данном модуле. Если нужно, они могут передаваться в качестве параметров в процедуры и функции в другие модули. Данные модуля тоже распределяются в статической памяти.

Локальные данные доступны в пределах процедуры или функции, в которой они объявлены. Их можно передавать в качестве параметров в любые другие процедуры или функции. Локальным данным выделяется динамическая память. Такая память выделяется в области программного стека для переменных размер которых меньше для ограничения для стека (по умолчанию меньше 5К), в противном случае память для них выделяется из “кучи”. Механизм распределения памяти можно изменить, используя атрибут STATIC, тем самым делая возможным существование значения этой переменной и в промежутке между обращениями к процедуре или функции.

Динамическое распределение памяти для локальных данных делает процедуры и функции истинно рекурсивными, получающими при каждом обращении к ним новые копии локальных переменных.

**Смотри также:** Прототипы процедур и функций, STATIC, THREAD.

## NEW (выделить память из кучи)

указатель &= NEW( тип\_данных )

указатель	Метка переменной-указателя, которая соответствует типу_данных.
<b>NEW</b>	Создать новый экземпляр типа_данных в куче.
тип_данных	Метка ранее объявленной структуры CLASS или QUEUE, или оператор объявления любого простого типа данных. Здесь не может быть имени составного типа данных (объявление которого должно заканчиваться оператором END).

Оператор NEW новый экземпляр типа\_данных в куче. Ключевое слово NEW допустимо только справа от оператора присваивания указателя. Память должна освобождаться явно, оператором DISPOSE.

### Пример:

StringRef &STRING !Указатель на любую переменную типа STRING

```

LongRef    &LONG    ! Указатель на любую переменную типа LONG

Animal     CLASS
Feed       PROCEDURE(short amount)
Weight     LONG
           END
AnimalRef  &Animal   ! Указатель на любой класс типа Animal

NameQ      QUEUE
Name       STRING(30)
           END
QueRef     &NameQ    ! Указатель на любую очередь из STRING(30)

CODE
AnimalRef  &= NEW(Animal)    !Создать новый экземпляр класса Animal

QueRef     &= NEW(NameQ)      ! Создать новый экземпляр очереди NameQ

StringRef  &= NEW(STRING(50)) ! Создать новую переменную STRING(50)

LongRef    &= NEW(LONG)       !Создать новую переменную типа LONG

```

## DISPOSE (освободить память в куче)

**DISPOSE**( указатель )

**DISPOSE**                      Освободить память в куче, ранее распределенную оператором NEW  
указатель                      Имя переменной-указателя ранее использовавшегося в операторе  
                                 присвоения указателя с оператором NEW.

Оператор DISPOSE освобождает память в куче, ранее распределенную оператором NEW.

### Пример:

```

StringRef  &STRING    !Указатель на переменную типа STRING

Animal     CLASS,TYPE
Feed       PROCEDURE(short amount),VIRTUAL
Weight     LONG
           END
AnimalRef  &Animal     !Указатель на класс Animal

NameQ      QUEUE
Name       STRING(30)
           END
QueRef     &NameQ      !Указатель на очередь из STRING(30)

```

**CODE**

```
AnimalRef &= NEW(Animal)    !Создать новый экземпляр класса Animal
DISPOSE(AnimalRef)         !освободить память класса Animal
```

```
QueRef &= NEW(NameQ)        !Создать новую очередь NameQ
DISPOSE(QueRef)            !Освободить память, занимаемую очередью
```

```
StringRef &= NEW(STRING(50)) !Создать переменную STRING(50)
DISPOSE(StringRef)         !Освободить память, занимаемую переменной
```

```
STRING(50)
```

## Шаблоны

Шаблоны обеспечивают заданный формат представления значений отображаемых и редактируемых переменных. Существует семь типов шаблонов: числовые и денежные, научной записи чисел, даты, времени, шаблоны пользователя, шаблоны редактирования вводимых строк и строковые шаблоны.

### Числовые и денежные шаблоны

**@N** [валюта][знак][заполнение] размер [символ разбиения]  
[длина дробной части][знак][валюта][**B**]

**@N** Все числовые и денежные шаблоны начинаются @N  
валюта Или знак доллара или заключенная в тильды (~) строковая константа.  
Когда отсутствует индикатор заполнения и обозначение валюты предшествует знаку, тогда обозначение валюты “плавает” слева от старшей значащей цифры. Если есть индикатор заполнения, то обозначение валюты постоянно находится в крайней левой позиции. Если обозначение валюты следует после размера и символа разбиения, то оно ставится в конце выводимого числа.

знак Указывает формат отрицательных чисел. Если минус в шаблоне стоит до размера и индикатора заполнения, то отрицательные числа будут отображаться с минусом перед числом. Если минус в шаблоне стоит после размера, символа разбиения, индикаторов длины дробной части и валюты, то отрицательные числа будут отображаться с минусом после числа. Если в обеих возможных позициях знака в шаблоне стоят круглые скобки, то отрицательные числа будут отображаться в скобках.

заполнение Указывает заполнение позиций незначащих нулей пробелами, нулями или звездочками (\*). Если индикатор заполнения опущен, то незначащие нули подавляются (пробелами).

- 0 Подавляет разбиение по тысячам и оставляет незначащие нули.
- \_ Подавляет разбиение по тысячам и заменяет незначащие нули



пробелами.

- \* Заменяет незначащие нули звездочками.

**размер** Размер требуется для того, чтобы указать общее число значащих цифр, включая число цифр в дробной части и любые символы форматирования.

**символ разбиения** Для того, чтобы указать разбиение по трем цифрам (по тысячам), справа от размера может помещаться символ разбиения отличный от запятой, которая используется по умолчанию.

- . Разбивает группы точками
- Разбивает группы пробелами
- Разбивает группы дефисами

**длина дробной части** Указывает количество цифр в дробной части и символ, разделяющий дробную и целую части числа. Число цифр в дробной части должно быть меньше чем длина, обозначенная в индикаторе размера. Разделителем целой и дробной части может быть точка (.), одинарная кавычка (‘) или буква “v”, используемая только в объявлении переменных типа STRING, но не в форматах отображения.

- . Точка указывает, что разделителем целой и дробной частей является точка.

- ‘ Одиночная кавычка указывает, что разделителем целой и дробной частей является запятая (точка используется для разбиения на группы, если не задано другое).

- v Указывает, что нет разделителя целой и дробной частей (относится только к хранению данных в переменных типа STRING).

- B Указывает, что всякий раз, когда значение равно нулю, шаблон отображается пробельным.

Числовые и денежные шаблоны форматируют числовое значение для вывода на экран или в отчет. Если величина больше максимально возможной для данного шаблона, то выводится строка звездочек.

### Пример:

Шаблон	Результат	Формат
@N9	4,550,000	Девять цифр разбиваются на тысячи запятой (по умолчанию)
@N_9B	4550000	Девять цифр, на тысячи не разбиваются, пробелы если 0
@N09	004550000	Девять цифр, незначащие нули не подавляются
@N*9	***45,000	Девять цифр, незначащие нули заменяются звездочками,
разбиваются по три цифры запятой		
@N9_	4 550 000	Девять цифр разбиваются на тысячи пробелом
@N9.	4.550.000	Девять цифр разбиваются на тысячи точкой

<u>С дробной частью</u>	<u>Результат</u>	<u>Формат</u>
@N9.2	4,550.75	Две цифры дробной части отделяются точкой
@N_9.2B	4550.75	Две цифры дробной части отделяются точкой, по 3 разряда не группируются, пробелы если значение равно 0.
@N_9'2	4550,75	Две цифры дробной части отделяются запятой
@N9.'2	4.550,75	Разделитель целой и дробной частей запятая, разбиение на тысячи точкой
@N9_'2	4 550,75	Разделитель целой и дробной частей запятая, разбиение на тысячи пробелом

<u>Со знаком</u>	<u>Результат</u>	<u>Формат</u>
@N-9.2B	-2,347.25	Минус перед числом, пробелы если 0
@N9.2-	2,347.25-	Минус после числа
@N(10.2)	(2,347.25)	Отрицательное значение в скобках

<u>В долларах</u>	<u>Результат</u>	<u>Формат</u>
@N\$9.2B	\$2,347.25	Знак доллара перед числом, пробелы если 0
@N\$10.2-	\$2,347.25-	Знак доллара перед числом, если отрицательное значение минус после числа
@N\$(11.2)	\$(2,347.25)	Знак доллара перед числом, отрицательное значение в скобках

<u>Различные валюты</u>	<u>Результат</u>	<u>Формат</u>
@N12_'2~ F~	1 5430.50 F	Франция
@N~L. ~12'	L. 1.430.050	Италия
@N~#~12.2	#1,240.50	Великобритания
@N~kr~12'2	kr1.430,50	Норвегия
@N~DM~12'2	DM1.430,50	Германия
@N12_'2~ mk~	1 430,50 mk	Финляндия
@N12'2~ kr~	1.430,50 kr	Швеция

<u>Шаблоны хранения данных</u>		
Variable1	STRING(@N_6v2)	!Объявить 6 байт для чисел без десятичной точки
	CODE	
Variable1 =	1234.56	!Присвоить значение, кот. будет храниться в файле
как	123456	
	SHOW(1,1,Variable1,@N_7.2)	!Вывести с десятичной точкой '1234.56'

## Шаблоны научной записи чисел

### @Emsn[B]

@E	Все шаблоны научной записи чисел начинаются с @E
m	Определяет общее число символов в формате, получающемся по данному шаблону.
s	Задаёт символ десятичной точки и символ разбиения мантииссы на группы из трех цифр, когда значение n больше 3.

.	(точка)	точка и запятая
..	(точка точка)	точка и точка
'	(верхняя кавычка)	запятая и точка
_.	(подчеркивание точка)	точка и пробел
n		Число цифр слева от десятичной точки.
B		Указывает, что всякий раз, когда значение равно нулю, шаблон отображается пробельным.

По шаблонам научной записи чисел форматируются очень маленькие или очень большие числа. Этот формат представляет собой десятичную смешанную дробь и степень десяти.

### Пример:

Формат	Значение	Результат
@E9.0	1,967,865	.20e+007
@E12.1	1,967,865	1.9679e+006
@E12.1B	0	
@E12.1	-1,967,865	-1.9679e+006
@E12.1	.0000000323	.2000e-008
@E12_.4	1,967,865	1 967.865e+003

## Шаблоны дат

@Dn [s] [B] [направление [диапазон] ]

@D	Все шаблоны дат начинаются с @D
n	Определяет номер формата даты. Существует 18 форматов представления дат. Ноль перед номером шаблона означает наличие незначащего нуля в числе или месяце.
s	Символ-разделитель между днем, месяцем и годом. По умолчанию в различных форматах представления дат между днем, месяцем и годом вставляется слэш (/). Альтернативные символы-разделители: <ul style="list-style-type: none"> <li>.</li> <li>-</li> <li>'</li> <li><u>  </u></li> <li>?</li> </ul> Точка, в формате используется точка Дефис, в формате используется дефис Кавычка, в формате используется запятая Знак подчеркивания, в формате используется пробел Задаёт национальный порядок даты и разделитель.
B	Указывает, что всякий раз, когда значение равно нулю, шаблон отображается пробельным.
направление	Правая или левая угловая скобка (> или <), которая указывает направление для параметра диапазон (> означает будущее, < означает прошлое). Допустимо только в шаблонах дат с изображением года из двух цифр.
диапазон	Целочисленная константа в диапазоне от 0 до 99, которая задает столетие для параметра направление. Допустимо только в шаблонах дат с изображением года из двух цифр. Если параметр опущен, то по умолчанию

принимается 80.

Даты могут храниться в числовых переменных (обычно LONG), переменных типа DATA (для совместимости с Btrieve) или в строковых переменных, объявленных с шаблоном даты. Даты, хранящиеся в числовых переменных, называются “стандартными датами Clarion”. Хранящееся значение представляет собой число дней прошедших с 28 декабря 1800 года. Согласно шаблону это число преобразуется в один из форматов даты.

Для шаблонов с изображением года двумя цифрами преобразование дат осуществляется с привлечением “интеллектуальной логики”. Для шаблонов дат, в которых отсутствуют параметры направление и диапазон подразумевается, что даты находятся в диапазоне “80 лет назад - 20 лет вперед”. Этот диапазон по умолчанию можно изменить с помощью параметров направление и диапазон. Параметр направление указывает, изменяет ли параметр диапазон границу 100-летнего диапазона в прошлом или в будущем. А противоположная граница диапазона принимает соответствующее значение, такое при котором диапазон составлял бы 100 лет и для любого года, изображенного двумя цифрами не возникало неоднозначности при определении остальных двух цифр года.

Например, шаблон @D1>60 задает нахождение соответствующего столетия для любого года в диапазоне 60 лет вперед - 40 лет назад. Если текущий год 1996-й (т.о. диапазон 1956-2056 г.г), и пользователь ввел “5/01/40”, то правильный год 2040 (1940 не попадает в диапазон правильных дат), а если пользователь ввел “5/01/60”, то дата приходится на 1960-й год.

### Пример:

Шаблон	Формат	Результат
@D1	mm/dd/yy	10/31/59
@D1>40	mm/dd/yy	10/31/59 !По умолчанию 1959
@D01	mm/dd/yy	01/01/95
@D2	mm/dd/yyyy	10/31/1959
@D3	mmm dd, yyyy	ОСТ 31, 1959
@D4	mmmmmmmm dd, yyyy	october 31, 1959
@D5	dd/mm/yy	31/10/59
@D6	dd/mm/yyyy	31/10/1959
@D7	dd mmm yy	31 OCT 59
@D8	dd mmm yyyy	31 OCT 1959
@D9	yy/mm/dd	59/10/31
@D10	yyyy/mm/dd	1959/10/31
@D11	yyymmdd	591031
@D12	yyyymmdd	19591031
@D13	mm/yy	10/59
@D14	mm/yyyy	10/1959
@D15	yy/mm	59/10
@D16	yyyy/mm	1959/10
@D17	Сокращенный формат даты, установленный в Windows	
@D18	Полный формат даты, установленный в Windows	

## Альтернативные разделители

@D1.	mm.dd.yy	Разделитель точка
@D2-	mm-dd-yyuy	Разделитель дефис
@D5_	dd mm yy	Разделитель пробел
@D6'	dd,mm,yyuy	Разделитель запятая

**Шаблоны времени****@Tn[s][B]**

@D	Все шаблоны времени начинаются с @T
n	Определяет номер формата времени. Существует 8 форматов представления времени. Ноль перед номером шаблона означает наличие незначащего нуля в часах или минутах.
s	Символ-разделитель. По умолчанию в различных форматах представления времени между часами, минутами и секундами вставляется двоеточие (:).
	Альтернативные символы-разделители:
.	Точка, в формате используется точка
-	Дефис, в формате используется дефис
'	Кавычка, в формате используется запятая
_	Знак подчеркивания, в формате используется пробел
?	Задаёт национальный разделитель в записи времени..

**B** Указывает, что всякий раз, когда значение равно нулю, шаблон отображается пробельным.

Время может храниться в числовых переменных (обычно LONG), переменных типа TIME (для совместимости с Btrieve) или в строковых переменных, объявленных с шаблоном времени. Время, хранящееся в числовых переменных, называются “стандартными временем”. Хранящееся значение представляет собой число сотых долей секунды, прошедших с полуночи. Согласно шаблону это число преобразуется в один из 6-ти форматов.

Форматы, содержащие алфавитно-цифровые символы (@T3, @T6) не могут использоваться для ввода времени с клавиатуры.

Форматы времени, использующие символ-разделитель '?' образуют запись времени с разделителем, указанным в файле COUNTRY.SYS DOS для заданного кода страны.

**Пример:**

<u>Шаблон</u>	<u>Формат</u>	<u>Результат</u>
@T1	hh:mm	17:30
@T2	hhmm	1730
@T3	hh:mmXM	5:30PM
@T03	hh:mmXM	05:30PM

@T4	hh:mm:ss	17:30:00
@T5	hhmmss	173000
@T6	hn:mm:ssXM	5:30:00PM
@T7		Сокращенный формат времени, установленный в Win-
dows		
@T8		Полный формат времени, установленный в Windows

#### Альтернативные разделители

@T1.	hh.mm	Разделитель точка
@T1-	hh-mm	Разделитель дефис
@T3_	hh mmXM	Разделитель пробел
@T4'	hh.mm.ss	Разделитель запятая

### Шаблоны пользователя

**@P[<][#][x]P[B]**

@P	Все пользовательские шаблоны начинаются с @P и заканчиваются разделителем P. Обе буквы P (латинские) должны быть либо прописные, либо строчные.
<	Задаёт десятичную цифру с подавлением незначащего нуля.
#	Задаёт десятичную цифру без подавления незначащего нуля.
x	Указывает необязательные символы для вывода в строке. Эти символы без изменений выводятся в отформатированной строке.
P	Все пользовательские шаблоны должны заканчиваться символом P. Он должен быть таким же как в сочетании @P, начинающем шаблон.
B	Указывает, что всякий раз, когда значение равно нулю, шаблон отображается пробельным.

Пользовательский шаблон содержит необязательные позиции десятичных цифр и необязательные символы редактирования. Любые символы отличные от < и # рассматриваются как символы редактирования и попадают в форматированную строку без изменений. Регистр разделителей @P и P (строчная или прописная буква) различается. Таким образом, если для обоих ограничителей используется прописная "P", то строчную "p" можно использовать как символ редактирования и наоборот.

Для того, чтобы допустить использование точки в качестве символа редактирования, в шаблонах пользователя не распознается десятичная точка. Поэтому, значение, формирующееся по шаблону пользователя должно быть целочисленным. Если по шаблону пользователя формируется величина с плавающей точкой, то в результате будет отражена только целая часть числа.

**Пример:**

<u>Шаблон</u>	<u>Значение</u>	<u>Результат</u>
@P###-##-####P	215846377	215-84-6377
@P<#/#/#P	103159	10/31/59
@P(###)###-####P	3057854555	(305)785-4555
@P###/###-####P	7854555	000/785-4555
@p<#:##PMp	530	5:30
@P<# '<#P	506	5' 6"
@P<#lb. <#oz.P	902	9lb. 2oz.
@P4##A-#P		112 411A-2
@PA##.C#P	312.45	A31.C2

**Шаблоны редактирования строк**

**@K[@][#][<][x][\][?][^][\_][|]K[V]**

@K	Все шаблоны редактирования строк начинаются с @K и заканчиваются разделителем K. Обе буквы K (латинские) должны быть либо прописные, либо строчные.
@	Устанавливает, что в строке только строчные и прописные буквы.
#	Задаёт десятичную цифру от 0 до 9.
<	Задаёт десятичную цифру с подавлением незначащего нуля.
x	Указывает необязательные постоянные символы для вывода в строке (любые отображаемые символы). Эти символы без изменений выводятся в отформатированной строке.
\	Показывает, что следом идет символ для вывода в строке. Позволяет использовать символы форматирования (@, #, <, \, ?, ^, _) в качестве символов, включаемых в строку-результат.
?	Указывает, что в данной позиции может помещаться любой символ.
^	Указывает, что в данной позиции могут помещаться только прописные буквы.
_	Указывает, что в данной позиции могут помещаться только строчные буквы.
	Позволяет при вводе данных пользователю “остановиться здесь”, если вводить в эту строку больше нечего. В отформатированную строку будут помещены только введенные данные и символы для вывода в строке, указанные до точки в шаблоне, отмеченной символом ( )
K	Все шаблоны редактирования строк должны заканчиваться символом K. Он должен быть таким же как в сочетании @K, начинающем шаблон.
V	Указывает, что всякий раз, когда значение равно нулю, шаблон отображается пробельным.

Шаблоны редактирования строк могут содержать позиции десятичных цифр (# <), позиции букв (@ ^ \_), позиции, в которых могут находиться любые символы (?) и символы для вывода в строке. Любой символ отличный от символов форматирования

рассматриваются как символ для вывода в строке, которые без изменений выводятся в отформатированной строке. Регистр разделителей @К и К (строчная или прописная буква) различается. Таким образом, если для обоих ограничителей используется прописная “К”, то строчную “к” можно использовать как символ для вывода в строке и наоборот.

Шаблоны редактирования строк используются с полями типа STRING, PSTRING и CSTRING для того, чтобы обеспечить пользователю управление редактированием поля и проверку правильности данных. Использование шаблонов редактирования строк, содержащих любой из буквенных символов редактирования (@ ^), с числовым полем ввода приводит к непредсказуемым результатам.

Использование режима вставки при вводе в поле с таким шаблоном тоже могло бы привести к непредсказуемым результатам. Поэтому, даже если для поля указан атрибут INS, ввод данных в поле происходит в режиме замещения.

### Пример:

Шаблон	Вводимое значение	Результат
@K###-##-####K	215846377	215-84-6377
@K##### #####K	33064	33064
@K##### #####K	330643597	33064-3597
@K<# ^^ ^ ##K	10AUG59	10 AUG 59
@K(###)@@@-##\@##K	305abc4555	(305)abc-4555
@K###/?##-####K	7854555	000/785-4555
@k<#:##^Mk	530P	5:30PM
@K<#’ <#”K	506	5’ 6”
@K4#_#A-#K	1g12	41g1A-2

## Строковые шаблоны

### @Sдлина

@S	Все строковые шаблоны начинаются с @S
длина	Определяет число символов в строке

Строковый шаблон описывает неформатированную строку заданной длины.

### Пример:

Name      STRING(@S20)    !Строковое поле длиной 20 символов



## Директивы компилятора

### EQUATE (присвоить метку)

	метка
метка <b>EQUATE</b> ( константа )	
	шаблон
	тип

метка	Метка любого оператора, предшествующего оператору EQUATE. Эта форма используется для того, чтобы объявить альтернативную метку оператора
константа	Числовая или строковая константа. Эта форма используется для того, чтобы объявить сокращенную метку для значения константы. Этот прием, облегчает нахождение и изменение константы.
шаблон	Шаблон. Эта форма используется для того, чтобы объявить сокращенную метку для шаблона. Однако форматы экрана и отчета в редакторе Clarion не воспринимают метку соответствия, как правильный шаблон.
тип	Тип данных. Обычно этот параметр используется для того, чтобы объявить единый способ объявления переменной как переменной одного из нескольких возможных типов в зависимости от установок параметров компиляции (подобно описанию typedef в C++ для простого типа данных).

Директива EQUATE не порождает какого-нибудь выделения памяти. Она используется для того, чтобы назначить метку другой метке или константе. Метка самой директивы не может совпадать с меткой - параметром директивы.

#### Пример:

```
Init      EQUATE(SetupProg)    !Установить метку-псевдоним
Off       EQUATE(0)            !Off означает 0
On        EQUATE(1)            !On означает 1
PI        EQUATE(3.1415927)    !PI есть метка значения пи
EnterMsg  EQUATE('Для того, чтобы сохранить нажмите Ctrl-Enter')
SocSecPic EQUATE(@P###-##-####P) !Метка шаблона
          OMIT('End16BitChk',Flag32Bit = 0)    !Опустить если выключена 32-
разрядная компиляция
SIGNED    EQUATE(LONG)         !SIGNED = LONG в случае 32-разрядной компиляции
End16BitChk
          OMIT('End32BitChk',Flag32Bit = 1)    !Опустить если включена 32-разрядная
компиляция
SIGNED    EQUATE(SHORT)        !SIGNED = SHORT в случае 16-разрядной компиляции
End32BitChk
```

**Смотри также:** Зарезервированные слова.

**SIZE (размер памяти в байтах)**

метка	метка <b>SIZE</b> (константа) шаблон
-------	--

метка	Метка ранее объявленной переменной.
константа	Числовая или строковая константа.
шаблон	Шаблон.

Директива **SIZE** предписывает компилятору подставить вместо директивы число байт памяти, используемой для хранения параметра директивы

**Пример:**

SavRec	STRING(1),DIM(SIZE(Cus:Record)	!Размерность массива равна длине записи
StringVar	STRING(SIZE('TopSpeed Corporation'))	!Длина строки равна длине константы.
LOOP I# = 1 TO SIZE(ParseString)		!Цикл по числу байтов в строке
PicLen = SIZE(@P(###)###-####P)		!Запомнить длину шаблона

## Глава 4 Выражения и операторы присваивания

### Выражения

Выражение - это математическая или логическая формула, по которой вычисляется значение. Выражение может стоять справа от знака равенства в операторах присваивания, быть параметром процедуры или функции, индексом массива переменных или условием в структурах IF, CASE, LOOP или EXECUTE. Выражения состоят из констант, переменных и функций, связанных символами логических и/или арифметических операций.

### Вычисление выражений

Выражения вычисляются в стандартном алгебраическом порядке операций. Старшинство операций определяется типом операции и скобками. Каждая операция вырабатывает промежуточное (внутреннее) значение, которое затем используется в последующих операциях. Скобки используются для того, чтобы группировать операции в выражении. Выражение вычисляется, начиная с подвыражения в самых вложенных скобках, последовательно к самому внешнему уровню.

Уровни старшинства операций в выражениях от наивысшего к наименьшему:

Уровень 1	()	Группировка скобками
Уровень 2	-	Унарный минус
Уровень 3	обращение	Получение возвращаемого значения. к функции
Уровень 4	^	Возведение в степень
Уровень 5	* / %	Умножение, деление, деление по модулю
Уровень 6	+ -	Сложение, вычитание
Уровень 7	&	Конкатенация (объединение)
Уровень 8	= <>	Логическое сравнение
Уровень 9	AND, NOT, OR	Булевы выражения

Результатом выражения может быть числовое, строковое или логическое (истина/ложь) значение. Выражение может вообще не содержать операций, это может быть просто переменная, константа или обращение к функции.

### Арифметические операции

Символ арифметической операции объединяет два операнда в арифметическое выражение для вычисления результата. Знаки операций:

+	сложение	(A + B означает A плюс B)
-	вычитание	(A - B означает A минус B)
*	умножение	(A * B означает A умножить на B)

/	деление	(A / B означает A разделить на B)
^	возведение в степень	(A ^ B означает A возвести в степень B)
%	Остаток от деления	(A % B означает остаток от деления A на B)

## Логические операции

Логическая операция проверяет два операнда и вырабатывает условие “правда” или “ложь”. Существует два вида логических операций: условные и булевы. В условных операциях сравниваются два значения или выражения. Символы булевых операций соединяют строковые, числовые или логические выражения, реализуя булеву алгебру. Символы булевых операций могут комбинироваться, образуя составные операции.

<b>Условные операции</b>	=	равно
	<	меньше
	>	больше
<b>Булевы операции</b>	NOT	булево (логическое) отрицание (НЕ)
	~	логическое НЕ
	AND	конъюнкция (булево И)
	OR	дизъюнкция (булево ИЛИ)
	XOR	исключающее ИЛИ

### Комбинированные операции

<>	не равно
~=	не равно
NOT =	не равно
<=	меньше или равно
=<	меньше или равно
~>	не больше
NOT >	не больше
>=	больше или равно
=>	больше или равно
~<	не меньше
NOT <	не меньше

Во время логических вычислений любая ненулевая величина означает условие “истина”, а пустая строка или нулевая величина - условие “ложь”.

### Пример:

Логическое выражение	Результат
$A = B$	Истина, если $A = B$
$A < B$	Истина, если $A$ меньше $B$
$A > B$	Истина, если $A$ больше $B$
$A <> B, A \sim B, A \text{ NOT} = B$	Истина, если $A$ не равно $B$
$A \leq B, A \geq B, A \text{ NOT} < B$	Истина, если $A$ не меньше $B$
$A \geq B, A \leq B, A \text{ NOT} > B$	Истина, если $A$ не больше $B$
$\sim A, \text{NOT } A$	Истина, если $A$ нуль или пустая строка
$A \text{ AND } B$	Истина, если $A$ истинно и $B$ истинно
$A \text{ OR } B$	Истина, если или $A$ истинно, или $B$ истинно, либо оба истинны
$A \text{ XOR } B$	Истина, если или $A$ истинно, или $B$ истинно, но не оба одновременно

## Числовые константы

Числовые константы представляют собой постоянные числовые значения. Константы могут встречаться в объявлении данных, выражениях, в качестве параметров процедур, функций или атрибутов. Числовая константа может представляться в десятичном (по основанию 10 - по умолчанию), двоичном (по основанию 2), восьмеричном (по основанию 8), шестнадцатеричном (по основанию 16) и экспоненциальном форматах. В числовых константах недопустимы символы форматирования, такие как знак доллара или запятая.

Десятичные (по основанию 10) числовые константы могут иметь впереди минус (как признак отрицательного числа), целую часть и необязательные десятичную точку с дробной частью числа. Двоичные (по основанию 2) числовые константы могут содержать необязательный знак минус спереди, символы цифр 0 и 1 и символ В или b (латинские) в конце. Восьмеричные числовые константы (по основанию 8) могут содержать необязательный знак минус спереди, символы цифр от 0 до 7 и символ О или o (латинские) в конце. Шестнадцатеричные (по основанию 16) числовые константы могут содержать необязательный знак минус спереди, символы цифр от 0 до 9, латинские буквы от А до F (представляющие цифры от 10 до 15) и символ Н или h (латинские) в конце. Если первой значащей цифрой (самым левым символом) в шестнадцатеричной константе является цифра от А до F, то перед ней надо поставить незначащий ноль.

### Пример:

-924	!Десятичные константы
76.346	
-45.0262	
1011b	!Двоичные константы

-1000110B !  
3403o !Восьмеричные константы  
-70413120O  
-1FFBh !Шестнадцатеричные константы  
0CD1F74FH

## Числовые выражения

---

Числовые выражения могут использоваться в качестве параметров процедур и функций, как условие в структурах IF, CASE, LOOP или EXECUTE, а также в правой части операторов присваивания, в которых слева стоит числовая переменная. Числовые выражения могут содержать операцию конкатенации, но в них не должно быть никаких логических операций. Исполненные в числовом выражении строковые константы и переменные преобразуются в промежуточное числовое значение. Если в выражении используется операция конкатенации, то она выполняется над строковыми промежуточными значениями, а результат преобразуется в число.

### Пример:

Count + 1	!Добавить 1 к переменной Count.
(1 - N * N) / R	!Вычесть из 1 произведение N на N затем
разделить на R	
305 & 7854555	!Соединить код региона с номером телефона

**Смотри также:** правила преобразования данных.

## Строковые константы

---

Строковые константы представляют собой строку символов, заключенную в одинарные кавычки (апострофы). Символы, которые нельзя ввести с клавиатуры, можно вставить в строку, заключая в угловые скобки (< >) их коды ASCII. Коды ASCII могут представляться десятичными или шестнадцатеричными числовыми константами. Максимальная длина строковой константы равна 255-ти символам.

Появление в строковой константе левой угловой скобки (<) подразумевает наличие соответствующей правой скобки. Поэтому для того, чтобы включить в строку символ угловой скобки, его необходимо ввести подряд дважды. Аналогично нужно ввести два апострофа подряд, чтобы в строку был включен апостроф. Константа, состоящая всего из двух апострофов (или с пробелами между ними) представляет пустую строку).

Последовательное повторение одного и того же символа в строке может указываться коэффициентом повторения. Коэффициент повторения указывается в фигурных скобках { } следом за символом, который должен повторяться. Для того, чтобы включить левую фигурную скобку в качестве части строковой константы, требуется ввести две левые

фигурные скобки подряд.

### Пример:

'string constant'	!Строковая константа
'It"s a girl'	!Вложенный апостроф
'<27,15>	!Десятичные коды ASCII
'<0Eh>	!Шестнадцатеричный код ASCII
'*{20}'	!Двадцать звездочек, запись с коэффициентом повторения
''	!Пустая строка

## Операция конкатенации

Операция конкатенации (&) используется для того, чтобы присоединить одну строку или переменную к другой. Длина результата равна сумме длин соединяемых строк. Могут соединяться и числовые константы или переменные со строкой или с другой числовой константой или переменной. Для того, чтобы удалить ненужные пробелы в конце соединяемых строк, используется функция CLIP.

### Пример:

CLIP(FirstName) & ' ' & Initial & ' ' & LastName	!Составить полное имя
'TopSpeed Corporation' & ' . Inc.'	!Соединить две константы

**Смотри также:** функция CLIP, числовые выражения, правила преобразования данных

## Строковые выражения

Строковые выражения могут использоваться в качестве параметров процедур и функций, либо в правой части операторов присваивания, если в левой части стоит строковая переменная. Строковым выражением может быть отдельная строка или числовая переменная, или сложное сочетание подвыражений, функций и операций.

### Пример:

StringVar	STRING(30)	
Name	STRING(10)	
Weight	STRING(3)	
Phone	LONG	
	CODE	
StringVar='Address:' & Cus:Address		!Конкатенация константы и переменной
StringVar = 'Phone:' & ' 305-' & FORMAT(Phone,@P###-####P)		!Конкатенация констант и значения, возвращаемого функцией FORMAT
StringVar=Weight & 'lbs.'		!Конкатенация константы и переменной

## Неявно объявленные строковые массивы и части строк

---

Дополнительно к явному объявлению все символьные строки неявно объявляются как массив из односимвольных строк. Такое неявное объявление эквивалентно объявлению второй переменной:

```
StringVar  STRING(10)
StringArray STRING(1),DIM(SIZE(StringVar)),OVER(StringVar)
```

Неявное объявление позволяет адресовать каждый символ строки как элемент массива, не требуя второго оператора объявления. Байт длины строки PSTRING адресуется как нулевой элемент массива (это единственный случай когда в Clarion допустима ссылка на нулевой элемент массива).

В случае, если оператор STRING уже имеет атрибут DIM, то это неявное объявление массива представляет собой самый последний (необязательный) уровень индексов (справа от объявленных явно). Функция MAXIMUM не работает с неявным уровнем размерности, вместо нее следует использовать функцию SIZE.

Кроме того, можно непосредственно адресоваться к нескольким символам внутри строки, используя технологию “частей строки”. Эта технология выполняет действия подобные функции SUB, только гораздо более гибкая и эффективная. Более гибкая потому, что “часть строки” может использоваться в операции присваивания с обеих сторон от знака равно (=), а функция SUB может использоваться только в качестве источника данных. А более эффективна потому, что требует меньших затрат памяти, чем присваивание отдельных символов или функция SUB.

Для того, чтобы взять “часть” строки, номера начального и конечного символов в этой части разделяются двоеточием и помещаются в квадратные скобки как индексы неявно объявленного массива. Номера символов могут быть целочисленными константами, переменными или выражениями. Если используются переменные, то между именами переменных и двоеточием должен быть по крайней мере один пробел, чтобы избежать путаницы с префиксами.

### Пример:

Name	STRING(15)	
CONTACT	STRING(15),DIM(4)	
	CODE	
	Name = 'Tammi'	!Присвоить значение
	Name[5] = 'y'	! затем изменить пятый символ
	Name[6] = 's'	! затем добавить шестой символ
	Name[0] = '<6>'	! и поменять байт длины



	Name[5:6] = 'ie'	! и изменить “часть” строки — пятый и шестой
СИМВОЛЫ	Contact[1] = 'First'	!Присвоить значение первому элементу
	Contact[1,2] = 'u'	!Изменить второй символ первого элемента
	Contact[1,2:3] = Name[5:6]	!Присвоить значение подстроке с 2-го по 3-й
СИМВОЛ		

**Смотри также:**    STRING, CSTRING, PSTRING

## Логические выражения

В логических выражениях в управляющих структурах IF, LOOP UNTIL, и LOOP WHILE происходит оценка логических условий “истина/ложь”. На основе окончательного результата выражения (истина или ложь) определяется дальнейшая последовательность выполнения программы. Логические выражения оцениваются слева направо. Правый операнд в операциях AND, OR или XOR оценивается только в том случае, когда он может повлиять на результат. Для того, чтобы избежать неоднозначности и управлять последовательностью оценки операндов, следует использовать скобки. Уровни старшинства логических операций:

Уровень 1	условные операции
Уровень 2	~, NOT
Уровень 3	AND
Уровень 4	OR, XOR

### Пример:

```

LOOP UNTIL KEYBOARD()      !Истинно, когда пользователь нажимает клавишу
                             '
                             ! операторы
END
IF A = B THEN RETURN.       !Выйти, если A = B
LOOP WHILE ~ DONE#         !Цикл пока “ложь” (DONE# = 0)
                             ! операторы
END
IF A >= B OR (C > B AND E = D) THEN RETURN.   !Истина если A >= B
! кроме того, истина если и C > B
! и E = D. Вторая часть выражения (после OR)
! оценивается только если первая часть “ложь”.

```

## **Строки динамических выражений**

В Clarion Database Developer for Windows есть возможность вычисления выражений на языке Clarion, динамически создаваемых во время выполнения. То есть в Clarion-программе возможно создание выражений “на ходу”. Также возможно позволить пользователю ввести выражение для вычисления.

Выражение представляет собой математическую или логическую формулу, в результате вычисления которой получается значение. Оно не является законченным оператором языка Clarion. Выражение может содержать только константы, переменные или обращения к функциям, соединенные символами логических и/или арифметических операций. Выражение может использоваться в правой части оператора присваивания, в качестве параметра процедуры или функции, в качестве индекса массива или в качестве условий в структурах IF, CASE, LOOP, или EXECUTE.

В качестве компонент строки динамического выражения можно использовать любые переменные Clarion-программы и большинство встроенных функций. Кроме того, в строке динамического выражения можно использовать написанные пользователем функции, которые отвечают конкретным определенным правилам, описанным в разделе, посвященном оператору BIND.

Во время выполнения программы в строке динамического выражения можно использовать все стандартные синтаксические правила конструирования выражений. Включая группировку скобками и все арифметические и логические операции и операции над строками. Динамические выражения вычисляются точно так же как любые другие выражения в языке Clarion и к ним применимы все стандартные правила старшинства операций, описанные в разделе Вычисление выражений.

### **Создание строки динамического выражения предполагает три этапа:**

- \* Переменные, которые можно использовать в динамических выражениях должны быть явно объявлены с атрибутом BIND.
- \* Динамическое выражение должно быть сконструировано. При этом возможно присоединение выбранных пользователем вариантов или непосредственный ввод пользователем его собственного выражения.
- \* Выражение передается функции EVALUATE, которая возвращает результат. Если выражение содержит синтаксическую ошибку, то устанавливается определенное значение функции ERRORCODE.

Как только выражение вычислено, его результат используется как если бы он был получен любым написанным вручную в программе выражением. Например, строка динамического выражения могла бы представлять фильтрующее выражение для исключения отдельных

записей при просмотре или печати базы данных ( выражение `FILTER` структуры `VIEW` подразумевает строку динамического выражения.

**BIND (объявить переменную для динамического выражения)**

<b>BIND</b> (	имя,переменная	)
	имя,функция	
	группа	

<b>BIND</b>	Идентифицирует переменные, которые могут использоваться в динамических выражениях.
имя	Строковая константа, содержащая идентификатор, используемый в динамическом выражении. Может совпадать с меткой переменной или функции.
переменная	Метка любой переменной (включая поля в структурах <code>FILE</code> , <code>GROUP</code> и <code>QUEUE</code> ) или передаваемого параметра. Если это массив, то только одномерный.
функция	Метка оператора <code>FUNCTION</code> языка <code>Clarion</code> , функции, которая возвращает значение типа <code>STRING</code> , <code>REAL</code> или <code>LONG</code> . Если функции передается параметры, то они должны быть строковыми параметрами-значениями (передаваемыми значением, а не адресом).
группа	Метка структуры <code>GROUP</code> , <code>RECORD</code> или <code>QUEUE</code> , объявленной с атрибутом <code>BINDABLE</code> .

Оператор **BIND** объявляет логическое имя, используемое для идентификации в строке динамического выражения переменной или написанной пользователем функции. До того, как переменную или пользовательскую функцию можно было использовать в строке динамического выражения в функции `EVALUATE` или в атрибуте `FILTER` структуры `VIEW`, ее нужно идентифицировать с помощью оператора `BIND`.

<b>BIND</b> (имя,переменная)	Указанное имя используется в выражении в качестве метки переменной.
<b>BIND</b> (имя,функция)	Указанное имя используется в выражении в качестве метки функции.
<b>BIND</b> (группа)	Объявляет, что все переменные в данной структуре <code>GROUP</code> , <code>RECORD</code> или <code>QUEUE</code> (с атрибутом <code>BINDABLE</code> ) можно использовать в динамическом выражении. В качестве логического имени в динамическом выражении используется значение атрибута <code>NAME</code> . Если атрибут <code>NAME</code> отсутствует, то используется имя переменной с префиксом.

Для структур `GROUP`, `QUEUE`, `FILE` и `VIEW`, объявленных с атрибутом `BINDABLE`, в исполняемом модуле для имен всех элементов данных этих структур дополнительно выделяется память. Таким образом программа становится больше и использует больше памяти чем обычно. Кроме того, большое число переменных, которые можно использовать

в динамических выражениях, замедляет работу функции EVALUATE. Поэтому форму BIND(группа) следует использовать, только если большая часть переменных, составляющих структуру, используется в динамических выражениях.

### Пример:

```

PROGRAM
MAP
  AllCapsFunc(String),String      !Функция на языке Clarion
END
Header      FILE,DRIVER('Clarion'),PRE(Hea),BINDABLE      !Объявление структуры файла
заголовков
OrderKey    KEY(Hea:OrderNumber)
Record      RECORD
OrderNumber                                LONG
ShipToName                                STRING(20)
StringVar   . .
            STRING(20)
            CODE
            BIND('ShipName',Hea:ShipToName)
            BIND('SomeFunc',AllCapsFunc)
            BIND('StringVar',StringVar)
            StringVar = 'SMITH'
            CASE EVALUATE('StringVar = SomeFunc(ShipName)')
            OF ''
              IF ERRORCODE()
                MESSAGE('Error ' & ERRORCODE() & ' — ' & ERROR())
              ELSE
                MESSAGE('Unkown error evaluating expression')
              END
            OF '0'
              DO NonSmithProcess
            OF '1'
              DO SmithProcess
            END
AllCapsFunc FUNCTION(PassedString)
            CODE
            RETURN(UPPER(PassedString))

```

**Смотри также:**    UNBIND, EVALUATE, PUSHBIND, POPBIND

**EVALUATE (получить результат динамического выражения)****EVALUATE**(выражение)**EVALUATE**    Вычислить динамическое выражение.

выражение    Строковая константа или переменная содержащая выражение для вычисления.

Функция EVALUATE возвращает результат вычисления динамического выражения в виде значения типа STRING. Если выражение синтаксически не верно с точки зрения языка Clarion, то возвращается пустая строка, а функция ERRORCODE возвращает код ошибки. Чтобы предотвратить появление сообщения компилятора об ошибке, в случае использования оператора “меньше чем” (<) его следует записать дважды (<<). Большое число переменных, связанных с логическими именами для использования в динамических выражениях, замедляет работу функции EVALUATE. Поэтому форму BIND(группа) следует использовать, только если большая часть переменных, составляющих структуру, действительно используется в динамических выражениях. А все неиспользуемые в данный момент в динамических выражениях переменные и пользовательские функции следует освободить оператором UNBIND.

Тип возвращаемого значения:    STRING

**Сообщения об ошибках:**

801	Неверное выражение
802	Переменная не найдена

**Пример:**

```

MAP
AllCapsFunc  FUNCTION(STRING),STRING  !Пользовательская функция
END
Header      FILE,DRIVER('Clarion'),PRE(Hea),BINDABLE      !Объявить структуру файла
заголовков
OrderKey    KEY(Hea:OrderNumber)
Record      RECORD
OrderNumber LONG
ShipToName  STRING(20)

StringVar   ..
            STRING(20)
            CODE
            BIND('ShipName',Hea:ShipToName)
            BIND('SomeFunc',AllCapsFunc)
            StringVar = 'SMITH'
            CASE EVALUATE('StringVar = SomeFunc(ShipName)')
            OF ''
                IF ERRORCODE()
                    MESSAGE('Error ' & ERRORCODE() & ' — ' & ERROR())
                END

```

```

        OF '0'
        DO NonSmithProcess
        OF '1'
        DO SmithProcess
    END
AllCapsFunc FUNCTION(PassedString)
    CODE
    RETURN(UPPER(PassedString))

```

**Смотри также:**     BIND, UNBIND, PUSHBIND, POPBIND

## **POPBIND (восстановить пространство имен динамических выражений)**

### **POPBIND**

Оператор **POPBIND** восстанавливает пространство имен предыдущего оператора BIND для ранее использованных в динамических выражениях функций и переменных. Таким образом восстанавливаются рамки предыдущих операторов BIND.

#### **Пример:**

```
SomeProc PROCEDURE
```

```

    OrderNumber                LONG
    Item        LONG
    Quantity    SHORT

```

```

    CODE
    BIND('OrderNumber',OrderNumber)
    BIND('Item',Item)
    BIND('Quantity',Quantity)

```

```
    AnotherProc                !Обратиться к другой процедуре
```

```

    UNBIND('OrderNumber',OrderNumber)
    UNBIND('Item',Item)
    UNBIND('Quantity',Quantity)

```

```
AnotherProc PROCEDURE
```

```

    OrderNumber                LONG
    Item        LONG
    Quantity    SHORT

```

```

    CODE
    PUSHBIND

```

!Создать новое пространство имен для

```
BIND
```

**Смотри также:** [PUSHBIND](#), [EVALUATE](#)

## PUSHBIND (очистить пространство имен динамических выражений)

## PUSHBIND

Оператор **PUSHBIND** освобождает пространство имен предыдущего оператора **BIND** для ранее использованных в динамических выражениях функций и переменных. Таким образом создается область для последующих операторов **BIND**, дающая возможность для новых операторов указывать переменные с теми же самыми именами, не вызывая конфликта с использованием этих переменных в предыдущих операторах.

### Пример:

UNBIND('Quantity',Quantity)

Quantity SHORT

```

CODE
PUSHBIND                                !Создать новое пространство имен для BIND
BIND('OrderNumber',OrderNumber)        !Указать Bind-переменные в новой области
BIND('Item',Item)
BIND('Quantity',Quantity)              !Выполнить некую обработку
UNBIND('OrderNumber',OrderNumber)
UNBIND('Item',Item)
UNBIND('Quantity',Quantity)
POPBIND                                !Восстановить предыдущее пространство имен

```

для BIND

**Смотри также:** POPBIND, EVALUATE

## UNBIND (освободить логическое имя)

**UNBIND( [имя] )**

**UNBIND** Освобождает переменные от использования в динамических выражениях. имя Строковая константа, которая указывает идентификатор использованный вычислителем динамических выражений. Если параметр опущен, то все переменные, использовавшиеся в динамических выражениях, освобождаются.

Оператор UNBIND освобождает логические имена, ранее связанные с переменными оператором BIND. Большое число переменных, которые можно использовать в динамических выражениях, замедляет работу функции EVALUATE. Поэтому все неиспользуемые в данный момент в динамических выражениях переменные и пользовательские функции следует освободить оператором UNBIND.

### Пример:

```

PROGRAM
MAP
  AllCapsFunc(STRING),STRING          !Пользовательская функция
END

Header  FILE,DRIVER('Clarion'),PRE(Hea)      !Объявить структуру файла заголовков
AcctKey  KEY(Hea:AcctNumber)
OrderKey KEY(Hea:OrderNumber)
Record   RECORD
AcctNumber          LONG
OrderNumber         LONG
ShipToName          STRING(20)
ShipToAddr          STRING(20)
ShipToCity          STRING(20)

```



**Смотри также:** [BIND](#), [EVALUATE](#), [PUSHBIND](#), [POPBIND](#)

## Операторы присваивания

## Простые операторы присваивания

метка назначение	Метка переменной
источник	Числовая или строковая константа, переменная, функция или выражение.

Знак '=' означает присваивание значения источника переменной, обозначенной меткой назначения. Если источник и назначение представляют собой различные типы данных, то результат получается на основе правил преобразования данных.

### Пример:

Name = 'JONES'                   !Переменная = строковая константа

PI = 3.14159 !Переменная = числовая константа  
 Cosine = SQRT(1 - sine \* sine) !Переменная = значение, возвращаемое функцией  
 A = B + C + 3 !Переменная = арифметическое выражение  
 Name = CLIP(FirstName) & ' ' Initial & ' ' & LastName  
 !Переменная = строковая переменная

**Смотри также:** правила преобразования данных

## Вычисляющие операторы присваивания

метка назначение	+=	источник
метка назначение	-=	источник
метка назначение	*=	источник
метка назначение	/=	источник
метка назначение	^=	источник
метка назначение	%=	источник

**источник** Числовая или строковая константа, переменная, функция или выражение.

**метка назначение** Это должна быть метка переменной. Не может быть любого типа свойством (окна, объекта в окне, отчета и т.п.)

Вычисляющие операторы присваивания выполняют арифметические действия над переменной. И хотя в приведенных ниже примерах операторы в левой и правой колонках функционально эквивалентны, вычисляющие операторы присваивания выполняются более эффективно.

### Пример:

Оператор присваивания	Функциональный эквивалент
A += 1	A = A + 1
A -= B	A = A - B
A *= -5	A = A * -5
A /= 100	A = A / 100
A ^= I + 1	A = A ^ (I + 1)
A %= 7	A = A % 7

## Операторы множественного присваивания

метка назначение	:=	источник
------------------	----	----------

**метка назначение** Метка структур данных GROUP, RECORD или QUEUE, или массива.

**источник** Метка структур данных GROUP, RECORD или QUEUE, числовая или строковая константа, переменная, функция или выражение.

Оператор := выполняет множественное присваивание, которое представляет собой несколько присваиваний значений отдельных компонент из одной структуры данных

другой. Эти присваивания выполняются только для переменных, которые имеют в точности совпадающие имена, без учета префиксов. Для того, чтобы выявить совпадающие имена переменных, компилятор просматривает и вложенные структуры GROUP. Значения любых переменных в структуре-назначении, которым не нашлось точно соответствующих по имени переменных в структуре-источнике, не изменяются.

Множественное присваивание выполняется совершенно также, как если бы каждой совпадающей по имени переменной индивидуально присваивалось значение соответствующей переменной. Это значит, что при присвоении значения совпадающей переменной действуют все обычные правила преобразования данных. Например, имя вложенной группы источника может соответствовать простой переменной или вложенной группе в структуре-назначении. В этом случае вложенная группа-источник присваивается назначению как строка, совершенно также как это происходит при обычном присвоении групп.

Могут совпадать и имена переменных, являющихся массивами. В этом случае значение каждого элемента массива-источника присваивается соответствующему элементу массива-назначения. Если число элементов массива-источника больше или меньше числа элементов массива-назначения, значения присваиваются только совпадающим элементам.

Если назначением является массив, который не является частью группы, структуры RECORD или QUEUE, а источник - это константа, переменная или выражение, то каждому элементу массива присваивается значение источника. Это более эффективный способ инициализации массива каким-либо конкретным значением, чем использование структуры LOOP и присвоение в цикле значения каждому элементу.

### Пример:

Group1	GROUP	
S	SHORT	
L	LONG	
	END	
Group2	GROUP	
L	SHORT	
S	REAL	
T	LONG	
	END	
ArrayField	SHORT,DIM(1000)	
	CODE	
	Group2 :=: Group1	!Эквивалентно: Group2:S = Group1:S и Group2:L =
Group1:L		
		! и выполняются все

необходимые преобразования данных

ArrayField :=: 7

!Эквивалентно:

```
! LOOP I# = 1 to 1000
!   ArrayField[I#] = 7
! END
```

## Операторы присваивания указателей

метка назначение &= источник

метка назначение      Метка переменной-указателя.  
источник                Метка другой переменной-указателя того типа что и назначение или метка переменной или структуры данных того типа, на которые может указывать переменная-назначение. Источник не может быть выражением, может быть только меткой данных.

Оператор &= выполняет операцию “присваивание указателя”, которая устанавливает значение переменной-указателя, указывающее на переменную-источник. В зависимости от типа данных переменной-указателю может присваиваться адрес памяти или адрес более сложной внутренней структуры данных (описывающей расположение и тип данных источника).

Операторы объявления переменной-указателя назначения и источника должны в точности совпадать; присваивание указателя не выполняет автоматического преобразования типов. Например, оператор “присваивание указателя” переменной-назначению, объявленной как &QUEUE должен иметь источником или другую переменную-указатель типа &QUEUE, или метку структуры QUEUE.

### Пример:

```
Queue1   QUEUE
ShortVar  SHORT
LongVar1  LONG
LongVar2  LONG
END

QueueRef &QUEUE           !Указатель только на QUEUE
LongRef  &LONG             !Указатель только на LONG
LongRef2 &LONG             ! Указатель только на LONG

CODE
QueueRef &= Queue1         !Присвоить указатель на Queue
IF SomeCondition           !Проверить некое условие
    LongRef &= Queue1:LongVar1 ! и установить указатель на соответствующую
переменную
ELSE
    LongRef &= Queue1:LongVar2
END
```

LongRef += 1

!Увеличить или LongVar1 или LongVar2  
! в зависимости от

установленного указателя

LongRef2 &amp;= LongRef

!Сделать второй указатель на те же данные

**CLEAR (очистить переменную)****CLEAR**(метка [,n])**CLEAR**

Очищает переменную

метка

Метка переменной

n

Числовая константа (1 или -1). Этот параметр указывает, что значение, которое заносится в переменную не равно 0 или пробелам. Если n равно 1, то устанавливается наибольшее возможное для данного типа переменной значение. Если n равно -1, то устанавливается наименьшее возможное для данного типа переменной значение; для строк типа STRING, PSTRING и CSTRING это значение равно коду ASCII 0.

Оператор **CLEAR** очищает переменную, указанную меткой. Если параметр опущен, то числовые переменные обнуляются, а строковые заполняются пробелами. Если метка является именем структуры GROUP, RECORD или QUEUE, то очищаются все переменные этой структуры. Если переменная имеет атрибут DIM, то очищается весь массив. Для очистки элемента массива нельзя использовать этот оператор.

**Пример:**

CLEAR(Count)

!Очистить переменную

CLEAR(Cus:Record)

!Очистить структуру

CLEAR(Amount,1)

!Установить максимально возможное значение

CLEAR(Amount,-1)

!Установить наименьшее возможное значение

**Правила преобразования типов данных**

В языке Clarion обеспечивается автоматическое преобразование типов данных. Однако некоторые присваивания могут давать неправильные значения. Непредсказуемый результат дает присваивание значения, выходящего за границы допустимого диапазона.

**Базовые типы**

Для облегчения автоматического преобразования типов данных Clarion использует четыре внутренних базовых типа, к которым автоматически преобразуются все элементы данных при выполнении над ними любых операций. Это типы: STRING, LONG, DECIMAL и REAL. Это стандартные типы данных Clarion

Базовый тип **STRING** используется в качестве промежуточного при всех строковых операциях. Во всех арифметических операциях используются базовые типы **LONG**, **DECIMAL** и **REAL**. Когда какой числовой тип используется определяется типом данных исходных операндов и типом выполняемой над ними операции.

“Нормальные” базовые типы для каждого типа данных:

Базовый тип **LONG**:

**BYTE**

**SHORT**

**USHORT**

**LONG**

**DATE**

**TIME**

Целочисленные константы

Базовый тип **DECIMAL**:

**ULONG**

**DECIMAL**

**PDECIMAL**

**STRING(@Nx.y)**

Десятичные константы

Базовый тип **REAL**:

**SREAL**

**REAL**

**BFLOAT4**

**BFLOAT8**

**STRING(@Ex.y)**

Константы в экспоненциальной записи

Нетипизированные (? and \*?) параметры

Базовый тип **STRING**:

**STRING**

**CSTRING**

**PSTRING**

Строковые константы

Типы данных **DATE** и **TIME** преобразуются в формат Стандартной даты Clarion и Стандартного времени Clarion имеют во всех операциях базовый тип **LONG**.

В большинстве случаев, использование внутренних базовых типов в Clarion прозрачно для программиста и не требует учета каких-либо особенностей при разработке прикладной программы. Однако, для экономических программ, содержащих числовые данные с дробной частью (денежные суммы, например), использование типов данных, для которых базовым

\* Операции с данными типа DECIMAL тесно связаны с обычной (десятичной) арифметикой.

Время двоично-десятичных вычислений зависит от данных, над которыми они производятся; затрачиваемое время пропорционально тому как долго такие вычисления производились бы вручную. Поэтому, чем длиннее числа над которыми производится операция, тем больше время выполнения. Однако, используются стандартные приемы ускорения вычислений (как например умножение на степень десяти переносом десятичной точки), делая быстрыми в реальных прикладных программах двоично-десятичные

вычисления с помощью библиотечных процедур.

Как двоично десятичные операции могут выполняться следующие операции:

Сложение (+), Вычитание (-), Умножение (\*) выполняются как двоично десятичные операции, когда ни один из операндов не имеет типа REAL. (оба типа LONG или DECIMAL) и один имеет базовый тип DECIMAL. Все цифры, возникающие справа от  $1 \times 10^{31}$ , исчезают (отбрасываются), и все слева от  $1 \times 10^{-30}$  округляется.

Деление (/) выполняется как двоично десятичные операция, когда ни один из операндов не имеет типа REAL. (оба типа LONG или DECIMAL). Все цифры, возникающие справа от  $1 \times 10^{31}$ , исчезают (отбрасываются), и все слева от  $1 \times 10^{-30}$  округляется.

Возведение в степень (^) выполняется как двоично десятичные операция, когда первый операнд имеет базовый тип DECIMAL или LONG, а второй операнд имеет базовый тип LONG. Все цифры, возникающие справа от  $1 \times 10^{31}$ , исчезают (отбрасываются), и все слева от  $1 \times 10^{-30}$  округляется.

ABS() отбрасывает знак значения переменной типа DECIMAL или промежуточного значения и возвращает значение типа DECIMAL.

INT() отбрасывает дробную часть промежуточного значения и возвращает значение типа DECIMAL.

ROUND() если базовый тип второго параметра LONG или DECIMAL, то округление выполняется как двоично-десятичная операция, в результате которой возвращается значение типа DECIMAL. Выполнение округления как двоично-десятичной операции очень эффективно и его следует использовать при сравнении переменных типа REAL с переменными типа DECIMAL по разрядности переменной типа DECIMAL.

## **Преобразование типов и промежуточные результаты**

---

Во внутреннем представлении двоично-десятичные промежуточные значения могут иметь точность до 31-й десятичной цифры с каждой стороны от десятичной точки, так что можно сложить два любых числа в формате DECIMAL, не теряя значащих цифр. Поэтому, перенос промежуточного двоично-десятичного значения в переменную какого-либо типа приводит к потере точности. Такое присвоение происходит по следующим правилам:

DECIMAL(x,y) = двоично-десятичное значение. Сначала двоично-десятичное значение округляется до y десятичных разрядов. Если результат превосходит x цифр, то старшие цифры отбрасываются (это похоже на “прохождение через 0” разрядов десятичного счетчика).



Целое = двоично-десятичное значение. Дробная часть начения отбрасывается. Целая часть преобразуется в двоичное представление без потери точности и может занимать по модулю до  $2^{32}$ .

String(@Nx.y) = двоично-десятичное значение. Двоично-десятичное значение округляется до у десятичных разрядов. Результат заполняет шаблонную строку. При возникновении переполнения результатом является неправильное представление в шаблоне (####).

Real = двоично-десятичное значение. Берутся 15 наиболее значащих цифр, а десятичная точка соответствующим образом передвигается.

Для тех операций и функций, которые не поддерживают тип DECIMAL, значения такого типа сначала преобразуются в REAL. В тех случаях, когда в двоично-десятичном значении было более 15 значащих цифр, происходит потеря точности.

**Замечание:** Нетипизированные параметры неявно имеют базовый тип REAL, следовательно, при передаче как нетипизированных параметров переменных с базовым типом DECIMAL передаются только 15 значащих цифр. Без потери точности переменные с базовым типом DECIMAL можно передавать как параметры \*DECIMAL.

При вычислении динамического выражения (или фильтра в структуре VIEW) используется базовый тип REAL.

## Преобразования типов данных при простых присваиваниях

---

Правила преобразования типов данных при простых присваиваниях:

BYTE = SHORT

BYTE = USHORT

BYTE = LONG

BYTE = ULONG

Знаковый бит исходной переменной игнорируется. В принимающую переменную переносятся младшие 8 бит источника.

BYTE = DECIMAL

BYTE = PDECIMAL

BYTE = REAL

BYTE = SREAL

BYTE = BFLOAT4

BYTE = BFLOAT8

Исходная переменная сначала преобразуется в LONG, при этом отбрасывается любая дробная часть. Затем в принимающую переменную переносятся младшие 8 бит промежуточного значения.

BYTE = STRING

BYTE = CSTRING

BYTE = PSTRING

Исходная строка должна содержать числовую величину без каких-либо символов форматирования. Исходная переменная сначала преобразуется в LONG, при этом отбрасывается любая дробная часть. Затем в принимающую переменную переносятся младшие 8 бит промежуточного значения.

SHORT = BYTE

Принимающая переменная становится равной источнику.

SHORT = USHORT

SHORT = LONG

SHORT = ULONG

В принимающую переменную переносятся младшие 16 бит числа источника.

SHORT = DECIMAL

SHORT = PDECIMAL

SHORT = REAL

SHORT = SREAL

SHORT = BFLOAT4

SHORT = BFLOAT8

Исходная переменная сначала преобразуется в LONG, при этом отбрасывается любая дробная часть. Затем в принимающую переменную переносятся младшие 16 бит промежуточного значения.

SHORT = STRING

SHORT = CSTRING

SHORT = PSTRING

Исходная строка должна содержать числовую величину без каких-либо символов форматирования. Исходной переменная сначала преобразуется в LONG, при этом отбрасывается любая дробная часть. Затем в принимающую переменную переносятся младшие 16 бит промежуточного значения.

USHORT = BYTE

В принимающую переменную переносится значение источника.

USHORT = SHORT

USHORT = LONG

USHORT = ULONG

В принимающую переменную переносятся младшие 16 бит источника.

USHORT = DECIMAL

USHORT = PDECIMAL

USHORT = REAL

USHORT = SREAL

USHORT = BFLOAT4

USHORT = BFLOAT8

Исходная переменная сначала преобразуется в LONG, при этом отбрасывается любая дробная часть. Затем в принимающую переменную переносятся младшие 16 бит промежуточного значения.

```
USHORT = STRING  
USHORT = CSTRING  
USHORT = PSTRING
```

Исходная строка должна содержать числовую величину без каких-либо символов форматирования. Исходная переменная сначала преобразуется в LONG, при этом отбрасывается любая дробная часть. Затем в принимающую переменную переносятся младшие 16 бит промежуточного значения.

```
LONG = BYTE  
LONG = SHORT  
LONG = USHORT  
LONG = ULONG
```

В принимающую переменную переносятся знак и значение источника.

```
LONG = DECIMAL  
LONG = PDECIMAL  
LONG = REAL  
LONG = SREAL  
LONG = BFLOAT4  
LONG = BFLOAT8
```

В принимающую переменную переносятся знак и значение источника, не превышающее 231. Если значение превышает 231, то переносится часть меньшая 231. Дробная часть отбрасывается.

```
LONG = STRING  
LONG = CSTRING  
LONG = PSTRING
```

Исходная строка должна содержать числовую величину без каких-либо символов форматирования. Исходная переменная сначала преобразуется в REAL, а затем в LONG,

```
DATE = BYTE  
DATE = SHORT  
DATE = USHORT  
DATE = ULONG
```

Принимающая переменная принимает значение в даты формате Btrieve. Подразумевается, что исходная переменная содержит дату в стандартном для Clarion формате.

```
DATE = DECIMAL  
DATE = PDECIMAL  
DATE = REAL  
DATE = SREAL  
DATE = BFLOAT4  
DATE = BFLOAT8
```

Исходная переменная сначала преобразуется в формат LONG как стандартная дата Clarion, при этом отбрасывается дробная часть, затем принимающая переменная принимает значение в формате даты Btrieve.

```
DATE = STRING
```

DATE = CSTRING

DATE = PSTRING

Исходная строка должна содержать числовую величину без каких-либо символов форматирования. Она сначала преобразуется в формат LONG как стандартная дата Clarion, при этом отбрасывается дробная часть, затем принимающая переменная принимает значение в формате даты Btrieve.

TIME = BYTE

TIME = SHORT

TIME = USHORT

TIME = ULONG

Принимающая переменная принимает значение времени в формате Btrieve. Подразумевается, что исходная переменная содержит время в стандартном для Clarion формате.

TIME = DECIMAL

TIME = PDECIMAL

TIME = REAL

TIME = SREAL

TIME = BFLOAT4

TIME = BFLOAT8

Исходная переменная сначала преобразуется в формат LONG как стандартное время Clarion, при этом отбрасывается дробная часть, затем принимающая переменная принимает значение в формате времени в Btrieve.

TIME = STRING

TIME = CSTRING

TIME = PSTRING

Исходная строка должна содержать числовую величину без каких-либо символов форматирования. Она сначала преобразуется в формат LONG как стандартное время Clarion, при этом отбрасывается дробная часть, затем принимающая переменная принимает значение в формате времени в Btrieve.

ULONG = BYTE

ULONG = SHORT

ULONG = USHORT

Исходная переменная сначала преобразуется в LONG, а затем все 32 бита переносятся в принимающую переменную.

ULONG = LONG

Все 32 бита исходной переменной переносятся в принимающую переменную.

ULONG = DECIMAL

ULONG = PDECIMAL

ULONG = REAL

ULONG = SREAL

ULONG = BFLOAT4

ULONG = BFLOAT8

Исходная переменная сначала преобразуется в LONG, при этом отбрасывается любая

дробная часть. Затем в принимающую переменную переносятся все 32 бит промежуточного значения.

```
ULONG = STRING
ULONG = CSTRING
ULONG = PSTRING
```

Исходная строка должна содержать числовую величину без каких-либо символов форматирования. Она сначала преобразуется в формат LONG, при этом отбрасывается дробная часть, затем в принимающую переменную переносятся все 32 бит промежуточного значения.

```
REAL = BYTE
REAL = SHORT
REAL = USHORT
REAL = LONG
REAL = ULONG
```

В принимающую переменную переносятся знак и значение источника.

```
REAL = DECIMAL
REAL = PDECIMAL
REAL = SREAL
REAL = BFLOAT4
REAL = BFLOAT8
```

В принимающую переменную переносятся знак, целая и дробная части значения источника.

```
REAL = STRING
REAL = CSTRING
REAL = PSTRING
```

Исходная строка должна содержать числовую величину без каких-либо символов форматирования. В принимающую переменную переносятся знак, целая и дробная части числа. Пробелы в конце строки игнорируются.

```
SREAL = BYTE
SREAL = SHORT
SREAL = USHORT
SREAL = LONG
SREAL = ULONG
```

В принимающую переменную переносятся знак и значение источника.

```
SREAL = DECIMAL
SREAL = PDECIMAL
SREAL = SREAL
```

В принимающую переменную переносятся знак целая и дробная части источника.

```
SREAL = STRING
SREAL = CSTRING
SREAL = PSTRING
```

Исходная строка должна содержать числовую величину без каких-либо символов форматирования. В принимающую переменную переносятся знак, целая и дробная части

числа. Пробелы в конце строки игнорируются.

BFLOAT8 = BYTE

BFLOAT8 = SHORT

BFLOAT8 = USHORT

BFLOAT8 = LONG

BFLOAT8 = ULONG

В принимающую переменную переносятся знак и значение источника.

BFLOAT8 = DECIMAL

BFLOAT8 = PDECIMAL

BFLOAT8 = REAL

В принимающую переменную переносятся знак, целая и дробная части источника.

BFLOAT8 = STRING

BFLOAT8 = CSTRING

BFLOAT8 = PSTRING

Исходная строка должна содержать числовую величину без каких-либо символов форматирования. В принимающую переменную переносятся знак, целая и дробная части числа. Пробелы в конце строки игнорируются.

BFLOAT4 = BYTE

BFLOAT4 = SHORT

BFLOAT4 = USHORT

BFLOAT4 = LONG

BFLOAT4 = ULONG

В принимающую переменную переносятся знак и значение источника.

BFLOAT4 = DECIMAL

BFLOAT4 = PDECIMAL

BFLOAT4 = REAL

В принимающую переменную переносятся знак, целая и дробная части источника.

BFLOAT4 = STRING

BFLOAT4 = CSTRING

BFLOAT4 = PSTRING

Исходная строка должна содержать числовую величину без каких-либо символов форматирования. В принимающую переменную переносятся знак, целая и дробная части числа. Пробелы в конце строки игнорируются.

DECIMAL = BYTE

DECIMAL = SHORT

DECIMAL = USHORT

DECIMAL = LONG

DECIMAL = ULONG

DECIMAL = PDECIMAL

В принимающую переменную переносятся знак и значение источника, соответственно обрезанное и округленное.

DECIMAL = REAL

DECIMAL = SREAL

В принимающую переменную переносятся знак, целая и старшие разряды дробной части значения источника. Значение округляется до последнего разряда, помещающегося в принимающей переменной.

```
DECIMAL = STRING
```

```
DECIMAL = CSTRING
```

```
DECIMAL = PSTRING
```

Исходная строка должна содержать числовую величину без каких-либо символов форматирования. В принимающую переменную переносятся знак, целая и дробная части числа. Пробелы в конце строки игнорируются.

```
PDECIMAL = BYTE
```

```
PDECIMAL = SHORT
```

```
PDECIMAL = USHORT
```

```
PDECIMAL = LONG
```

```
PDECIMAL = ULONG
```

```
PDECIMAL = DECIMAL
```

В принимающую переменную переносятся знак и значение источника, соответственно обрезанное и округленное.

```
PDECIMAL = REAL
```

```
PDECIMAL = SREAL
```

```
PDECIMAL = BFLOAT4
```

```
PDECIMAL = BFLOAT8
```

В принимающую переменную переносятся знак, целая и старшие разряды дробной части значения источника. Значение округляется до последнего разряда, помещающегося в принимающей переменной.

```
PDECIMAL = STRING
```

```
PDECIMAL = CSTRING
```

```
PDECIMAL = PSTRING
```

Исходная строка должна содержать числовую величину без каких-либо символов форматирования. В принимающую переменную переносятся знак, целая и дробная части числа. Пробелы в конце строки игнорируются.

```
STRING = BYTE
```

```
STRING = SHORT
```

```
STRING = USHORT
```

```
STRING = LONG
```

```
STRING = ULONG
```

В принимающую переменную переносятся знак и неформатированное число. Значение в строке выравнивается влево.

```
STRING = DECIMAL
```

```
STRING = PDECIMAL
```

```
STRING = REAL
```

```
STRING = SREAL
```

```
STRING = BFLOAT4
```

```
STRING = BFLOAT8
```

В принимающую переменную переносятся знак, целая и дробная части значения источника, округленное в соответствии с шаблоном строки. Значение в строке выравнивается влево.

CSTRING = BYTE

CSTRING = SHORT

CSTRING = USHORT

CSTRING = LONG

CSTRING = ULONG

В принимающую переменную переносятся знак и неформатированное число. Значение в строке выравнивается влево.

CSTRING = DECIMAL

CSTRING = PDECIMAL

CSTRING = REAL

CSTRING = SREAL

В принимающую переменную переносятся знак, целая и дробная части значения источника, округленное в соответствии с шаблоном строки. Значение в строке выравнивается влево.

PSTRING = BYTE

PSTRING = SHORT

PSTRING = USHORT

PSTRING = LONG

PSTRING = ULONG

В принимающую переменную переносятся знак и неформатированное число. Значение в строке выравнивается влево.

PSTRING = DECIMAL

PSTRING = PDECIMAL

PSTRING = REAL

PSTRING = SREAL

В принимающую переменную переносятся знак, целая и дробная части значения источника, округленное в соответствии с шаблоном строки. Значение в строке выравнивается влево.



## Глава 5 Управляющие структуры и операторы

### Управляющие структуры

#### CASE (структура условного выполнения)

```

CASE условие
OF выражение [ТО выражение ]
    операторы1
[OROF выражение] [ТО выражение]
    операторы2
[ELSE]
    операторы
    
```

<b>CASE</b>	Начинает структуру условного выполнения
условие	Числовая или строковая переменная или выражение.
<b>OF</b>	Когда выражение следующее за OF равно условию в операторе CASE, то выполняются идущие следом за OF операторы. В структуре CASE может быть несколько ветвей OF.
выражение	Числовая или строковая константа, переменная или выражение.
<b>ТО</b>	Позволяет указать в операторе OF или OROF диапазон значений. Операторы, следующие за OF (или OROF), выполняются, когда значение условия попадает в диапазон, указанный выражениями. Выражение, следующие за OF (или OROF), должно задавать нижнюю, а следующее за ТО, - верхнюю границу диапазона.
<b>OROF</b>	Операторы, следующие за OROF, выполняются, когда выражение, стоящее после него, равно условию в операторе CASE. С одним OF может быть связано несколько выражений OROF.
<b>ELSE</b>	Операторы, следующие за ELSE, выполняются, когда значение выражений во всех предыдущих ветвях OF или OROF не удовлетворяло условию в CASE. Ветвь ELSE необязательна, однако, если она присутствует, то должна быть последней ветвью в структуре.
операторы	Любые допустимые исполняемые операторы языка Clarion.

Структура **CASE** позволяет избирательно выполнять отдельные ветви, основываясь на равенстве условия и выражения (или попадании в диапазон). Эта структура может быть вложена в другие исполняемые структуры, а другие исполняемые структуры могут быть вложены в нее. Структура CASE должна оканчиваться оператором END или точкой.

В тех случаях, когда логика программы позволяет использовать или структуру CASE или составную структуру IF/ELSIF, для структуры CASE чаще всего генерируется более эффективный объектный код. А для тех случаев, когда выбор происходит на основе

целочисленного (от 1 до n) результата вычисления выражения , более эффективно использование структуры EXECUTE.

### Пример:

```

CASE FIELD()      !Функция, возвращающая номер обрабатываемого поля
OF ?Name          !Если это поле Name
  ERASE(?Address,?Zip)    ! очистить поля от Address до Zip
  GET(NameFile,NameKey)   ! прочитать запись
CASE Action       !Что за действие?
  OF 1             ! добавление, запись не существует
    IF NOT ERRORCODE()   ! должна быть ошибка, если запись уже есть
      ErrMsg = 'ALREADY ON FILE'  ! вывести сообщение
      DISPLAY(?Address,?Zip) ! вывести поля от Address до Zip
      SELECT(?Name)        ! повторно ввести имя
    .               ! конец IF
  OF 2 OROF 3      ! изменение или удаление - запись существует
    DISPLAY(?Address,?Zip) ! вывести поля от Address до Zip
  .               ! конец структуры CASE Action
CASE SUB(Name,1,1) !Взять первую букву имени
  OF 'A' TO 'M'     !Первая половина латинского алфавита
  OROF 'a' TO 'm'
  DO FirstHalf
  OF 'N' TO 'Z'     !Вторая половина латинского алфавита
  OROF 'n' TO 'z'
  DO SecondHalf
  .
  !Конец структуры CASE SUB(Name
OF ?Address        !Если это поле Address
  DO AddressVal    ! Выполнить п/п проверки значения поля
.

```

### EXECUTE (структура выбора оператора)

```

EXECUTE выражение
      оператор 1
      оператор 2
      [BEGIN
      операторы
      .]
      оператор n
.

```

EXECUTE                      Начинает структуру, избирательно выполняющую отдельный оператор.

выражение                    Целочисленное выражение или переменная, содержащая целое число.

оператор 1                    Единичный оператор, который выполняется только, когда выражение равно 1.

оператор 2	Единичный оператор, который выполняется только, когда выражение равно 2.
BEGIN	Оператор BEGIN отмечает начало операторной структуры, содержащей ряд строк исходного текста. В EXECUTE такая структура рассматривается как единичный оператор. Структура BEGIN заканчивается точкой или оператором END.
оператор n	Единичный оператор, который выполняется только, когда выражение равно n .

Основываясь на значении выражения, структура EXECUTE выбирает исполняемый оператор (или операторную структуру).

Если выражение равно 1, то выполняется первый оператор (оператор 1). Если выражение равно 2, то выполняется второй (оператор 2) и так далее. Если значение выражения равно 0 или больше числа операторов в структуре, то происходит переход на первый оператор, стоящий после структуры EXECUTE.

Эта структура может быть вложена в другие исполняемые структуры, а другие исполняемые структуры (IF, CASE, LOOP, EXECUTE и BEGIN) могут быть вложены в нее. В тех случаях, логика программы допускает использование и структуры EXECUTE и структур CASE и IF/ELSIF, для EXECUTE генерируется более эффективный объектный код, и предпочтительнее использовать ее.

### Пример:

```
EXECUTE Transact!Какую транзакцию выполнять
ADD(Customer)      ! Выполнить, если Transact = 1
PUT(Customer)      ! Выполнить, если Transact = 2
DELETE(Customer)   ! Выполнить, если Transact = 3
.                  !Конец структуры
EXECUTE CHOICE()   !В зависимости от значения функции CHOICE
OrderPart          !Выполнить если CHOICE() = 1
BEGIN              !Выполнить если CHOICE() = 2
    SavVendor" = Vendor
    UpdVendor
    IF Vendor <> SavVendor"
        Mem:Message = 'Имя продавца изменено'
.
CASE VendorType    !Выполнить если CHOICE() = 3
OF 1
    UpdPartNo1
OF 2
    UpdPartNo2
.
RETURN            !Выполнить если CHOICE() = 4
```

. !Конец структуры EXECUTE

Смотри также: оператор BEGIN

## IF (структура условного выполнения)

IF логическое выражение [THEN]  
 операторы  
 [ELSEIF логическое выражение] [THEN]  
 операторы  
 [ELSE]  
 операторы  
 .

IF	Начинает структуру условного выполнения операторов.
логическое выражение	Числовая или строковая переменная, выражение или функция. Дальнейшая последовательность выполнения определяется на основе результата вычисления этого логического выражения (истина или ложь). Нуль или пробельное значение рассматриваются как ложь, любое другое значение - как истинное.
THEN	Операторы, следующие за THEN выполняются, когда результатом логического выражения в IF является истина. Если операторы начинаются на следующей за выражением строке, то THEN должно быть опущено.
операторы	Исполняемый оператор или последовательность таких операторов.
ELSEIF	Логическое выражение, следующее за ELSEIF вычисляется, только когда условия в IF и во всех предшествующих ELSEIF оказались ложными.
ELSE	Операторы, следующие за ELSE, выполняются, если условия в IF и во всех предшествующих ELSEIF оказались ложными. Ветвь ELSE не является обязательной, но когда присутствует, она должна быть последней ветвью в структуре IF.

Структура IF управляет выполнением программы, основываясь на результате одного или более логических выражений. Она может содержать любое количество групп операторов возглавляемых конструкцией ELSEIF THEN. Эта структура может быть вложена в другие исполняемые структуры, а другие исполняемые структуры могут быть вложены в структуру IF.

### Пример:

```
IF Cus:TransCount = 1                !Если новый покупатель
  AcctSetup                          ! вызвать процедуру инициализации счета
ELSEIF Cus:TransCount > 10 AND Cus:TransCount < 100    !Постоянный покупатель
  DO RegularAcct                    ! обработать счет
  ELSEIF Cus:TransCount > 100        !Если особый покупатель
  DO SpecialAcct                    ! обработать счет
```

```

ELSE                !Иначе
DO NewAcct          ! обработать счет
IF Cus:Credit THEN CheckCredit ELSE CLEAR(Cus:CreditStat) . !проверить состояние кредита
.                  !Конец структуры IF
IF ERRORCODE() THEN ErrHandler(Cus:AcctNumber,Trn:InvoiceNbr). !Обработка ошибок

```

## LOOP (структура повторения)

```

                число TIMES
метка LOOP [i = начало TO предел [BY шаг]
                UNTIL логическое выражение
                WHILE логическое выражение
                операторы
END
UNTIL логическое выражение
WHILE логическое выражение

```

**LOOP** Начиная структуру циклического выполнения операторов.

число Числовая константа, переменная или выражение, которая определяет число циклов выполнения операторов в структуре.

**TIMES** Конструкция TIMES указывает, что операторы выполняются заданное число раз.

i Метка переменной (счетчика цикла), которая автоматически увеличивается (или уменьшается - прим. перев) при каждом повторении цикла.

= Присвоение нового значения счетчику цикла (i) для каждой итерации.

начало Числовая константа, переменная или выражение, которая задает начальное значение, присваиваемое переменной цикла при первом выполнении тела цикла.

**TO** Синтаксическая связка для параметра предел.

**BY** Синтаксическая связка для параметра шаг.

предел Когда переменная i становится больше (или при отрицательном шаге меньше - прим. перев) чем предел, выполнение структуры LOOP прекращается.

шаг Числовая константа, переменная или выражение. Шаг определяет величину приращения счетчика при каждом выполнении цикла (может быть отрицательным - прим. перев). По умолчанию шаг равен 1.

**UNTIL** Помещенный внутрь структуры LOOP оператор UNTIL означает, что перед каждым выполнением цикла вычисляется логическое выражение. Если же структура LOOP им заканчивается, то логическое выражение вычисляется после каждой итерации. Если его значение есть условие "истина", то выполнение структуры LOOP прекращается.

**WHILE** Помещенный внутрь структуры LOOP оператор WHILE означает,

что перед каждым выполнением цикла вычисляется логическое выражение. Если же структура LOOP им заканчивается, то логическое выражение вычисляется после каждой итерации. Если его значение есть условие “ложь”, то выполнение структуры LOOP прекращается.

логическое выражение Числовая или строковая переменная, выражение или функция. Оно имеет своим значением условие. На его основе (истина или ложь) определяется дальнейшая последовательность выполнения программы. Нуль или пробельное значение рассматриваются как ложь, любое другое значение - как истинное.

Структура LOOP предназначена для циклического повторения находящихся в ней операторов. Условия выполнения цикла всегда проверяются вначале, перед выполнением цикла. Эта структура может быть вложена в другие исполняемые структуры, а другие исполняемые структуры могут быть вложены в структуру LOOP. Структура LOOP должна завершаться оператором END, точкой или операторами UNTIL или WHILE.

Структура LOOP без условий в начале или в конце повторяется непрерывно, до обнаружения оператора BREAK или RETURN. Оператор BREAK прерывает выполнение цикла, и управление передается на оператор, следующий за структурой LOOP. Оператор CYCLE прерывает текущую итерацию, а управление передается на начало цикла, не выполняя операторы, стоящие в теле цикла после CYCLE.

### Пример:

```

LOOP           !Безусловный цикл
Char = GetChar()! получить символ
IF Char <> CarrReturn ! если это не “возврат каретки”
  Field = CLIP(Field) & Char  ! “прицепить” символ
ELSE
  BREAK           ! выйти из цикла
..               !Конец IF и цикла
IF ERRORCODE() !При ошибке
  LOOP 3 TIMES   ! Повторить 3 раза
  BEEP          ! дать звуковой сигнал
..             !Конец IF и цикла
LOOP I# = 1 TO 365 BY 7!Повторять, увеличивая I# каждый раз на 7
  GET(DailyTotal,I#)      ! читать каждую 7-ю запись
  DO WeeklyJob           ! выполнить подпрограмму
.                         !Конец цикла
SET(MasterFile) !Встать на первую запись
LOOP UNTIL EOF(MasterFile) !Обработать все записи
  NEXT(MasterFile)      ! прочитать запись
  ProcMaster           ! выполнить процедуру
.                       !Конец цикла
LOOP WHILE KEYBOARD()   !Очистить клавиатурный буфер

```

ASK  
 UNTIL KEYCODE() = EscKey ! но прекратить цикл по нахлждениии Escape

Смотри также: BREAK, CYCLE

## Управляющие операторы

### BREAK (прервать выполнение цикла)

**BREAK**[ метка ]

**BREAK** Передать управление на первый оператор, следующий за прерываемой структурой LOOP или АССЕПТ.  
 метка Метка структуры LOOP или АССЕПТ, выполнение которой должно прерваться. Это должна быть метка вложенной структуры, содержаще оператор BREAK.

Оператор BREAK прерывает работу в цикле the LOOP или АССЕПТ и передает управление первому оператору, следующему за заканчивающим структуру оператором END, WHILE, или UNTIL структуры LOOP, или оператором END структуры АССЕПТ.

Оператор BREAK можно использовать только в структурах LOOP и АССЕПТ. Использование необязательного аргумента метка, позволяет целенаправленно прервать несколько вложенных циклов, исключая тем самым наиболее частый случай применения оператора GOTO.

#### Пример:

```

LOOP                !Цикл
ASK                ! ждать ввода с клавиатуры
IF KEYCODE() = EscKey    ! если нажата клавиша Esc
    BREAK          ! выйти из цикла
ELSE                ! иначе
    BEEP           ! дать звуковой сигнал
END
END
```

```

Loop1  LOOP          !Loop1 это метка
        DO ParentProcess
Loop2  LOOP          !Loop2 это тоже метка
        DO ChildProcess
        IF SomeCondition
            BREAK Loop1    !Прервать оба вложенных цикла
        END
```

```

END
END

ACCEPT          !Цикл ACCEPT
CASE ACCEPTED()
  OF ?Ok
    CallSomeProc
  OF ?Cancel
    BREAK      ! выйти из цикла
END
END

```

Смотри также: LOOP, CYCLE, ACCEPT

## CHAIN (выполнить другую программу)

### CHAIN(программа)

**CHAIN**           Заканчивает выполнение текущей программы и выполняет другую программу.

программа       Строковая константа или переменная, содержащая имя программы, которая должна быть выполнена. Это может быть любая программа (.EXE или .COM)

Оператор **CHAIN** прерывает выполнение текущей программы, закрывает все файлы, возвращает занимаемую память операционной системе и сообщает ей о необходимости выполнить другую программу.

### Пример:

```

PROGRAM          !Программа главного меню
CODE
EXECUTE CHOICE()
  CHAIN('Ledger')    !Выполнить LEDGER.EXE
  CHAIN('Payroll')   !Выполнить PAYROLL.EXE
  RETURN             !Вернуться в DOS
.
PROGRAM          !Фрагмент программы Ledger
CODE
EXECUTE CHOICE()
  CHAIN('MainMenu')  !Вернуться в главное меню
  RETURN             !Возврат в DOS
.

```



```

PROGRAM          !Фрагмент программы Payroll
CODE
EXECUTE CHOICE()
  CHAIN('MainMenu') !Вернуться в главное меню
  RETURN          !Возврат в DOS

```

Смотри также: RUN

## CYCLE (переход в начало цикла)

**CYCLE**[ метка ]

**CYCLE** Передать управление на начало цикла или цикла ACCEPT.  
 метка Метка оператора LOOP или ACCEPT на который передать управление. Это должна быть метка вложенной структуры LOOP или ACCEPT, содержащей оператор LOOP.

Оператор **CYCLE** передает управление в начало цикла. Оператор CYCLE может использоваться только в структурах LOOP или ACCEPT. Использование необязательной метки позволяет целенаправленно передать управление на один из вложенных циклов, исключая тем самым один наиболее частых случаев применения оператора GOTO

В цикле ACCEPT для некоторых событий (таких как EVENT:Move) оператор CYCLE прерывает автоматически выполняемые действия до начала их выполнения. Такое его действие описано для каждого события, к которому это относится.

### Пример:

```

SET(MASTER_FILE)          !Встать на первую запись
LOOP UNTIL EOF(MASTER_FILE) !Цикл по всем записям
  NEXT(MASTER_FILE)        !Читать следующую запись
  DO MatchMaster           ! проверить соответствие
  IF NoMatch               ! если не соответствует
    CYCLE                 ! на начало цикла
  .                         ! конец IF
  DO TransVal              ! проверить транзакцию
  PUT(MasterFile)          ! занести запись
  .                         !Конец цикла
Loop1 LOOP                 !Loop1 - это метка
  DO ParentProcess
Loop2 LOOP                 !Loop2 это тоже метка
  DO ChildProcess
  IF SomeCondition
    CYCLE Loop1 !Перейти на начало внешнего цикла
  END
END
END
END

```

Смотри также: LOOP, BREAK, ACCEPT

## DO (выполнить локальную подпрограмму)

**DO** метка

**DO** Выполняет локальную подпрограмму.

метка Метка оператора ROUTINE

Оператор **DO** используется для выполнения подпрограммы локальной по отношению к программе, процедуре или функции. Когда выполнение локальной подпрограммы завершено, управление передается оператору, непосредственно следующему за оператором DO, вызвавшим подпрограмму. Локальная подпрограмма может вызываться только внутри содержащей ее программной секции.

**Пример:**

DO NextRecord !Вызвать подпрограмму чтения следующей записи

DO CalcNetPay !Вызвать подпрограмму CalcNetPay

**Смотри также:** EXIT, ROUTINE

## EXIT (прекратить выполнение локальной подпрограммы)

**EXIT**

Оператор **EXIT** прекращает выполнение локальной подпрограммы и возвращает управление оператору, следующему за оператором DO, который вызвал данную подпрограмму. Это необязательный оператор в локальной подпрограмме. Подпрограмма без оператора EXIT автоматически заканчивается после выполнения последнего исполняемого оператора в ней.

**Пример:**

CalcNetPay ROUTINE

IF GrossPay = 0 !Если оплаты нет

EXIT ! выйти из подпрограммы

.

NetPay = GrossPay - FedTax - Fica

QtdNetPay += NetPay

YtdNetPay += NetPay

## GOTO (безусловный переход)

### GOTO метка

**GOTO** Безусловная передача управления на другой оператор программы.  
 метка Метка другого исполняемого оператора в программе, процедуре, функции или локальной подпрограмме.

Оператор **GOTO** передает управление от одного оператора к другому. Метка перехода в этом операторе не должна быть меткой оператора ROUTINE, PROCEDURE или FUNCTION.

Область действия оператора GOTO ограничивается данной локальной подпрограммой, процедурой или функцией; он не может передать управление за пределы локальной подпрограммы, процедуры или функции, в которой он используется.

#### Пример:

```
ComputeIt FUNCTION(Level)
    CODE
    IF Level = 0 THEN GOTO PassCompute.    !Пропустить вычисления
    Rate = Level * Markup    !Вычислить Rate
    RETURN(Rate)    ! и вернуть его
PassCompute    RETURN(999999)
```

## HALT (вернуться в DOS)

### HALT([код завершения],[сообщение])

**HALT** Немедленное завершение программы.  
 код завершения Положительная целочисленная константа или переменная в диапазоне от 0 до 250, которая представляет собой код возврата, передаваемый в DOS. Это значение принимает переменная среды ERRORLEVEL. Этот параметр не является обязательным. Если он опущен, а второй параметр присутствует, то требуется запятая, обозначающая пропуск первого параметра.  
 сообщение Строковая константа или переменная, которая выводится после завершения программы на экран.

Оператор **HALT** немедленно возвращает управление в DOS, устанавливая значение переменной ERRORLEVEL, с возможным выводом на экран сообщения после завершения программы. Если имеется процедура SHUTDOWN, то перед завершением программы она выполняется.

Если завершаемая оператором HALT программа запущена из другой программы на языке Clarion оператором RUN или RUNSMALL, то установленный оператором HALT код завершения можно определить, используя функцию RUNCODE.

### Пример:

```

Password  STRING(10)
          CODE
          BLANK
          SHOW(1,1,'Введите, пожалуйста, пароль')
          Password = ENTRY(2,3,@S10)
          IF Password <> 'Pay$MeMore'
            HALT(0,'Введен неправильный пароль')
```

Смотри также: RUN, RUNCODE, STOP

## IDLE (включить периодически исполняемую процедуру)

**IDLE**([процедура] [,интервал])

<b>IDLE</b>	Включает периодически исполняемую процедуру.
процедура	Метка оператора PROCEDURE. Эта процедура не может принимать никаких параметров.
интервал	Целое число, задающее минимальное время ожидания (в секундах) между обращениями к процедуре. Интервал равный 0 устанавливает, что обращение к процедуре происходит постоянно. Если параметр интервал опущен, то по умолчанию устанавливается 1 секунда.

Процедура **IDLE** активна, когда операторы ASK или ACCEPT ожидают реакции пользователя. В любой момент времени может быть активна только одна фоновая процедура, и она выполняется в нулевом исполняемом процессе. Указание новой фоновой процедуры автоматически приводит к деактивации старой. Оператор IDLE без параметров выключает периодическое обращение к фоновой процедуре.

Процедура типа IDLE обычно описывается прототипом в структуре MAP в программном модуле (не в member-модуле). Если же ее прототип содержится в member-модуле, то операторы IDLE, включающие и выключающие обращение к этой процедуре должны содержаться внутри процедур и функций этого же модуля.

### Пример:

IDLE(ShoTime,10)	!Вызывать shotime каждые 10 секунд
IDLE(CheckNet)	!Проверять работу сети каждую секунду
IDLE	!Выключить обращение к фоновой процедуре

**Смотри также:** ASK, ACCEPT, PROCEDURE, MAP

## **RETURN (возврат в вызвавшую процедуру или функцию)**

**RETURN**([выражение])

**RETURN** Заканчивает программу, процедуру или функцию.  
 выражение Выражение передает возвращаемое функцией значение в то выражение, где функция используется. Это выражение обязательно для оператора **RETURN** в функции, но не может использоваться в процедуре или программе.

Оператор **RETURN** завершает выполнение программы, процедуры или функции и передает управление в вызвавшую программу, процедуру или функцию. Когда этот оператор выполняется в секции исполняемых операторов программного модуля, то все файлы закрываются, программа завершается, а управление передается операционной системе.

Оператор **RETURN** требуется в функции и необязателен в процедуре или программе. Если в процедуре или программе не используется оператор **RETURN**, то его неявное выполнение происходит вслед за последним исполняемым оператором. Конец секции исполняемых операторов определяется по концу исходного файла или началу другой процедуры, функции или локальной подпрограммы.

При выходе из процедуры или функции (явном или неявном) автоматически закрывает любые локальные структуры **APPLICATION**, **WINDOW**, **REPORT** или **VIEW**, открытые в данной процедуре или функции. Но при этом не закрываются глобальные или структуры **APPLICATION**, **WINDOW**, **REPORT** или **VIEW**, описанные в секции данных модуля без атрибута **STATIC**. Кроме того, при этом освобождаются любые локальные структуры **QUEUE**, объявленные без атрибута **STATIC**.

### **Пример:**

```

      IF Done# THEN RETURN      !Закончить, когда все сделано
DayOfWeek FUNCTION(Date) !Функция, возвращающая день недели
CODE
EXECUTE (Date % 7) + 1 !Определить номер дня недели
RETURN('Sunday')      ! и вернуть его строкой
RETURN('Monday')
RETURN('Tuesday')
RETURN('Wednesday')
RETURN('Thursday')
```



Смотри также: RUNCODE, HALT

## SHUTDOWN (включить процедуру завершения)

**SHUTDOWN**([процедура])

**SHUTDOWN** Установить процедуру, к которой происходит обращение при завершении программы  
процедура Метка оператора PROCEDURE. Если этот параметр опущен, то процесс завершения программы пользовательской процедурой не инициируется.

Оператор **SHUTDOWN** устанавливает процедуру, к которой происходит обращение при завершении программы. Процедура, установленная оператором **SHUTDOWN**, вызывается при нормальном завершении или при аварийном останове программы. Иногда при аварийном останове невозможно выполнить такую процедуру в зависимости от состояния системных ресурсов в момент аварии. Не вызывается эта процедура и при перезагрузке компьютера или сбое электропитания. Не рекомендуется внутри процедуры, указанной в **SHUTDOWN**, использовать оператор **RESTART**.

Точно такого же эффекта как указанием процедуры в операторе **SHUTDOWN**, можно более безболезненно добиться, просто вызывая эту процедуру по событию **EVENT:CloseDown** в приложении.

### Пример:

**SHUTDOWN**(CloseSys) !Установить CloseSys в качестве завершающей процедуры

## STOP (приостановить выполнение программы)

**STOP**([сообщение])

**STOP** Приостановить выполнение программы и раскрыть окно с сообщением.  
сообщение Необязательное строковое выражение (до 64 Кбайт), которое выводится в окне сообщения.

Оператор **STOP** приостанавливает программу и выводит на экран окно сообщения. Он предлагает пользователю возможность продолжить программу. В случае выхода закрываются все файлы и освобождается выделенная программе память.

### Пример:

```
PswdScreen      WINDOW(3,30)
                STRING('Введите, пожалуйста, пароль ')
                ENTRY(@10),USE>Password,HIDE      !Поле для ввода пароля
.
CODE
OPEN(PswdScreen)      !Вывести запрос на ввод пароля
ACCEPT      ! и ввести его с клавиатуры
CASE ACCEPTED
OF ?PASSWORD
    IF Password <> 'PayMe$moRe' !Правильный пароль ?
        STOP('Введен неправильный пароль — доступ запрещен')
        HALT(0,'Неправильный пароль')      !Если нет, отвергнуть пользователя
    END
END
END
END
```







## **Глава 6. Окна и меню**

### **Окна в Clarion**

#### **Обзор окон**

---

В большинстве программ в среде Windows используется три типа окон: окно прикладной программы, документальное окно и диалоговое окно. Окно прикладной программы - это первое раскрытое в программе окно, и обычно, оно содержит главное меню как средство обращения к остальным частям программы. Все остальные окна в программе - документальные и диалоговые окна.

Наряду с экранными окнами этих трех типов в программах под Windows используется две системы правил построения пользовательского интерфейса: интерфейс одного документа (Single Document Interface - SDI) и интерфейс нескольких документов (Multiple Document Interface - MDI).

Программы с интерфейсом одного документа обычно имеют только линейную логику, которая позволяет пользователю в данный момент времени выполнять только одну цепочку действий (один процесс). Такая программа не порождает отдельных процессов, между которыми пользователь может переключать свое внимание. Точно такой тип логики используется в большинстве программ в среде DOS. Программа с SDI не должна содержать структуру APPLICATION в качестве структуры, описывающей окно прикладной программы. Чтобы определить окно прикладной программы с MDI используется языковая структура WINDOW, без атрибута MDI, а последующие документальные и диалоговые окна раскрываются поверх окна прикладной программы.

Программы с MDI позволяют пользователю выбрать несколько выполняющихся процессов и в любое время переключаться между ними. Это очень распространенный интерфейс пользователя программ в среде Windows. Он используется прикладными программами как средство организации и группировки окон, представляющих различные исполняемые процессы, для запустившего их пользователя:

В языке Clarion окно прикладной программы с MDI определяется с помощью структуры APPLICATION. Окно прикладной программы с MDI работает как порождающее окно для всех порождаемых MDI-окон (документальных и диалоговых окон), в пределах которого все они размещаются и автоматически перемещаются при перемещении окна прикладной программы. Кроме того порожденные окна могут быть скрыты совсем при минимизации порождающего окна. В Clarion-программе для среды Windows в любой момент времени может быть открыта только одна структура APPLICATION.

Документальные и диалоговые окна очень похожи в том смысле, что оба определяются посредством структуры WINDOW языка Clarion. Они различаются в том контексте, в котором они обычно используются, и соглашениями, касающимися появления их на экране и атрибутов. Часто эти отличия неразличимы и не имеют значения. Общим термином и для документального окна, и для диалогового окна служит термин “окно”, который и будет использоваться повсюду в этой книге.

Обычно в документальных окнах отображаются данные. По соглашениям эти окна являются перемещаемыми и изменяемыми в размерах. Обычно у них есть заголовок, системное меню, и кнопка Maximize. Например, в оболочке Windows документальным окном является окно программной группы “Main”, которое появляется при двойном щелчке мышью на пиктограмме “Main” находящейся в рамках окна диспетчера программ (Program Manager).

В диалоговых окнах от пользователя обычно запрашиваются данные, или пользователь оповещается о некой ситуации, обычно перед тем как выполнить некоторые действия, запрошенные пользователем. Эти окна могут быть (а могут и не быть) перемещаемыми и таким образом могут иметь (или не иметь) системное меню и заголовок. По правилам они не являются изменяемыми в размерах, хотя могут иметь кнопку Maximize, с помощью которой окно может принимать два альтернативных размера. Диалоговое окно может быть модальным относительно системы (пользователь должен обязательно каким-то образом прореагировать, прежде чем что-либо делать далее в среде Windows) или модальным относительно прикладной программы (пользователь должен обязательно каким-то образом прореагировать, прежде чем что-либо делать далее в прикладной программе) либо немодальным. К примеру в среде Clarion окно, которое раскрывается при выборе в меню File пункта Open представляет собой диалоговое окно, имя файла, который надлежит открыть.

## **Управляющие поля и “фокус ввода”**

---

Объекты, объявленные в структуре APPLICATION или WINDOW, представляют собой “управляющие поля” (controls). Это стандартный в Windows термин, использующийся для ссылки на любой экранный объект: командные кнопки, поля для ввода текста, кнопки радио, окна списка и т.д. . В большинстве программ для DOS термин “поле” обычно используется для указания на те же объекты. В данном руководстве термины “управляющее поле” и “поле” являются взаимозаменяемыми.

Управляющие поля появляются только в структурах MENUBAR, TOOLBAR и WINDOW. Только когда в данное поле переключен “фокус ввода” становится возможным для пользователя выбирать поле и/или редактировать содержащиеся в нем данные. Переключение фокуса происходит, когда пользователь использует клавишу TAB, манипулятор мышь или комбинацию “быстрых клавиш” для выделения управляющего поля.

Окно также приобретает фокус, когда оно является верхним окном в активном в данный момент исполняемом процессе. Поскольку Clarion для Windows допускает создание программ с несколькими исполняющимися процессами вопрос о том, на какое окно переключен фокус имеет важное значение. Активной является только тот процесс, которому принадлежит самое верхнее окно. Редактировать данные в управляющем поле окна пользователь может только если на это поле переключен фокус.

## **Метки соответствия полей**

---

В структурах Windows каждому управляющему полю, имеющему USE-переменную, компилятором присваивается номер. По умолчанию эти номера полей начинаются с единицы (1) и присваиваются управляющим полям в порядке их следования в тексте структуры WINDOW. Посредством второго параметра атрибута USE присваиваемый номер может быть переустановлен. Порядок следования полей в тексте структуры определяет “естественный” порядок выбора полей для структуры ACCEPT (который может изменяться с помощью оператора SELECT). Порядок следования в тексте структуры не зависит от расположения поля на экране. Поэтому нет необходимости в каком-либо соответствии положения управляющего поля на экране и назначаемым компилятором номером поля.

Существует ряд операторов, в которых эти номера полей используются в качестве параметров. Было бы очень утомительным жестко кодировать номера полей при использовании их в таких операторах. Поэтому Clarion имеет механизм разрешения этой проблемы - метки соответствия полей.

Метки соответствия полей всегда начинаются со знака вопроса (?), за которым следует имя USE-переменной управляющего поля. Знак вопроса в начале имени обозначает для компилятора метку соответствия поля. Метки соответствия очень похожи на обычные директивы компилятора EQUATE. Во время компиляции компилятор подставляет вместо метки соответствия номер поля. Метки соответствия делают ненужным знать впоследствии номера полей.

Метки соответствия для USE-переменных, которые являются элементами массива всегда начинаются со знака вопроса, за которым следует имя USE-переменной, в конце которого знак подчеркивания и номер элемента массива. Например, метка соответствия для USE(ArrayField[1]) будет ?ArrayField\_1. Многомерные массивы рассматриваются подобным же образом (?ArrayField\_1\_1, ?ArrayField\_1\_2, ...).

Метки соответствия для USE-переменных, которые являются элементами составной структуры данных всегда начинаются с вопросительного знака, за которым идет имя USE-переменной с двоеточиями (:) вместо точек (.). Например, метка соответствия для USE(Phones.Rec.Name) будет ?Phones:Rec:Name.

Два и более управляющих поля, имеющих одну и ту же USE-переменную, в одной структуре WINDOW или APPLICATION порождают одинаковые метки соответствия. Поэтому, когда компилятор встречает такую ситуацию все метки соответствия для этой USE-переменной отменяются. Это делает невозможным сослаться на любое из этих полей в исполняемых операторах, не вызывая неоднозначности относительно того, к которому полю в действительности намеревались обратиться. Кроме того, можно осознанно создавать такую ситуацию для того, чтобы выводить содержимое одной переменной в нескольких управляющих полях, используя различное графическое представление. Для некоторых полей в качестве USE-переменной может использоваться только метка соответствия поля (уникальная метка, начинающаяся со знака вопроса). Это обеспечивает способ обращения к этим полям в исполняемых операторах.

В структурах APPLICATION каждому пункту меню на линейке меню и каждому управляющему полю на панели инструментов компилятором присваивается номер. По умолчанию эта нумерация начинается с минус единицы (-1) и уменьшается на 1 для каждого управляющего поля в порядке следования этих управляющих полей в тексте структуры APPLICATION.

## **Структуры, описывающие окно**

### **APPLICATION (объявить MDI окно)**

```
метка APPLICATION(“заголовок”)[,AT()][,CENTER][,SYSTEM][,MAX][,ICON()]
[STATUS()][,HLP()][,CURSOR()][,TIMER()][,ALRT()][,ICONIZE]
[ MAXIMIZE ][, MASK ][, FONT()][, MSG()]
[ , HSCROLL ][, DOUBLE ]
VSCROLL NOFRAME
HVSCROLL RESIZE
[ MENUBAR
    объявления нескольких меню и/или пунктов
END ]
[ TOOLBAR
    объявления нескольких управляющих полей
END ]
END
```

#### **APPLICATION**

Объявляет окно программы с интерфейсом MDI

метка

Допустимая в Clarion метка. В операторе APPLICATION метка обязательна.

заголовок

Задаёт текст заголовка основного окна прикладной программы.

AT

Задаёт начальный размер и расположение основного окна

	прикладной программы.
<b>CENTER</b>	Указывает, что по умолчанию первоначально окно центрируется на экране. Этот атрибут действует, только если по крайней мере один параметр атрибута AT опущен.
<b>SYSTEM MAX</b>	Задаёт наличие системного меню. Задаёт наличие кнопок приведения к максимальному и минимальному размерам и кнопки восстановления “естественного” размера окна.
<b>ICON</b>	Задаёт наличие кнопок приведения к максимальному, минимальному и естественному размерам, и указывает имя файла или стандартного идентификатора пиктограммы, высвечиваемой при минимизации размера окна.
<b>STATUS</b>	Задаёт наличие линейки состояния внизу окна прикладной программы.
<b>HLP</b>	Задаёт “идентификатор подсказки” связанной с окном прикладной программы и обеспечивает идентификатор по умолчанию для любого порожденного окна.
<b>CURSOR</b>	Задаёт форму курсора мыши, которую он принимает в пределах окна прикладной программы. Если этот атрибут опущен, то это курсор используемый по умолчанию в среде Windows.
<b>TIMER ALRT</b>	Задаёт генерирование периодических временных событий. Указывает “горячую” комбинацию клавиш для всего окна прикладной программы.
<b>ICONIZE</b>	Указывает, что при открытии окно прикладной программы минимизируется до пиктограммы.
<b>MAXIMIZE</b>	Указывает, что при открытии окно прикладной программы принимает максимальный размер.
<b>MASK</b>	Задаёт режим редактирования по шаблону вводимых данных для всех вводных управляющих полей на панели инструментов.
<b>FONT</b>	Указывает используемый по умолчанию шрифт для всех управляющих полей на инструментальной панели.
<b>MSG</b>	Задаёт строковую константу, содержащую текст, выводимый по умолчанию в линейке состояния для всех управляющих полей структуры APPLICATION.
<b>HSCROLL</b>	Указывает, что к рамке окна прикладной программы автоматически добавляется линейка горизонтального скроллинга, если какая-либо часть порожденного окна по горизонтали выходит за границы видимой области.
<b>VSCROLL</b>	Указывает, что к рамке окна прикладной программы автоматически добавляется линейка вертикального скроллинга, если какая-либо часть порожденного окна по вертикали выходит за границы видимой области.
<b>HVSCROLL</b>	Указывает, что к рамке окна прикладной программы автоматически добавляется и линейка горизонтального и линейка вертикального скроллинга, если какая-либо часть порожденного окна выходит за границы

видимой области.

- DOUBLE**                   Задаёт вокруг окна рамку двойной ширины. Окно такого типа не может изменяться в размерах.
- NOFRAME**               Задаёт окно без рамки. Окно такого типа не может изменяться в размерах.
- RESIZE**                 Задаёт толстую рамку вокруг окна, которая позволяет изменять размеры окна.
- MENUBAR**              Определяет структуру меню (необязательную). Меню заданное в структуре APPLICATION является “глобальным меню”.
- TOOLBAR**              Определяет структуру панели инструментов (необязательной). Панель инструментов, заданная в структуре APPLICATION является “глобальной”.

Структура APPLICATION объявляет обрамляющее окно с интерфейсом нескольких документов (MDI). Интерфейс MDI представляет собой часть стандартного интерфейса системы Windows и используется прикладными программами для того, чтобы представлять в разных окнах вывод на монитор разных процессов. Структура APPLICATION представляет собой способ организации и группировки MDI-окон. Обрамляющее MDI-окно (описываемое структурой APPLICATION) функционирует как “порождающее” окно по отношению к другим “порожденным” MDI-окнам (описываемым структурами WINDOW с атрибутом MDI). Эти порожденные MDI-окна располагаются в границах обрамляющего окна и автоматически перемещаются при его перемещении и могут быть вообще скрыты при его минимизации до пиктограммы.

В любой момент времени в Clarion-программе может быть открыто только одно порождающее MDI-окно и оно должно быть открыто до того как будет открыто какое-либо порождаемое MDI-окно. Однако не-MDI-окна могут открываться и перед открытием структуры APPLICATION. Структура APPLICATION, описывающая обычное порождающее окно должно иметь атрибуты ICON, MAX, STATUS, RESIZE b SYSTEM. В этом случае создается окно изменяемых размеров с рамкой, кнопками минимизации и максимизации, линейкой состояния и системным меню. Кроме того, в структуре APPLICATION должна быть структура MENUBAR, содержащая элементы глобального меню, и может быть структура TOOLBAR, описывающая мгновенный доступ к элементам глобального меню. Таким образом создается стандартный для Windows интуитивно понятный интерфейс.

Окно APPLICATION не может содержать управляющих полей кроме как внутри структур MENUBAR и TOOLBAR, и не может использоваться для вывода данных. Для этого требуются документальные и диалоговые окна (описываемые с помощью структур WINDOW).

Если сначала открывается окно APPLICATION, то оно остается скрытым до выполнения первого оператора DISPLAY или цикла ACCEPT. Это позволяет выполнить перед выводом



окна некоторые изменения в его внешнем виде. Например, посредством установки параметров во время выполнения можно настроить заголовок или размеры окна.

Генерируемые события:

EVENT:PreAlertKey	Пользователь нажал указанную атрибутом ALRT горячую клавишу.
EVENT:AlertKey	Пользователь нажал указанную атрибутом ALRT горячую клавишу.
EVENT:CloseWindow	Окно закрывается.
EVENT:CloseDown	Приложение завершается.
EVENT:OpenWindow	Окно раскрывается.
EVENT:LoseFocus	Фокус переключается от данного окна на другой процесс.
EVENT:GainFocus	Данное окно получает фокус от другого процесса.
EVENT:Suspend	Фокус еще остается на данном окне, но управление передается другому процессу для обработки таймерных событий.
EVENT:Resume	Фокус остается на данном окне, и управление снова получает управление.
EVENT:Timer	Переключен атрибут TIMER (произошло таймерное событие).
EVENT:Move	Пользователь перемещает окно. Оператор CYCLE прерывает перемещение.
EVENT:Moved	Пользователь переместил окно.
EVENT:Size	Пользователь изменяет размеры окна. Оператор CYCLE прерывает изменение.
EVENT:Sized	Пользователь изменил размеры окна.
EVENT:Restore	Пользователь восстанавливает предыдущие размеры окна. Оператор CYCLE прерывает изменение размеров.
EVENT:Restored	Пользователь восстановил предыдущие размеры окна.
EVENT:Maximize	Пользователь устанавливает максимальные размеры окна. Оператор CYCLE прерывает изменение размеров.
EVENT:Maximized	Пользователь установил максимальные размеры окна.
EVENT:Iconize	Пользователь уменьшает размеры окна до пиктограммы. Оператор CYCLE прерывает изменение размеров.
EVENT:Iconized	Пользователь свернул окно до пиктограммы.
EVENT:Completed	В безостановочном режиме закончена обработка всех объектов в окне.
EVENT:DDErequest	Клиент запросил элемент данных от данного приложения - сервера в процессе динамического обмена данными.
EVENT:DDEadvise	Клиент запросил непрерывное обновление элемента данных от данного приложения - сервера в процессе динамического обмена данными.
EVENT:DDEexecute	Клиент выполнил оператор DDEEXECUTE для данного приложения - сервера в процессе динамического обмена

EVENT:DDEpoke	данными. Клиент прислал незатребованный элемент данных данному приложению - серверу в процессе динамического обмена данными.
EVENT:DDEdata	Сервер динамического обмена данными прислал обновленный элемент д а н н ы х данному приложению - клиенту.
EVENT:DDEclosed	Сервер динамического обмена данными прервал связь с данным приложением - клиентом.

### Пример:

!Изменяемых размеров окно прикладной программы MDI с системным меню кнопками  
!минимизации и максимизации, линейкой состояния, линейками скроллинга, содержащее  
!главное меню и панель инструментов для прикладной программы

```

MainWin  APPLICATION('My Appllcation'),SYSTEM,MAX,ICON('Mylcon.ICO'),STATUS    |
        ,HVSCROLL,RESIZE
        MENUBAR
        MENU('&File'),USE(?FileMenu)
        ITEM('&Open...'),USE(?OpenFile)
        ITEM('&Close'),USE(?CloseFile),DISABLE
        ITEM('E&xit'),USE(?MainExit)
        END
        MENU('&Edtt'),USE(?EditMenu)
        ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut),DISABLE
        ITEM('&Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy),DISABLE
        ITEM('&Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste),DISABLE
        END
        MENU('&Window'),STD(STD:WindowList),LAST
        ITEM('&Tile'),STD(STD:TileWindow)
        ITEM('&Cascade'),STD(STD:CascadeWindow)
        ITEM('&Arrange Icons'),STD(STD:Arrangelcons)
        END
        MENU('&Help'),USE(?HelpMenu)
        ITEM('&Contents'),USE(?HelpContents),STD(STD:HelpIndex)
        ITEM('&Search for Help On...'),USE(?HelpSearch),STD(STD:HelpSearch)
        ITEH('&How to Use Help'),USE(?HelpOnHelp),STD(STD:HelpOnHelp)
        ITEM('&About MyApp...'),USE(?HelpAbout)
        END
        END
        TOOLBAR
        BUTTON('E&xit'),USE(?MainExitButton)

```

```

        BUTTON('&Open'),USE(?OpenButton),ICON(ICON:Open)
    END
END
CODE
OPEN(MainWin)                !Открыть окно APPLICATION
ACCEPT !Вывести окно APPLICATION и прочитать реакцию пользователя
CASE ACCEPTED()              !Какое поле выбрано ?
OF ?OpenFile                  !Пункт "Open"
OROF ?OpenButton              !или кнопка на панели инструментов
    START(OpenFileProc)      !Начать новый процесс
OF ?MainExit                  !Пункт "Exit"
OROF ?MainExitButton          !или кнопка на панели инструментов
    BREAK !прервать цикл ACCEPT
OF ?HelpAbout                 !выбран пункт ABOUT
    HelpAboutProc             !Вызвать процедуру вывода помощи
END
END
CLOSE(MainWin)                !Закрыть окно APPLICATION

```

## WINDOW (объявить диалоговое окно)

```

метка    WINDOW('заголовок')[,AT()][,CENTER][,SYSTEM][,MAX][,ICON()]
[STATUS()][,HLP()][,CURSOR()][,MDI][,MODAL][,MASK][,FONT()]
        [,TIMER()][,ALRT()][,ICONIZE][,MAXIMIZE][,MSG()][,GRAY]
        [,TOOLBOX] [,PALETTE] [,DROPID] [,IMM] [,AUTO]
        [, HSCROLL] [, DOUBLE] [,COLOR()]
        VSCROLL NOFRAME
        HVSCROLL RESIZE
    [ MENUBAR
        объявления нескольких меню и/или пунктов
    END]
    [ TOOLBAR
        объявления управляющих полей
    END]
    объявления управляющих полей
    END

```

### WINDOW

Объявляет документальное или диалоговое окно

метка

Допустимая в Clarion метка. В операторе WINDOW метка обязательна.

заголовок

Задаёт текст заголовка основного окна.

AT

Задаёт начальный размер и расположение окна. Если этот атрибут опущен, то значения по умолчанию устанавливаются библиотечной процедурой.

CENTER

Указывает, что по умолчанию первоначально окно центрируется

относительно порождающего окна. Этот атрибут действует, только если по крайней мере один параметр атрибута AT опущен.

**SYSTEM  
MAX**

Задаёт наличие системного меню.

Задаёт наличие кнопок приведения к максимальному и минимальному размерам и кнопки восстановления “естественного” размера окна.

**ICON**

Задаёт наличие кнопок приведения к максимальному, минимальному и естественному размерам, и указывает имя файла или стандартного идентификатора пиктограммы, высвечиваемой при минимизации размера окна.

**STATUS**

Задаёт наличие линейки состояния внизу окна.

**HLP**

Задаёт “идентификатор подсказки” связанной с этим окном.

**CURSOR**

Задаёт форму курсора мыши, которую он принимает в пределах данного окна. Эта форма курсора наследуется всеми управляющими полями данного окна, если для них явно не задаётся другая форма курсора.

**MDI**

Указывает, что окно соответствует обычным правилам для порожденного MDI-окна.

**MODAL**

Задаёт модальность окна относительно среды Windows и должно быть закрыто прежде, чем пользователь сможет дальше что-либо делать.

**MASK**

Задаёт режим редактирования по шаблону вводимых данных для всех вводных управляющих полей в окне.

**FONT**

Указывает используемый по умолчанию шрифт для всех управляющих полей в окне.

**GRAY**

Указывает, что окно имеет серый фон для использования полей выглядящих объёмными.

**TIMER**

Задаёт генерирование периодических временных событий.

**ALRT**

Указывает какие “горячие” клавиши активны, когда данное окно имеет фокус.

**ICONIZE**

Указывает, что при открытии окно минимизируется до пиктограммы.

**MAXIMIZE**

Указывает, что при открытии окно принимает максимальный размер.

**MSG**

Задаёт строковую константу, содержащую текст, выводимый по умолчанию в линейке состояния для всех управляющих полей данного окна.

**TOOLBOX**

Задаёт, что окно “всегда сверху”, а на объектах в нем никогда не задерживается фокус.

**PALETTE**

Задаёт количество, поддерживаемых аппаратно цветов, используемых при выводе графики в окне.

**DROPID**

Указывает, что данное окно может служить областью, в которой происходит вторая часть операции “перетащить и отпустить” - отпускание объекта.

**IMM**

Задаёт, что для окна при изменении его размеров генерируются соответствующие события.

**AUTO**

Указывает, что значения USE-переменных для объектов данного

	окна	выводятся вновь каждый раз при повторении цикла ACCEPT.
<b>NOFRAME</b>		Задаёт окно без рамки. Окно такого типа не может изменяться в размерах.
<b>RESIZE</b>		Задаёт толстую рамку вокруг окна, которая позволяет изменять размеры окна.
<b>COLOR</b>		Указывает цвет фона для окна и используемый по умолчанию цвет фона и цвет выбранного управляющего объекта в окне.
<b>MENUBAR</b>		Определяет структуру меню (необязательную).
меню и/или пункты		Объявления меню и/или пунктов меню, которые определяют выбор.
<b>TOOLBAR</b>		Определяет структуру панели инструментов (необязательную).
объявления управляющих полей		Определяют объекты, доступные на панели инструментов, или управляющие поля в окне.

Структура WINDOW объявляет документальное или диалоговое окно, которое может содержать поля и использоваться для вывода данных. Когда окно открывается в первый раз, оно остается скрытым до первого выполнения оператора DISPLAY или цикла ACCEPT. Это позволяет выполнить перед выводом окна некоторые изменения в его внешнем виде. Например, посредством установки параметров во время выполнения можно настроить заголовок или размеры окна. Любое открытое ранее в этом исполняемом процессе окно становится недоступным.

Если не указан один из параметров DOUBLE, NOFRAME или RESIZE, окну автоматически устанавливается рамка одинарной толщины. Координаты на экране измеряются в условных единицах (dialog units). Условные единицы определяются как одна четвертая часть средней ширины и одна восьмая средней высоты символа в шрифте заданном атрибутом FONT структуры WINDOW (или системного шрифта, если этот атрибут не задан).

Окно, имеющее атрибут MODAL является модальным относительно системы; оно получает исключительное управление ресурсами компьютера. Это означает, что до тех пор пока не будет закрыто окно, имеющее атрибут MODAL, будет приостановлено выполнение любой программы, выполнявшейся в фоновом режиме. Следовательно, этот атрибут следует использовать, только когда это крайне необходимо. Кроме того, при наличии атрибута MODAL игнорируется атрибут RESIZE, а окно становится перемещаемым.

Окно без атрибута MDI, будучи открыто в MDI-программе, является модальным по отношению к приложению. Это означает, что пользователь должен отреагировать на запрос в этом окне, прежде чем переключиться на любое другое окно в прикладной программе. Однако на другую программу, выполняющуюся в это же время в среде Windows, пользователь может переключиться. Окно без атрибута MDI можно открыть или до, или после открытия окна APPLICATION его может открыть или тот же самый процесс, что и окно приложения, или любой порожденный процесс, открывший MDI-окно, (в этом случае

окно модально по отношению к приложению) или совершенно отдельный процесс (в этом случае окно немодально относительно приложения).

Окно, у которого есть атрибут MDI, это порожденное MDI-окно. Порожденное MDI-окно располагается в границах окна прикладной программы, объявляемого структурой APPLICATION, перемещается вместе с ним автоматически и вообще может исчезать при минимизации до пиктограммы порождающего APPLICATION-окна. Порождаемое MDI-окно является немодальным, т.е. пользователь в любое время может переключиться на верхнее окно в другом исполняемом процессе внутри этой же прикладной программы или на любую другую выполняемую в среде Windows программу. Порожденное MDI-окно не должно быть в том же самом процессе, что и окно всего приложения. Поэтому, любое порожденное MDI-окно, к которому обращаются непосредственно из окна приложения должно располагаться в отдельной процедуре, так что для инициирования нового процесса можно использовать функцию START. После того как запущен новый процесс, в нем можно открыть несколько порожденных MDI-окон.

Линейка меню, заданная в структуре WINDOW с атрибутом MDI, когда на это окно переключается фокус, автоматически объединяется с “глобальным меню” (из структуры APPLICATION), если только структура MENUBAR в APPLICATION или в WINDOW не имеет атрибута NOMERGE. Линейка же меню, описанная в структуре WINDOW без атрибута MDI, никогда не объединяется с “глобальным меню” - она всегда выводится в своем окне”.

Панель инструментов, заданная в структуре WINDOW с атрибутом MDI, когда на это окна переключается фокус, автоматически объединяется с “глобальной панелью” (из структуры APPLICATION), если только структура TOOLBAR в APPLICATION или в WINDOW не имеет атрибута NOMERGE. Панель же инструментов, описанная в структуре WINDOW без атрибута MDI, никогда не объединяется с “глобальной” - она всегда выводится в своем окне”.

Окно с атрибутом TOOLBOX автоматически выводится “всегда сверху”, а на объектах в нем никогда не задерживается фокус (как если бы все они имели атрибут SKIP). Таким образом создается окно, объекты в котором ведут себя так же как объекты на панели инструментов. Обычно окно с атрибутом TOOLBOX раскрывается и обрабатывается в своем собственном процессе.

### Генерируемые события:

EVENT:PreAlertKey	Пользователь нажал указанную атрибутом ALRT горячую клавишу.
EVENT:AlertKey	Пользователь нажал указанную атрибутом ALRT горячую клавишу.
EVENT:CloseWindow	Окно закрывается.

EVENT:CloseDown	Приложение завершается.
EVENT:OpenWindow	Окно раскрывается.
EVENT:LoseFocus	Фокус переключается от данного окна на другой процесс.
EVENT:GainFocus	Данное окно получает фокус от другого процесса.
EVENT:Suspend	Фокус еще остается на данном окне, но управление передается другому процессу для обработки таймерных событий.
EVENT:Resume	Фокус остается на данном окне, и управление снова получает управление.
EVENT:Timer	Переключен атрибут TIMER (произошло таймерное событие).
EVENT:Move	Пользователь перемещает окно. Оператор CYCLE прерывает перемещение.
EVENT:Moved	Пользователь переместил окно.
EVENT:Size	Пользователь изменяет размеры окна. Оператор CYCLE прерывает изменение.
EVENT:Sized	Пользователь изменил размеры окна.
EVENT:Restore	Пользователь восстанавливает предыдущие размеры окна. Оператор CYCLE прерывает изменение размеров.
EVENT:Restored	Пользователь восстановил предыдущие размеры окна.
EVENT:Maximize	Пользователь устанавливает максимальные размеры окна. Оператор CYCLE прерывает изменение размеров.
EVENT:Maximized	Пользователь установил максимальные размеры окна.
EVENT:Iconize	Пользователь уменьшает размеры окна до пиктограммы. Оператор CYCLE прерывает изменение размеров.
EVENT:Iconized	Пользователь свернул окно до пиктограммы.
EVENT:Completed	В безостановочном режиме закончена обработка всех объектов в окне.
EVENT:DDErequest	Клиент запросил элемент данных от данного приложения - сервера в процессе Управляющие структуры и операторы динамического обмена данными.
EVENT:DDEadvise	Клиент запросил непрерывное обновление элемента данных от данного приложения - сервера в процессе динамического обмена данными.
EVENT:DDEexecute	Клиент выполнил оператор DDEEXECUTE для данного приложения - сервера в процессе динамического обмена данными.
EVENT:DDEpoke	Клиент прислал незатребованный элемент данных данному приложению - серверу в процессе динамического обмена данными.
EVENT:DDEdata	Сервер динамического обмена данными прислал обновленный элемент данных данному приложению - клиенту.
EVENT:DDEclosed	Сервер динамического обмена данными прервал связь с данным приложением

**Пример:**

! Порождаемое MDI-окно с системным меню, кнопками минимизации и максимального увеличения,  
 ! линейкой состояния, линейками скроллинга, изменяемых размеров, содержащее линейки меню  
 ! и инструментов, объединяющиеся в линейки меню и инструментов прикладной программы

```
MDIChild WINDOW('Child One'),MDI,SYSTEM,MAX,ICON('WinIcon.ICO'),STATUS |
    ,HVSCROLL,RESIZE
MENUBAR
    MENU('File'),USE(?FileMenu)
    ITEM('Close'),USE(?CloseFile)
END
MENU('Edit'),USE(?EditMenu)
    ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo)
    ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut)
    ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy)
    ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste)
END
END
TOOLBAR
    BUTTON('Cut'),USE(?CutButton),ICON(ICON:Cut),STD(STD:Cut)
    BUTTON('Copy'),USE(?CopyButton),ICON(ICON:Copy),STD(STD:Copy)
    BUTTON('Paste'),USE(?PasteButton),ICON(ICON:Paste),STD(STD:Paste)
END
TEXT,HVSCROLL,USE(Pre:Field)
BUTTON('&OK'),USE(?Exit),DEFAULT
END
```

! Не MDI-окно с системным меню, кнопкой минимизации, линейкой состояния и  
 ! рамкой, не допускающей изменения размеров, содержащее управляющие поля

```
NonMDI WINDOW('Dialog Window'),SYSTEM,MAX,STATUS
    TEXT,HVSCROLL,USE(Pre:Field),MSG('Enter some text here')
    BUTTON('&OK'),USE(?Exit),DEFAULT
END
```

!Системно-модальное окно с рамкой, не допускающей изменения размеров,  
 !содержащее только текст сообщения и кнопку OK

```
ModalWin WINDOW('ModalWindow'),MODAL
    Управляющие структуры и операторы
    ICON(ICON:Exclamation)
    STRING('An ERROR has occurred')
    BUTTON('&OK'),USE(?Exit),DEFAULT
END
```



## Атрибуты структур WINDOW и APPLICATION

### ALRT (установить “горячие клавиши”)

ALRT(код клавиши)

**ALRT** Указывает горячую клавишу активную, когда на данную структуру WINDOW или APPLICATION переключен фокус ввода.

код клавиши      Задающая код клавиши числовая константа или мнемоническое имя клавиши, установленное оператором EQUATE.

Атрибут ALRT задает горячую клавишу активную, когда на данную структуру WINDOW или APPLICATION переключен фокус ввода. Когда пользователь нажимает горячую клавишу, заданную атрибутом ALRT для структуры APPLICATION или WINDOW, генерируется независимое от полей в структуре событие (и функция ACCEPTED, и функция SELECTED возвращают 0. Допустимы несколько атрибутов ALRT в одной структуре APPLICATION или WINDOW.

#### Пример:

Screen	WINDOW,ALRT(F10Key)	!F10 работает для всех полей
END		
CODE		
OPEN(Screen)		!Открыть экран для обработки
ACCEPT		! и обработать все поля
IF KEYCODE() = F10Key		!Контролировать горячую клавишу
CLOSE(Screen)		
RETURN		
END		
END		

### AT (установить положение и размеры окна)

AT([x][,y][,ширина][,высота])

**AT**      Задает начальное положение и размеры окна

x      Целочисленная константа или константное выражение, указывающее начальную координату левого верхнего угла по горизонтали . Если этот параметр опущен, то значение по умолчанию берется из библиотеки во время выполнения.

y      Целочисленная константа или константное выражение, указывающее начальную координату по вертикали левого верхнего угла . Если этот параметр опущен, то значение по умолчанию берется из библиотеки во время выполнения.

ширина      Целочисленная константа или константное выражение, задающее начальную ширину окна. Если этот параметр опущен, то значение по умолчанию

берется из библиотеки во время выполнения.

**высота** Целочисленная константа или константное выражение, задающее начальную высоту окна. Если этот параметр опущен, то значение по умолчанию берется из библиотеки во время выполнения.

Атрибут AT определяет начальное положение и размеры окна APPLICATION или WINDOW. Если любой из параметров опущен, то значение по умолчанию берется из библиотеки во время выполнения.

Координаты x и y отсчитываются относительно левого верхнего угла раstra экрана монитора, если атрибут AT относится к структуре APPLICATION или к структуре WINDOW без атрибута MDI, которая открывается до того, как программой будет открыта структура APPLICATION. И эти координаты отсчитываются относительно левого верхнего угла окна APPLICATION если атрибут AT относится к структуре WINDOW с атрибутом MDI или к структуре WINDOW без атрибута MDI, но которая открывается после того, как уже открыто окно APPLICATION.

Параметры ширина и высота задают размер “клиентской области” или “рабочего пространства” в окне APPLICATION. Это область ниже линейки меню и над линейкой состояния, в которой располагается панель инструментов и открываются порожденные MDI-окна. Для окна, описываемого структурой WINDOW, эти параметры задают размер “рабочей области”, которая может содержать управляющие поля.

Если дополнительно не указан атрибут THOUS, MM или POINTS, то значения содержащиеся в параметрах x, y, длина и ширина, измеряются в условных единицах. Условные единицы определяются как одна четвертая часть средней ширины на одну восьмую средней высоты символа. Размер условной единицы зависит от размера шрифта, используемого для окна. Эти единицы измерения основываются на шрифте, указанном для окна атрибутом FONT или системным шрифтом, заданным оболочкой Windows.

### Пример:

```
! Окно в верхнем левом углу относительно рамки окна APPLICATION
WinOne WINDOW,AT(0,0,380,200),MDI
END
! Окно в верхнем левом углу относительно раstra экрана монитора
WinTwo WINDOW,AT(0,0,380,200)
END
```

**AUTO (автоматическое обновление на экране значения USE-переменной)****AUTO**

Атрибут AUTO указывает, что при каждом выполнении цикла ACCEPT на экране заново выводятся значения USE-переменных всех объектов окна и панели инструментов. Использование этого атрибута влечет за собой некоторое дополнительное увеличение программного кода и времени выполнения, но обеспечивает актуальность высвечиваемых данных без явного выполнения оператора DISPLAY.

**Пример:**

```
WinOne WINDOW,AT(,,380,200),MDI,CENTER,AUTO    !Значения всех полей всегда
высвечивать
        !экранные объекты
END
        CODE
        ACCEPT                                !В цикле ACCEPT автоматически выводятся
значения измененных USE-переменных
END
```

**CENTER (центрировать окно)****CENTER**

Атрибут CENTER показывает, что данное окно центрируется. Окно с атрибутом MDI центрируется относительно окна APPLICATION. Окно APPLICATION центрируется относительно раstra экрана монитора. Окно без атрибута MDI центрируется относительно породившего его окна (окно на котором находился фокус в момент раскрытия не MDI-окна).

Этот атрибут не имеет значения, если не опущен хотя бы один параметр атрибута AT. Это означает, что атрибут CENTER обеспечивает значения по умолчанию для опущенных параметров атрибута AT.

**Пример:**

```
        ! Окно центрируется относительно окна APPLICATION
WinOne WINDOW,AT(,,380,200),MDI,CENTER
END

        ! Окно центрируется относительно породившего его
WinTwo WINDOW,AT(,,380,200),CENTER
END
```

**COLOR (установить цвет окна)**

**COLOR**( цвет [ , выбранный\_передн ] [ ,выбранный\_фон ] )

**COLOR**

Указать цвета окна.

цвет

Целочисленная константа типа LONG или ULONG, или задающая константу метка соответствия, содержащая в трех младших байтах красную, зеленую и синюю компоненты, составляющие цвет; или метка соответствия для стандартного в Windows значения цвета.

выбранный\_передн

Целочисленная константа типа LONG or ULONG, или задающая константу метка соответствия, содержащая в трех младших байтах (байты 0, 1, and 2), красную, зеленую и синюю компоненты, составляющие цвет; или метка соответствия для стандартного в Windows значения цвета. Этот параметр задает используемый по умолчанию цвет переднего плана для текста объекта, на который может переключаться фокус ввода.

выбранный\_фон

Целочисленная константа типа LONG или ULONG, или задающая константу метка соответствия, содержащая в трех младших байтах (байты 0, 1, and 2), красную, зеленую и синюю компоненты, составляющие цвет; или метка соответствия для стандартного в Windows значения цвета. Этот параметр задает используемый по умолчанию цвет фона для текста объекта, на который может переключаться фокус ввода.

Атрибут COLOR задает цвет фона в окне и используемые по умолчанию цвет переднего плана и фона для всех объектов в структуре WINDOW, которые не имеют атрибута COLOR.

Операторы EQUATE для стандартных в Windows цветов, содержатся в файле EQUATES.CLW. Для видеоконтроллера используемого при выполнении программы Windows автоматически находит наиболее подходящий заданному цвет. В Панели Управления Windows пользователь может изменить настройку стандартных цветов. При этом все объекты, для которых использовались стандартные для Windows цвета будут окрашены в новые цвета.

**Пример:**

```
WinOne WINDOW,AT(0,0,160,400),COLOR(00FF0000h,0000FF00h,000000FFh)
```

!Синий фон, Зеленый передний план для активизируемых полей, и красный фон для них  
END

## CURSOR (установить форму курсора мыши)

### CURSOR(файл)

**CURSOR** Задаёт форму, которую должен принимать курсор мыши, при попадании в область данного окна.

файл Строковая константа, содержащая имя файла с расширением .CUR или мнемоническое имя стандартной в Windows формы курсора.

Атрибут CURSOR задаёт форму, которую должен принимать курсор мыши, когда он располагается в данном окне. И эта форма курсора по умолчанию сохраняется и для всех управляющих полей в окне, если для них явно не указана другая форма.

Стандартные для Windows формы курсора, мнемонические имена которых содержатся в файле EQUATES.CLW:

CURSQR:None	Нет курсора
CURSOR:Arrow	Обычный курсор в виде стрелки
CURSOR:IBeam	Курсор в виде заглавной буквы I похожий на двутавр
CURSOR:Wait	Песочные часы
CURSOR:Cross	Курсор в виде большого символа плюс
CURSOR:UpArrow	Курсор в виде стрелки направленной вверх
CURSOR:Size	Курсор в виде четырех стрелок, направленных в разные стороны.
CURSOR:Icon	Пиктограмма в рамке
CURSOR:SizeNWSE	Стрелки в направлении северо-запад - юго-восток
CURSOR:SizeNESW	Стрелки в направлении северо-восток - юго-запад
CURSOR:SizeWE	Стрелки в направлении запад - восток
CURSOR:SizeNS	Стрелки в направлении север - юг
CURSOR:DragWE	Стрелки в направлении запад - восток

### Пример:

```
! Окно с курсором мыши в виде большого символа плюс
WinOne WINDOW,CURSOR('CURSOR:Cross')
END
```

## DOUBLE, NOFRAME, RESIZE (установить для окна тип рамки)

**DOUBLE**  
**NOFRAME**  
**RESIZE**

Атрибуты DOUBLE, NOFRAME и RESIZE задают тип рамки для окна, отличной от используемой по умолчанию рамки одинарной толщины. Атрибут DOUBLE задает вокруг окна рамку двойной толщины, а NOFRAME указывает, что у окна нет рамки. Окно с такими рамками не может изменять размеры.

Атрибут RESIZE задает вокруг окна толстую рамку. Это единственный тип рамки, при котором пользователь может динамически изменять размеры окна. Для окна, имеющего атрибут MODAL, атрибут RESIZE игнорируется.

### Пример:

!Окно с рамкой одинарной толщины

```
Win1 WINDOW
END
```

!Окно изменяемых размеров

```
Win2 WINDOW,RESIZE
END
```

!Окно с рамкой двойной толщины

```
Win3 WINDOW,DOUBLE
END
```

!Окно без рамки

```
Win4 WINDOW,NOFRAME
END
```

## FONT (установить для окна шрифт)

**FONT**([начертание][,размер][,цвет][,стиль])

<b>FONT</b>	Задаёт шрифт, используемый по умолчанию для окна.
начертание	Строковая константа, содержащая название шрифта. Если этот параметр опущен, то используется системный шрифт.
размер	Целочисленная константа, содержащая размер шрифта в пунктах Окна и меню(1 пункт 1/72 дюйма). Если этот параметр опущен, то используется размер системного шрифта, используемого по умолчанию.
цвет	Целочисленная константа типа LONG, содержащая в младших трех байтах значения для красной, зеленой и голубой составляющих цвета или

мнемоническое имя, задаваемое оператором EQUATE  
для стандартных в Windows значений, определяющих цвета.

стиль

Целочисленная константа или константное выражение или  
мнемоническая метка, задающая толщину и стиль букв шрифта.

В структурах WINDOW и APPLICATION атрибут FONT задает шрифт для вывода содержимого всех управляющих полей, которые не имеют своего атрибута FONT.

Параметр начертание может указывать любой шрифт зарегистрированный в системе Windows. В файле EQUATES.CLV содержатся значения для стандартных стилей. Значения в диапазоне от 0 до 1000 задают “яркость” шрифта. Это значение можно добавить к величине, означающей курсив, подчеркнутый или перечеркнутый шрифт. В файле EQUATES.CLV содержатся значения:

```
FONT:thin    EQUATE (100)
FONT:regular EQUATE (400)
FONT:bold    EQUATE (700)
FONT:italic  EQUATE (01000H)
FONT:underline EQUATE (02000H)
FONT:strikeout EQUATE (04000H)
```

#### Пример:

!Окна, использующее шрифт Times New Roman размером 14

```
Win1    WINDOW, FONT('Times New Roman', 14, 00H)
        END
```

!Окна, использующее шрифт Times New Roman размером 14, полужирный курсив

```
Win2    WINDOW, FONT('Times New Roman', 14, 00H, FONT:italic+FONT:bold)
        END
```

## GRAY (установить фон для объемных полей)

### GRAY

Атрибут **GRAY** означает, что окно имеет серый фон, подходящий для использования с объемными управляющими полями. Все поля в структуре WINDOW, имеющей атрибут GRAY, автоматически имеют трехмерное изображение. На инструментальной панели поля всегда имеют трехмерное изображение, и без атрибута GRAY.

Трехмерное изображение полей можно выключить оператором SET3DLOOK.

#### Пример:

!Окно с объемными полями

```
Win1    WINDOW, GRAY
        END
```

**HLP (установить для окна идентификатор диалоговой справки)****HLP(идентификатор справки)**

**HLP**        Задает идентификатор интерактивной справки для структур APPLICATION, WINDOW или управляющего поля.

идентификатор справки        Строковая константа, указывающая “ключ”, используемый для обращения к системе справки. Это может быть или ключевое слово системы справки, или “контекстная строка”.

Атрибут HLP задает идентификатор интерактивной справки для структур APPLICATION или WINDOW. Каждый раз, когда пользователь нажимает клавишу F1, среда Windows автоматически выводит систему справки, если она доступна.

Если пользователь нажимает F1, запрашивая систему справки, когда раскрыто только APPLICATION-окно, и нет активизированного меню, то для нахождения текста справки используется идентификатор, заданный атрибутом HLP в структуре APPLICATION. В противном случае библиотечная процедура использует идентификатор справки, заданный атрибутом активного меню или самого верхнего управляющего поля или окна, выполняя поиск в иерархическом порядке до тех пор, пока объект с заданным идентификатором не будет найден. Идентификатор справки структуры APPLICATION находится наверху этой иерархии.

Идентификатор справки может содержать ключевое слово системы справки или строку контекста. Ключевое слово системы справки это слово или фраза, которая высвечивается в списке окна Help Search. Если, когда пользователь нажимает F1, это ключевое слово относится только к одному разделу файла подсказок, то он открывается на этом разделе; если относится к нескольким разделам, то пользователю раскрывается диалоговое окно.

Контекстная строка отличается в идентификаторе справки символом тильда (~), за которым следует уникальная последовательность символов (не допускается наличие в ней пробелов), связанная точно с одним разделом справки. Если тильда пропущена, то подразумевается, что идентификатор справки должен быть ключевым словом справочной системы. При нажатии пользователем клавиши F1 система справки раскрывается на конкретном разделе, связанном с контекстной строкой.

**Пример:**

```
!Окно с подсказкой по контекстной строке
Win1  WINDOW,HLP('~Win1Help')
      END
!Окно с подсказкой по ключевому
Win2  WINDOW,HLP('Window One Help')
      END
```



HSCROLL, VSCROLL, HVSCROLL (установить линейки скроллинга)  
HSCROLL  
VSCROLL  
HVSCROLL

При использовании атрибутов HSCROLL, VSCROLL и HVSCROLL в окно APPLICATION или WINDOW помещаются линейки скроллинга. Когда указан атрибут HSCROLL внизу окна помещается линейка горизонтального скроллинга, если указан VSCROLL, то с правой стороны окна располагается линейка вертикального скроллинга, а если HVSCROLL - то обе линейки.

Линейка вертикального скроллинга позволяет пролистывать содержимое окна вниз и вверх с помощью мыши. А с помощью линейки горизонтального скроллинга можно смещать содержимое влево и вправо. Линейки скроллинга появляются всякий раз, когда какая-либо часть подлежащей скроллингу области окна выходит за пределы раstra видеомонитора и структура APPLICATION или WINDOW имеет атрибут RESIZE.

### Пример:

```
!Окно с линейкой горизонтального скроллинга
Win1 WINDOW,HSCROLL,RESIZE
END
!Окно с линейкой вертикального скроллинга
Win2 WINDOW,VSCROLL,RESIZE
END
!Окно с обеими линейками
Win2 WINDOW,HVSCROLL,RESIZE
END
```

## ICON (установить для окна пиктограмму)

ICON([файл])

**ICON**                      Задает пиктограмму, которая должна высвечиваться для данного APPLICATION-окна или документального окна.

файл                      Строковая константа, содержащая имя файла .ICO, или мнемоническая метка стандартной пиктограммы из системы Windows, которая подлежит выводу на экран..

Атрибут ICON задает пиктограмму для данного окна, описываемого структурой APPLICATION или WINDOW. Кроме того, он задает наличие в структуре APPLICATION или WINDOW управляющего поля минимизации окна. Кнопка минимизации выводится в верхнем правом углу в виде указывающего вниз треугольника. Когда пользователь щелкнет кнопкой мыши, указав курсором на эту кнопку, окно свернется до пиктограммы, не останавливая выполнения процесса в этом окне. Если минимизируется окно APPLICA-

TION или не MDI окно, то пиктограмма, задаваемая параметром файл высвечивается на поверхности системного “рабочего стола”; если минимизируется окно с атрибутом MDI, то пиктограмма высвечивается в пределах окна APPLICATION.

В файле EQUATES.C LW содержатся следующие мнемонические имена пиктограмм:

ICON:None	нет пиктограммы
ICON:Application	
ICON:Hand	
ICON:Question	?
ICON:Exclamation	!
ICON:Asterisk	*
ICON:Pick	
ICON:Save	
ICON:Print	
ICON:Paste	
ICON:Open	
ICON:New	
ICON:Help	
ICON:Cut	
ICON:Copy	
ICON:VCRtop	>>
ICON:VC Rrewind	<<
ICON:VCRback	<
ICON:VCRplay	>
ICON:VCRfastforward	>>
ICON:VCRbottom	<<
ICON:VCRlocate	?

### Пример:

```
Win2      !Окно с кнопкой минимизации:
          WINDOW,ICON('Mylcon.ICO')
          END
```

## ICONIZE (установить, что в момент раскрытия выводится пиктограмма)

### ICONIZE

Атрибут ICONIZE указывает, что окно раскрывается минимизированным до пиктограммы, заданной атрибутом ICON. Если минимизируется окно APPLICATION или не MDI окно, то пиктограмма, задаваемая параметром файл атрибута ICON высвечивается на поверхности системного “рабочего стола”; если минимизируется окно с атрибутом MDI, то пиктограмма высвечивается в пределах окна APPLICATION.

**Пример:**

```
!Окно с кнопкой минимизации, раскрывающееся в виде пиктограммы:  
Win2 WINDOW,ICON('Mylcon.ICO'),ICONIZE  
END
```

**IMM (немедленная генерация события при изменении размера окна)****IMM**

Атрибут IMM для структур WINDOW и APPLICATION задает, что, как только пользователь начинает изменять размеры окна или перемещать его, немедленно происходит генерация события. Перед выполнением действия на экране генерируется одно из следующих событий:

```
EVENT:Move  
EVENT:Size  
EVENT:Restore  
EVENT:Maximize  
EVENT:Iconize
```

Если подпрограмма, обрабатывающая эти события выполнит оператор CYCLE, то действие не выполняется. Тем самым можно не позволить пользователю переместить окно или изменить его размеры. Как только действие выполнено, генерируется одно из следующих событий:

```
EVENT:Moved  
EVENT:Sized  
EVENT:Restored  
EVENT:Maximized  
EVENT:Iconized
```

Несколько генерируемых после выполнения действия событий обусловлены тем, что некоторые действия имеют множественные результаты. Например, если пользователь щелкнул мышью на кнопке максимизации, генерируется событие EVENT:Maximize. Если при обработке этого события не было выполнения оператора CYCLE, то действие выполняется, затем генерируются события EVENT:Maximized, EVENT:Moved и EVENT:Sized. Это происходит потому, что окно приняло максимальные размеры, что также подразумевает его перемещение и изменение размеров.

**Пример:**

```

Win2    WINDOW('Some Window'),AT(58,11,174,166),MDI,DOUBLE,MAX,IMM
        LIST,AT(109,48,50,50),USE(?List),FROM('Que'),IMM
        BUTTON('&Ok'),AT(111,108,,),USE(?Ok)
        BUTTON('&Cancel'),AT(111,130,,),USE(?Cancel)
        END
        CODE
        OPEN(Win2)
        ACCEPT
        CASE EVENT()
        OF EVENT:Move                !не дать пользователю переместить окно
        CYCLE
        OF EVENT:Maximized            !При максимизации
        ?List{PROP:Height} = 100    ! изменить размер списка
        OF EVENT:Restored            !При восстановлении из пиктограммы
        ?List{PROP:Height} = 50    ! изменить размер списка
        END
        END

```

**MASK (установить режим ввода данных по шаблону)****MASK**

Атрибут MASK задает для всех полей в окне режим ввода данных с редактированием по шаблону. Это означает, что в процессе ввода пользователем данных каждый символ автоматически проверяется на соответствие шаблону данного поля, для того чтобы обеспечить корректный ввод данных (только цифры для числового шаблона и т.п.). Этот режим вынуждает пользователя вводить данные в формате, заданном шаблоном поля.

В случае, когда этот атрибут опущен, Windows допускает свободный ввод данных в поле. Свободный ввод означает, что данные пользователя форматируются в соответствии с шаблоном управляющего поля только после их ввода. Это позволяет пользователям вводить данные как им хочется, а после ввода данные автоматически форматируются в соответствии с шаблоном данного поля. Если пользователь вводит данные в формате, не соответствующем заданному для этого поля шаблону, то библиотечные процедуры пытаются определить использованный пользователем формат, и преобразовать данные в соответствии с шаблоном поля. Например, если пользователь вводит с клавиатуры "January 1, 1995" в поле с шаблоном @D1, библиотечная процедура преобразует введенную дату в вид: "1/1/95". Это преобразование происходит только после того, как пользователь закончит ввод даты и переместит фокус ввода на другое поле. Если же библиотечная процедура не может определить использованный пользователем формат, то значение USE-переменной не обновляется. Затем подается звуковой сигнал, фокус ввода переключается на поле, где вводились данные, чтобы ввести их заново.

**Пример:**

```
!Окно с включенным режимом ввода по шаблону
Win2 WINDOW,MASK
END
```

**MAX (установить наличие кнопки максимизации окна)****MAX**

Атрибут MAX задает наличие кнопки максимизации размеров окна APPLICATION и или окна, описываемого структурой WINDOW. Кнопка максимизации выводится в верхнем правом углу или в виде указывающего вверх треугольника, или в виде двух треугольников друг над другом: верхний указывает вверх, а нижний - вниз. Когда пользователь щелкнет кнопкой мыши, указав курсором на эту кнопку, окно APPLICATION или не MDI-окно расширяется так, чтобы занимать весь экран монитора; MDI-окно расширяется таким образом, чтобы занимать все окно APPLICATION. Кнопка максимизации в окне, расширенном до максимальных размеров, выглядит в виде двух треугольников друг над другом, указывающих верхний - вверх, а нижний - вниз. Повторный щелчок мышью на такой кнопке, - и окно возвращается к своему предыдущему размеру, а кнопка выводится в виде одного треугольника, указывающего вверх.

**Пример:**

```
!Окно с кнопкой максимизации:
Win2 WINDOW,MAX
END
```

**MAXIMIZE (установить раскрытие окна максимального размера)****MAXIMIZE**

Атрибут MAXIMIZE указывает, что окно раскрывается увеличенным до максимально возможного размера. Будучи увеличенным до максимальных размеров, окно APPLICATION или не MDI-окно расширяется так, чтобы занимать весь экран монитора; MDI-окно расширяется таким образом, чтобы занимать все окно APPLICATION. В максимально увеличенном окне кнопка максимизации выглядит как два треугольника друг над другом, указывающих вверх и вниз.

**Пример:**

```
!Окно с кнопкой максимизации, раскрывающееся максимального размера
Win2 WINDOW,MAX,MAXIMIZE
END
```

## MDI (установить для окна тип MDI)

### MDI

Атрибут MDI задает структуру WINDOW, которая описывает окно “порожденное” по отношению к окну APPLICATION. Порожденные MDI-окна располагаются в пределах окна APPLICATION - они могут перемещаться только в пределах границ окна APPLICATION. Порожденные MDI-окна автоматически перемещаются при перемещении окна APPLICATION и могут быть скрыты совсем при минимизации порождающего окна. Структуру WINDOW с атрибутом MDI нельзя открыть пока не открыта порождающая их структура APPLICATION.

Окна без атрибута MDI функционируют независимо от какой-либо ранее открытой структуры APPLICATION. Однако такое окно (без атрибута MDI) деактивизирует окно APPLICATION или любое порожденное MDI-окно, если находится с ним в одном и том же процессе. Это делает не MDI-окно, открытое в MDI-программе окном модальным по отношению к прикладной программе, которое эффективно выключает выполнение прикладной программы на то время, пока не MDI-окно раскрыто перед пользователем. Однако, оно не мешает пользователю переключиться на другую прикладную программу, выполняющуюся в среде Windows.

#### Пример:

```
!Порождаемое MDI-окно:  
Win2 WINDOW,MDI  
END
```

## MODAL (установить системную модальность окна)

### MODAL

Атрибут MODAL указывает, что окно является “модальным относительно системы”. Это означает, что пока фокус ввода на данном окне, ни на какое другое окно (в этой же или в выполняющейся параллельно программе) нельзя переключить фокус ввода - т.е. программа, открывшая это окно имеет в данный момент исключительный контроль над ресурсами компьютера. Обычно, окно с атрибутом MODAL используется для вывода сообщения об ошибке или сообщения, которое требует немедленного привлечения внимания пользователя, как например: “Please insert disk in drive A:”.

Окно без атрибута MODAL может быть модальным относительно прикладной программы или немодальным вообще. Модальным относительно прикладной программы является окно без атрибута MDI, раскрытое в MDI-программе. Модальное по отношению к прикладной программе не позволяет пользователю переключиться на любое другой

процесс в прикладной программе, но не запрещает переключиться на другую программу, выполняющуюся в это же время в среде Windows.

Немодальным является порожденное MDI-окно не имеющее атрибута MODAL. Из немодального окна с помощью мыши, нажатием последовательности клавиш или выбором пункта в меню можно выбрать другое окно или другой процесс. В этом случае другое окно приобретает фокус ввода и становится верхним на экране. Но даже из немодального окна нельзя выбрать окно, раскрытое ранее в исполняющемся процессе.

### Пример:

!Окно, модальное по отношению к системе:

```
Win2 WINDOW,MODAL
END
```

## MSG (установить сообщение на линейке состояния)

### MSG(текст)

<b>MSG</b>	Задаёт текст, который следует вывести в линейке состояния.
текст	Строковая константа, содержащая сообщение, которое следует вывести на линейке состояния.

Атрибут MSG задает в структурах APPLICATION и WINDOW текст, который следует вывести в первой зоне линейки состояния, в то время, когда фокус ввода переключен на поле, не имеющее своего собственного атрибута MSG.

### Пример:

```
WinOne WINDOW,AT(0,0,160,400),MSG('Enter Data')!Сообщение по умолчанию
COMBO(@S8),AT(120,120,20,20),USE(?C1),FROM(Q1:F2),MSG('Enter or Select')
LIST,AT(120,0,20,20),USE(?L1),FROM(Que1),MSG('Select One')
SPIN(@N8.2),AT(280,0,20,20),USE(SpinVar1),FROM(Q),MSG('Choose One')
TEXT,AT(20,0,40,40),USE(E2) !Сообщение по умолчанию
ENTRY(@S8),AT(100,200,20,20),USE(E2) !Сообщение по умолчанию
CHECK('&A'),AT(0,120,20,20),USE(?C7),MSG('On or Off')
OPTION('Option 1'),USE(OptVar),MSG('Pick One or Two')
RADIO('Radio 1'),AT(120,0,20,20),USE(?R1)
RADIO('Radio 2'),AT(140,0,20,20),USE(?R2)
END
END
```

**PALETTE (установить аппаратное количество цветов)****PALETTE(цвета)**

**PALETTE**                      Указывает аппаратное количество цветов, которыми отображаются различные части окна.

цвета                              Целочисленная константа, указывающая аппаратное количество цветов, которыми отображаются различные части окна.

Атрибут **PALETTE** структуры **WINDOW** определяет сколько цветов в аппаратной палитре следует использовать для данного окна, когда оно является фоновым. Этот атрибут применим только для аппаратных графических режимов, в которых используется палитра и имеются в наличии свободные цвета (не зарезервированные системой) - практически это означает режим с 256 цветами. Этот атрибут включает использование частичного набора цветов для вывода графики. При 24-битовом представлении цвета (16.7М) аппаратная палитра не используется. Не рекомендуется использовать значение атрибута **PALETTE** больше 256.

**Пример:**

```
WinOne      WINDOW,AT(0,0,160,400),PALETTE(256) !Выводить в режиме 256-ти цветов
             IMAGE,AT(120,120,20,20),USE(ImageField)
             END
```

**STATUS (установить наличие линейки состояния)****STATUS(ширина)**

**STATUS**                      Задает наличие в окне линейки состояния.

ширина                        Список целочисленных констант (разделенных запятыми), указывающих размер каждой области на линейке состояния. Если этот параметр опущен, то линейка состояния состоит из одной области во всю ширину линейки.

Атрибут **STATUS** указывает наличие линейки состояния в основании окна **APPLICATION** или **WINDOW**. Линейка состояния для **MDI**-окна всегда высвечивается внизу окна **APPLICATION**. Для окна без атрибута **MDI** линейка состояния высвечивается внизу самого окна. Если в структуре **APPLICATION** или **WINDOW** отсутствует атрибут **STATUS**, то линейка состояния для такого окна не выводится.

Линейку состояния можно с помощью параметра **ширина** разделить на ряд областей. Размер каждой области задается в условных единицах, если только атрибутом **THOUS**, **ММ** или **POINTS** не задана другая единица измерения. Отрицательное значение означает, что область расширяемая, но имеет минимальную ширину, указанную абсолютной



величиной параметра. Если параметр ширина не указан, то создается линейка состояния из одной расширяемой области без ограничения минимального размера, что соответствует записи STATUS(-1).

Первая зона линейки состояния всегда используется для отображения сообщения, задаваемого атрибутом MSG. Строка, указанная в атрибуте MSG, высвечивается на линейке состояния на протяжении всего времени, в течение которого на соответствующем управляющем поле сосредоточен фокус ввода. Переключение фокуса ввода на поле, не имеющее атрибута MSG, приводит к тому, что линейка возвращается к своему первоначальному состоянию (или пробельному, или отображению предыдущего сообщения в данной области).

Используя оператор MESSAGETEXT можно поместить текст в любую область линейки состояния. А получить текст, высвечиваемый в какой-либо области можно при помощи функции GETMESSAGETEXT. Текст отображается на линейке состояния до тех пор, пока не будет замещен другим сообщением.

Конфигурацию линейки состояния, кроме того, можно изменить динамически, обращаясь во время выполнения к функции SETSTATUSBAR.

### Пример:

```
!Окно APPLICATION с линейкой состояния из одной области
MainWin APPLICATION,STATUS
END
```

```
!Окно WINDOW с линейкой состояния из двух областей
Win1 WINDOW,STATUS(160,160)
END
```

## SYSTEM (установить наличие системного меню)

### SYSTEM

**SYSTEM** Задает наличие системного меню Windows

Атрибут SYSTEM задает наличие в окне APPLICATION или WINDOW системного меню Windows (иначе называемого управляющим меню). Это меню содержит стандартные для Windows пункты, как-то Close, Minimize, Maximize (данное окно) и Switch To (переключиться на другое окно). Реальное доступность этих пунктов для данного окна зависит от значений атрибутов, указанных для него.

### Пример:

```

!Окно APPLICATION с системным меню:
MainWin  APPLICATION,SYSTEM
        END
!Окно WINDOW с системным меню:
Win1     WINDOW,SYSTEM
        END

```

## TOOLBOX (поведение инструментальной панели)

### TOOLBOX

Атрибут TOOLBOX указывает, что окно “всегда сверху”. Ни само окно, ни объекты в нем не удерживают фокус ввода. В этом случае поведение объектов в окне выглядит как если бы все они имели атрибут SKIP. Обычно окно с атрибутом TOOLBOX раскрывается в своем собственном исполняемом процессе, чтобы обеспечить набор инструментов (сервисных кнопок, функций) для активного окна. Когда курсор мыши указывает на объект, значение атрибута MSG этого объекта выводится в строке состояния.

#### Пример:

```

MainWin  PROGRAM
        APPLICATION('My Application')
        MENUBAR
        MENU('File'),USE(?FileMenu)
        ITEM('E&xit'),USE(?MainExit),LAST
        END
        MENU('Edit'),USE(?EditMenu)
        ITEM('Use Tools'),USE(?UseTools)
        ...
Pre:Field  STRING(400)
UseToolsThread  BYTE
ToolsThreadBYTE
        CODE
        OPEN(MainWin)
        ACCEPT
        CASE ACCEPTED()
        OF ?MainExit
        BREAK
        OF ?UseTools
        UseToolsThread = START(UseTools)
        ..

UseTools  PROCEDURE                                !Процедура, использующая панель
инструментов
MDIChild  WINDOW('Use Tools Window'),MDI
        TEXT,HVSCROLL,USE(Pre:Field)

```

```

        BUTTON('&OK'),USE(?Exit),DEFAULT
    END
    CODE
    OPEN(MDICHild)                !Открыть окно
    DISPLAY                        ! и вывести данные
    ToolsThread = START(Tools)    !Открыть панель инструментов
    ACCEPT
    CASE EVENT()                  !Проверить определенные
пользователем события
        OF 401h                  ! для объектов на панели
инструментов
            Pre:Field += ' ' & FORMAT(TODAY(),@D1)    !присоединить дату к концу поля
        OF 402h
            Pre:Field += ' ' & FORMAT(CLOCK(),@T1)    ! присоединить время к концу
поля
        END

    CASE ACCEPTED()
    OF ?Exit
        POSTEVENT(400h,,ToolsThread)    !Дать сигнал закрыть панель
инструментов
        BREAK
    ..
    CLOSE(MDICHild)

Tools Win1    PROCEDURE                !Процедура обработки панели инструментов
    WINDOW('Tools'),TOOLBOX
    BUTTON('Date'),USE(?Button1)
    BUTTON('Time'),USE(?Button2)
    END
    CODE
    OPEN(Win1)
    ACCEPT
    IF EVENT() = 400h THEN BREAK. !Проверить сигнал закрыть панель
инструментов
    CASE ACCEPTED()
    OF ?Button1
        POSTEVENT(401h,,UseToolsThread)    !Дать сигнал поставить дату
    OF ?Button2
        POSTEVENT(402h,,UseToolsThread)    ! Дать сигнал поставить время
    ..
    CLOSE(Win1)

```

## **TIMER (установить генерацию переодических событий)**

**TIMER**(интервал)

### **TIMER**

Задаёт генерацию периодических событий.

интервал                      Целочисленная константа или константное выражение, указывающее интервал в сотых долях секунды между периодическими событиями. Максимальная величина интервала равна 6553 (ограничение присущее Windows).

Атрибут TIMER устанавливает генерацию периодических событий по истечении заданного параметром интервал времени. Файл EQUATES.CLW содержит мнемоническое имя IVENT:TIMER, которое присвоено периодически генерируемому событию. При возникновении периодического события функции ACCEPTED() и SELECTED() обе возвращают ноль. Функция FIELD() возвращает номер управляющего поля, на которое переключен фокус ввода.

### Пример:

```
RunClock PROCEDURE
ShowTime LONG
    !Окно генерацией периодического события через каждую секунду:
Win1    WINDOW,TIMER(100)
        STRING(@T4),USE>ShowTime)
    END
CODE
OPEN(Win1)
ShowTime = CLOCK()
ACCEPT
CASE EVENT()
OF EVENT:Timer
    ShowTime = CLOCK()
    DISPLAY
END
END
CLOSE(Win1)
```

## Структуры **MENUBAR** и **TOOLBAR**

### **MENUBAR** (объявить структуру спускающегося меню)

```
MENUBAR[,NOMERGE]
  [MENU()
    [ITEM()]
    [MENU()
      [ITEM()]
    END]
  END]
  [ITEM()]
END
```

<b>MENUBAR</b>	Объявляет структуру спускающегося меню в окне APPLICATION или WINDOW.
<b>NOMERGE</b>	Задаёт возможность или невозможность объединения меню.
<b>MENU</b>	Пункт меню вместе со связанным с ним окном, содержащим вложенные пункты меню.
<b>ITEM</b>	Пункт меню.

Оператор **MENUBAR** объявляет для окна **APPLICATION** или **WINDOW** структуру спускающегося меню. Структура **MENUBAR** должна находиться в исходном тексте программы до объявления структуры **TOOLBAR** или объявления какого-либо управляющего поля.

В структуре **APPLICATION** оператор **MENUBAR** объявляет главное (или глобальное) спускающееся меню, относящееся ко всей программе. Это меню всегда активно и доступно из всех порожденных **MDI**-окон (если только собственное меню окна не имеет атрибута **NOMERGE**). Если атрибут **NOMERGE** указан в структуре **MENUBAR** окна **APPLICATION**, то в этом случае меню является локальным и выводится только когда нет открытых **MDI**-окон.

В структуре **WINDOW** с атрибутом **MDI** оператор **MENUBAR** пункты меню, которое автоматически объединяются с глобальным меню. Таким образом в то время, как на порожденное **MDI**-окно переключен фокус ввода, активны и глобальное меню и меню окна. После переключения фокуса на другое окно пункты меню предыдущего окна удаляются из глобального меню. Если же атрибут **NOMERGE** указан в структуре **MENUBAR** **MDI**-окна, то его меню замещает и перекрывает глобальное меню.

Спускающееся меню окна без атрибута **MDI** никогда не объединяется с глобальным меню. Она всегда выводится в самом окне, а не в окне **APPLICATION**, которое могло быть раскрыто ранее.

События, генерируемые пунктами локального меню обычным образом передаются циклу АССЕРТ. События же, генерируемые элементами глобального меню, передаются активному АССЕРТ-циклу процесса, который открыл структуру APPLICATION (для обычной прикладной программы с несколькими процессами это означает, что события передаются АССЕРТ-циклу самой прикладной программы).

Динамические изменения пунктов меню, которые касаются только активного в данный момент окна, влияют только на вид меню в данный момент, даже если изменяются глобальные элементы. Изменения же, сделанные в пунктах глобального меню, когда текущим является окно APPLICATION, или в пунктах, которые касаются окна APPLICATION, влияют на глобальную часть всех меню, независимо от того открыто то меню или нет.

Когда линейка меню окна объединяется с линейкой меню окна APPLICATION, сначала выводятся пункты глобального меню, за которыми следуют пункты локального меню, если только для отдельных пунктов не указаны атрибуты FIRST и LAST.

### Пример:

```
!Окно прикладной программы с MDI, в котором находится главное меню программы
MainWin APPLICATION('My Application')
  MENUBAR
    MENU('File'),USE(?FileMenu)
    ITEM('Open...'),USE(?OpenFile)
    ITEM('Close'),USE(?CloseFile),DISABLE
    ITEM('E&xit'),USE(?MainExit),LAST
  END
  MENU('Edit'),USE(?EditMenu)
    ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut),DISABLE
    ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy),DISABLE
    ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste),DISABLE
  END
  MENU('Help'),USE(?HelpMenu),LAST
    ITEM('Contents'),USE(?HelpContents),STD(STD:HelpIndex)
    ITEM('Search for Help On...'),USE(?HelpSearch),STD(STD:HelpSearch)
    ITEM('How to Use Help'),USE(?HelpOnHelp),STD(STD:HelpOnHelp)
    ITEM('About MyApp...'),USE(?HelpAbout)
  END
END
END
END
!Порожденное MDI-окно с линейкой меню, объединяемой с линейкой меню
прикладной программы
MDIChild WINDOW('Child One'),MDI
  MENUBAR
```

```

MENU('File'),USE(?FileMenu)    !Объединяется с меню File
ITEM('Close'),USE(?CloseFile)  !Заменяет пункт глобального меню
ITEM('Pick...'),USE(?PickFile) !Добавляется к пунктам глобального меню
END
MENU('Edit'),USE(?EditMenu)    !Добавляется к меню Edit
ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo) !Добавляется к меню
!Следующие пункты заменяют пункты глобального меню
ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut)
ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy)
ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste)
END
MENU('Window'),STD(STD:WindowList),LAST
ITEM('Tile'),STD(STD:TileWindow)
ITEM('Cascade'),STD(STD:CascadeWindow)
END
END
TEXT,HVSCROLL,USE(Pre:Field)
BUTTON('&OK'),USE(?Exit),DEFAULT
END
!MDI-окно со своим собственным меню, замещающим главное меню
MDIChild2 WINDOW('Dialog Window'),MDI,SYSTEM,MAX,STATUS
MENUBAR,NOMERGE
MENU('File'),USE(?FileMenu)
ITEM('Close'),USE(?CloseFile)
END
MENU('Edit'),USE(?EditMenu)
ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo)
ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut)
ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy)
ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste)
END
END
TEXT,HVSCROLL,USE(Pre:Field),MSG('Enter some text here')
BUTTON('&OK'),USE(?Exit),DEFAULT
END !He MDI-окно со своим собственным меню
NonMDI WINDOW('Dialog Window'),SYSTEM,MAX,STATUS
MENUBAR
MENU('File'),USE(?FileMenu)
ITEM('Close'),USE(?CloseFile)
END
MENU('Edit'),USE(?EditMenu)
ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo)
ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut)
ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy)
ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste)
END
END

```

```
TEXT,HVSCROLL,USE(Pre:Field),MSG('Enter some text here')
BUTTON('&OK'),USE(?Exit),DEFAULT
END
```

## TOOLBAR (объявить панель инструментов)

```
TOOLBAR,[AT()],[CURSOR()],[FONT()],[NOMERGE],[COLOR]
```

управляющие поля

```
END
```

<b>TOOLBAR</b>	Объявляет панель инструментов для окна APPLICATION или WINDOW
<b>AT</b>	Задаёт первоначальные размеры панели инструментов. Если этот параметр опущен, то значения по умолчанию выбираются библиотечной процедурой.
<b>CURSOR</b>	Задаёт форму, в виде которой курсор мыши должен изображаться при позиционировании на панель инструментов. Если этот параметр опущен, то используется форма, заданная атрибутом CURSOR структуры APPLICATION или WINDOW, если и там этот атрибут отсутствует, используется форма курсора принятая в среде Windows по умолчанию.
<b>FONT</b>	Задаёт шрифт используемый по умолчанию для полей на панели инструментов.
<b>NOMERGE</b>	Задаёт возможность или невозможность слияния инструментов с другой панелью.
<b>COLOR</b>	Задаёт цвет фона для структуры TOOLBAR и используемые по умолчанию цвета фона и цвет активизированного поля для объектов в данной структуре TOOLBAR.
управляющие поля	Объявление управляющих полей, которые определяют доступные инструменты.

Структура TOOLBAR объявляет панель инструментов, высвечиваемую в окне APPLICATION или WINDOW. В структуре APPLICATION оператор TOOLBAR определяет инструменты глобальные для данной программы. Если в операторе TOOLBAR структуры APPLICATION задан атрибут NOMERGE, то эти инструменты являются локальными, и высвечиваются, только когда нет раскрытых MDI-окон; в этом случае нет глобальных инструментов. Глобальные инструменты активны и доступны из всех порожденных MDI-окон, если только структура TOOLBAR самого MDI-окна не имеет атрибута NOMERGE. Если же имеет такой атрибут, то инструменты порожденного MDI-окна замещают глобальные инструменты.

В структуре WINDOW с атрибутом MDI оператор TOOLBAR определяет инструменты, которые автоматически присоединяются к глобальной панели инструментов. После чего, и глобальные инструменты и инструменты данного окна активны на протяжении времени,



в течение которого на данное порожденное MDI-окно переключен фокус ввода. После переключения фокуса на другое окно инструменты предыдущего удаляются из глобальной панели. Если в структуре TOOLBAR MDI-окна указан атрибут NOMERGE, то его инструменты замещают и перекрывают глобальные. Инструменты окна без атрибута MDI никогда не объединяются с глобальными инструментами. В этом случае панель инструментов высвечивается в самом окне, а не в ранее открытом окне APPLICATION.

События, генерируемые локальными инструментами обычным образом передаются циклу АСЦЕРТ. События же, генерируемые инструментами глобальной панели, передаются активному АСЦЕРТ-циклу процесса, который открыл структуру APPLICATION. Для обычной прикладной программы с несколькими процессами это означает, что события передаются АСЦЕРТ-циклу самой прикладной программы.

Для управляющих полей на панели инструментов события генерируются обычным образом. Однако на них не переключается фокус ввода и ими нельзя манипулировать с помощью клавиатуры, если только для инструментов не определены горячие клавиши. Как только действие пользователя посредством поля на панели инструментов выполнено, фокус ввода возвращается на окно и локальное управляющее поле на котором он был установлен ранее.

Динамические изменения инструментов, которые касаются активного в данный момент окна, влияют только на вид меню в данный момент, даже если изменяются глобальные инструменты. Изменения же, сделанные в инструментах глобальной панели, когда текущим является окно APPLICATION, или в инструментах, которые касаются окна APPLICATION, влияют на глобальную часть всех инструментальных линеек программы, независимо от того отображаются они в данный момент или нет.

Когда панель инструментов окна объединяется с панелью инструментов окна APPLICATION, сначала выводятся инструменты глобальной панели, за которыми следуют инструменты локальной. Панели объединяются таким образом, что управляющие поля панели окна WINDOW выводятся справа от точки, заданной значением параметра ширина атрибута AT структуры TOOLBAR окна APPLICATION. Высота высвечиваемой панели равна высоте “самого высокого” инструмента, глобального ли, локального ли не имеет значения. Если некоторая часть какого-то инструмента выступает ниже линии панели, то высота панели соответственно увеличивается.

### Пример:

!Порождающее окно прикладной программы с MDI, содержащее главное меню,  
! и панель инструментов для данной программы

```
MainWin  APPLICATION('My Appllcation'),SYSTEM,MAX,ICON('Mylcon.ICO'),STATUS |  
          ,HVSCROLL,RESIZE  
          MENUBAR  
          ITEM('E&xit'),USE(?MainExit)
```

```

    END
    TOOLBAR
    BUTTON('Exit'),USE(?MainExitButton)
    END
END
!Порожденное MDI-окно с локальной панелью инструментов,
! встраивающейся в панель инструментов окна APPLICATION
MDIChild WINDOW('Child One'),MDI
    TOOLBAR
    BUTTON('Cut'),USE(?CutButton),STD(STD:Cut)
    BUTTON('Copy'),USE(?CopyButton),STD(STD:Copy)
    BUTTON('Paste'),USE(?PasteButton),STD(STD:Paste)
    END
    TEXT,HVSCROLL,USE(Pre:Field)
    BUTTON('&OK'),USE(?Exit),DEFAULT
    END
!MDI-окно со своей собственной панелью инструментов, замещающей главную
панель
MDIChild2 WINDOW('Dialog Window'),MDI,SYSTEM,MAX,STATUS
    TOOLBAR,NOMERGE
    BUTTON('Cut'),USE(?CutButton),STD(STD:Cut)
    BUTTON('Copy'),USE(?CopyButton),STD(STD:Copy)
    BUTTON('Paste'),USE(?PasteButton),STD(STD:Paste)
    END
    TEXT,HVSCROLL,USE(Pre:Field),MSG('Enter some text here')
    BUTTON('&OK'),USE(?Exit),DEFAULT
    END

```

## Атрибуты структур *MENUBAR* и *TOOLBAR*

### COLOR (установить цвета для панели инструментов)

**COLOR**( цвет [,активизир\_передн ] [,активизир\_фон ] )

<b>COLOR</b>	Указать цвета.
цвет	Целочисленная константа типа LONG или ULONG, или задающая константу метка соответствия, содержащая в трех младших байтах красную, зеленую и синюю компоненты, составляющие цвет; или метка соответствия для стандартного в Windows значения цвета.
активизир_передн	Целочисленная константа типа LONG or ULONG, или задающая константу метка соответствия, содержащая в трех младших байтах (байты 0, 1, and 2), красную, зеленую и синюю компоненты, составляющие цвет; или метка соответствия для стандартного в Windows значения цвета. Этот параметр задает используемый по умолчанию цвет переднего плана для текста объекта, на который может переключаться фокус ввода.
активизир_фон	Целочисленная константа типа LONG или ULONG, или задающая константу метка соответствия, содержащая в трех младших байтах (байты 0, 1, and 2), красную, зеленую и синюю компоненты, составляющие цвет; или метка соответствия для стандартного в Windows значения цвета. Этот параметр задает используемый по умолчанию цвет фона для текста объекта, на который может переключаться фокус ввода.

Атрибут COLOR задает цвет фона в окне и используемые по умолчанию цвет переднего плана и фона для всех объектов в структуре TOOLBAR, которые не имеют атрибута COLOR.

Операторы EQUATE для стандартных в Windows цветов, содержатся в файле EQUATES.CLW. Для видеоконтроллера используемого при выполнении программы Windows автоматически находит наиболее подходящий заданному цвет. В Панели Управления Windows пользователь может изменить настройку стандартных цветов. При этом все объекты, для которых использовались стандартные для Windows цвета будут окрашены в новые цвета.

#### Пример:

```
WinOne WINDOW,AT(0,0,160,400)
    TOOLBAR,COLOR(00FF0000h,0000FF00h,000000FFh)
    !Синий фон, Зеленый передний план для активизируемых полей, и красный фон для
них
    END
END
```

**CURSOR (установить форму курсора мыши)****CURSOR(файл)**

**CURSOR**                      Задает форму, которую должен принимать курсор мыши, при попадании в область панели инструментов.

файл                              Строковая константа, содержащая имя файла с расширением .CUR или мнемоническое имя стандартной в Windows формы курсора.

Атрибут CURSOR задает форму, которую должен принимать курсор мыши, когда он располагается в границах панели инструментов. И эта форма курсора по умолчанию сохраняется для всех управляющих полей на панели, если для них явно не указана другая форма.

Стандартные для Windows формы курсора, мнемонические имена которых содержатся в файле EQUATES.CLW:

CURSQR:None	Нет курсора
CURSOR:Arrow	Обычный курсор в виде стрелки
CURSOR:IBeam	Курсор в виде заглавной буквы I похожий на двутавр
CURSOR:Wait	Песочные часы
CURSOR:Cross	Курсор в виде большого символа плюс
CURSOR:UpArrow	Курсор в виде стрелки направленной вверх
CURSOR:Size	Курсор в виде четырех стрелок, направленных в разные стороны.
CURSOR:Icon	Пиктограмма в рамке
CURSOR:SizeNWSE	Стрелки в направлении северо-запад - юго-восток
CURSOR:SizeNESW	Стрелки в направлении северо-восток - юго-запад
CURSOR:SizeWE	Стрелки в направлении запад - восток
CURSOR:SizeNS	Стрелки в направлении север - юг
CURSOR:DragWE	Стрелки в направлении запад - восток

**Пример:**

```

!Панель инструментов, на которой курсор имеет форму большого знака плюс
WinOne  WINDOW
        TOOLBAR,CURSOR('CURSOR:Cross')
        BUTTON('Cut'),USE(?CutButton),STD(STD:Cut)
        BUTTON('Copy'),USE(?CopyButton),STD(STD:Copy)
        BUTTON('Paste'),USE(?PasteButton),STD(STD:Paste)
        END
    END

```

**FONT (установить шрифт для панели инструментов)****FONT**([начертание][,размер][,цвет][,стиль])

<b>FONT</b>	Задаёт шрифт, используемый для панели инструментов по умолчанию.
начертание	Строчковая константа, содержащая название шрифта. Если этот параметр опущен, то используется системный шрифт.
размер	Целочисленная константа, содержащая размер шрифта в пунктах (1 пункт 1/72 дюйма). Если этот параметр опущен, то используется размер системного шрифта, используемого по умолчанию.
цвет	Целочисленная константа типа LONG, содержащая в младших трех байтах значения для красной, зеленой и голубой составляющих цвета или мнемоническое имя, задаваемое оператором EQUATE для стандартных в Windows значений, определяющих цвета.
стиль	Целочисленная константа или константное выражение или мнемоническая метка, задающая толщину и стиль букв шрифта.

В структуре TOOLBAR атрибут FONT задаёт шрифт для вывода содержимого всех управляющих полей на панели, которые не имеют своего атрибута FONT.

Параметр начертание может указывать любой шрифт зарегистрированный в системе Windows. В файле EQUATES.CLW содержатся значения для стандартных стилей. Значения в диапазоне от 0 до 1000 задают “яркость” шрифта. Это значение можно добавить к величине, означающей курсив, подчеркнутый или перечеркнутый шрифт. В файле EQUATES.CLW содержатся значения:

```

FONT:thin      EQUATE (100)
FONT:regular   EQUATE (400)
FONT:bold      EQUATE (700)
FONT:italic     EQUATE (01000H)
FONT:underline EQUATE (02000H)
FONT:strikeout EQUATE (04000H)

```

**Пример:**

```

Win1      WINDOW      !Панель инструментов, использующая шрифт
Times New Roman размером 14
      TOOLBAR,FONT('Times New Roman',14,00H)
      BUTTON('Cut'),USE(?CutButton),STD(STD:Cut)
      BUTTON('Paste'),USE(?PasteButton),STD(STD:Paste)
      END
      END
Win2      WINDOW      !Панель инструментов, использующая шрифт Times New Roman 14,

```

полужирный курсив

```
TOOLBAR, FONT('Times New Roman', 14, 00H, FONT:Italic+FONT:bold)
  BUTTON('Cut'), USE(?CutButton), STD(STD:Cut)
  BUTTON('Paste'), USE(?PasteButton), STD(STD:Paste)
END
END .
```

## **NOMERGE (установить возможность или невозможность слияния)**

### **NOMERGE**

Атрибут NOMERGE означает, что линейка меню или панель инструментов окна WINDOW не должны объединяться с глобальным меню или глобальной панелью инструментов соответственно.

В операторе MENUBAR структуры APPLICATION атрибут NOMERGE означает, что линейка меню локальная и должна высвечиваться, только когда нет открытых порожденных MDI-окон. Таким образом в этом случае глобального меню нет.

Без атрибута NOMERGE линейка меню или панель инструментов MDI-окна автоматически объединяется с глобальным меню или глобальной панелью инструментов, а затем высвечивается в окне APPLICATION. Если же атрибут NOMERGE указан, то меню или панель инструментов окна WINDOW замещают глобальное меню или глобальную панель инструментов. Во то время, когда на окно WINDOW переключен фокус, выводимое меню или панель инструментов является собственным меню или панелью инструментов окна. Однако и в этом случае они выводятся в окне APPLICATION.

Линейка спускающегося меню или инструментальная панель заданные для окна без атрибута MDI, никогда не объединяются с глобальными - они выводятся в самом окне.

### **Пример:**

!APPLICATION окно с локальными линейками меню и инструментов:

```
MainWin  APPLICATION('My Application'), SYSTEM, MAX, ICON('Mylcon.ICO'), STATUS
  MENUBAR, NOMERGE
    ITEM('E&xit'), USE(?MainExit)
  END
  TOOLBAR, NOMERGE
    BUTTON('Exit'), USE(?MainExitButton)
  END
END
```

!MDI-окно со своими собственными линейками меню и инструментов, замещающими соответствующие элементы !окна APPLICATION

```
MDIChild WINDOW('Dialog Window'), MDI, SYSTEM, MAX, STATUS
  MENUBAR, NOMERGE
```

```

MENU('Edit'),USE(?EditMenu)
  ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut)
  ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy)
  ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste)
END
EHD
TOOLBAR,NOMERGE
  BUTTON('Cut'),USE(?CutButton),STD(STD:Cut)
  BUTTON('Copy'),USE(?CopyButton),STD(STD:Copy)
  BUTTON('Paste'),USE(?PasteButton),STD(STD:Paste)
END
TEXT,HVSCROLL,USE(Pre:Field),MSG('Enter some text here')
  BUTTON('&OK'),USE(?Exit),DEFAULT
END

```

## Управляющие поля структуры **MENUBAR**

### **MENU** (объявить окно меню)

```

MENU(ТЕКСТ)[,USE()][,KEY()][,MSG()][,HLP()][,STD()][,RIGHT]
  [,FIRST][,DISABLE]
  LAST
END

```

<b>MENU</b>	Объявляет окно меню внутри структуры <b>MENUBAR</b>
текст	Строковая константа, содержащая выводимый для пункта меню текст.
<b>USE</b>	Метка соответствия поля, служащая для того, чтобы ссылаться на пункт меню в исполняемых операторах.
<b>KEY</b>	Задаёт целочисленную константу или мнемоническое имя кода клавиши, которая немедленно открывает окно меню.
<b>MSG</b>	Задаёт строковую константу, содержащую текст, который следует вывести в строке состояния когда окно меню раскрыто.
<b>HLP</b>	Задаёт строковую константу, содержащую идентификатор системы справки для данного меню.
<b>STD</b>	Целочисленная константа или мнемоническое имя, которая определяет для данного окна меню "стандартное поведение в системе Windows".
<b>RIGHT</b>	Указывает, что данный пункт на линейке меню выводится насколько это возможно правее.
<b>FIRST</b>	Задаёт, что при слиянии меню данный пункт выводится слева (или сверху).
<b>LAST</b>	Задаёт, что при слиянии меню данный пункт выводится справа (или

снизу).

## DISABLE

Указывает, что во время первого раскрытия окна APPLICATION или WINDOW данное меню выводится “затушеванным”.

Оператор MENU объявляет в структуре MENUBAR спускающееся или раскрывающееся рядом с пунктом окно меню. Когда выбран данный пункт меню в раскрывающемся окне выводятся вложенные в него меню и/или пункты, заданные операторами ITEM. Обычно окно меню раскрывается (спускается) непосредственно под соответствующим ему текстом на линейке меню (или над, если ниже нет места). Когда в окне меню клавишей ENTER или СТРЕЛКА ВПРАВО выбирается пункт, то справа от заданного параметром текст пункта (или слева, если справа нет места) раскрывается следующее окно меню (каскад меню). Клавиша СТРЕЛКА ВЛЕВО возвращает нас в предыдущее меню. Атрибутом KEY меню назначается специальная “горячая” клавиша. Чтобы окно меню немедленно спускалось, параметром атрибута KEY должен быть допустимый в Clarion код клавиши.

Строка текст может содержать амперсанд (&), назначающий в качестве горячей клавиши следующую за ним букву, которая при выводе пункта меню автоматически подчеркивается. Если это пункт меню на линейке меню, то нажатие клавиши Alt в сочетании с заданной амперсандом горячей клавишей выделит пункт и раскроет меню. Если это вложенное меню, то в раскрытом меню нажатие одной горячей клавиши вызовет выполнение пункта меню. Если в значении параметра текст отсутствует амперсанд, то в качестве горячей клавиши принимается первый отличный от пробела символ в этом тексте, но при выводе он не будет подчеркиваться. Для того, чтобы использовать амперсанд как часть текста, поместите в текстовую строку два символа амперсанда (&&), а отображаться будет только один.

## Пример:

!Главное окно прикладной программы с MDI. Окно содержит главное меню для прикладной программы

```
MainWin  APPLICATION('My Application'),SYSTEM,MAX,ICON('Mylcon.ICO'),STATUS |
           ,HVSCROLL,RESIZE
           MENUBAR
             MENU('File'),USE(?FileMenu),FIRST
               ITEM('Open...'),USE(?OpenFile)
               ITEM('Close'),USE(?CloseFile),DISABLE
               ITEM('E&xit'),USE(?MainExit)
             END
             MENU('Edit'),USE(?EditMenu),KEY(CtrlE),HLP('EditMenuHelp')
               ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo),DISABLE
               ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut),DISABLE
               ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy),DISABLE
               ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste),DISABLE
             END
             MENU('Window'),STD(STD:WindowList),MSG('Arrange or Select Window'),LAST
               ITEM('Tile'),STD(STD:TileWindow)
```



```

ITEM('Cascade'),STD(STD:CascadeWindow)
ITEM('Arrange Icons'),STD(STD:Arrangelcons)
END
MENU('Help'),USE(?HelpMenu),LAST,RIGHT
ITEM('Contents'),USE(?HelpContents),STD(STD:HelpIndex)
ITEM('Search for Help On...'),USE(?HelpSearch),STD(STD:HelpSearch)
ITEM('How to Use Help'),USE(?HelpOnHelp),STD(STD:HelpOnHelp)
ITEM('About MyApp...'),USE(?HelpAbout)
END
END
END

```

## ITEM (объявить пункт меню)

```

ITEM(текст)[,USE()][,KEY()][,MSG()][,HLP()][,STD()][,CHECK]
[,FIRST][,SEPARATOR][,DISABLE]
LAST

```

<b>ITEM</b>	Объявляет пункт меню внутри структуры MENUBAR или MENU
текст	Строковая константа, содержащая выводимый для пункта меню текст.
<b>USE</b>	Метка соответствия поля, служащая для того, чтобы можно было ссылаться на пункт меню в исполняемых операторах.
<b>KEY</b>	Задаёт целочисленную константу или мнемоническое имя кода клавиши, которая вызывает немедленное выполнение пункта меню.
<b>MSG</b>	Задаёт строковую константу, содержащую текст, который следует вывести в строке состояния когда пункт меню подсвечен.
<b>HLP</b>	Задаёт строковую константу, содержащую идентификатор системы справки для данного пункта меню.
<b>STD</b>	Целочисленная константа или мнемоническое имя, которая определяет для данного пункта меню “стандартное в системе Windows действие”.
<b>CHECK</b>	Указывает, что данный пункт является переключателем с двумя состояниями.
<b>DISABLE</b>	Указывает, что во время первого раскрытия окна APPLICATION или WINDOW данный пункт меню выводится “затусованным”.
<b>FIRST</b>	Задаёт, что при слиянии меню данный пункт выводится сверху.
<b>LAST</b>	Задаёт, что при слиянии меню данный пункт выводится снизу.
<b>SEPARATOR</b>	Указывает, что данный пункт во время выполнения программы представляет собой горизонтальную сплошную линию поперек окна меню, служащую разделителем групп пунктов меню.

Оператор ITEM объявляет внутри структуры MENUBAR или MENU пункт меню. Строка текст может содержать амперсанд (&), назначающий в качестве горячей клавиши следующую за ним букву, которая при выводе пункта меню автоматически подчеркивается.

Если это пункт на линейке меню, то нажатие клавиши Alt в сочетании с заданной амперсандом горячей клавишей выделит пункт и вызовет его выполнение. Если пункт находится во вложенном меню, то в раскрытом меню нажатие одной горячей клавиши вызовет выделение и выполнение этого пункта меню. Если в значении параметра текст отсутствует амперсанд, то в качестве горячей клавиши принимается первый отличный от пробела символ в этом тексте, но при выводе он не будет подчеркиваться. Для того, чтобы использовать амперсанд как часть текста, поместите в текстовую строку два символа амперсанда (&&), а отображаться будет только один. Для немедленного выполнения предусмотренных данным пунктом меню действий атрибутом KEY для поля назначается отдельная горячая клавиша. Это может быть любой допустимый в Clarion код клавиши или комбинации клавиш.

Конкретный пункт в структуре MENU выделяется прямоугольной полосой-курсором. Обычно, если не задан атрибут STD, каждый пункт меню связывается с некоторой подпрограммой, которая должна выполняться при выборе данного пункта. Атрибут STD определяет, что данный пункт меню выполняет стандартные в среде Windows действия, такие как расположение окон каскадом (пункт Cascade в меню WINDOW) или рядом друг с другом (пункт Tile). Атрибут SEPARATOR создает пункт, который служит только для того, чтобы разделять группы пунктов, поэтому он не должен иметь ни параметра текст, ни других атрибутов. С его помощью создается сплошная горизонтальная линия поперек окна меню.

Операторы ITEM, которые находятся вне структуры MENU, создают пункты, помещаемые на линейку меню. Они не связаны со окнами спускающихся меню. Для того, чтобы показать это пользователю, обычным правилом служит завершение текста для данного пункта меню восклицательным знаком (!). Например, для того чтобы известить пользователя об исполняемой сути данного пункта меню, параметр текст мог бы содержать 'Exit!'.

Генерируемые события:

EVENT:Accepted Пользователь нажал Enter или щелкнул мышью на активном объекте.

### Пример:

!Порождающее окно прикладной программы с MDI, содержащее главное меню программы:

```
MainWin  APPLICATION('My Application'),SYSTEM,MAX,ICON('Mylcon.ICO'),STATUS |
          ,HVSCROLL,RESIZE
          MENUBAR
            ITEM('E&xit!'),USE(?MainExit),FIRST
            MENU('File'),USE(?FileMenu),FIRST
              ITEM('Open...'),USE(?OpenFile),HLP('OpenFileHelp'),FIRST
              ITEM('Close'),USE(?CloseFile),HLP('CloseFileHelp'),DISABLE
              ITEM('Auto Increment'),USE(ToggleVar),CHECK
            END
            MENU('Edit'),USE(?EditMenu),KEY(CtrlE),HLP('EditMenuHelp')
```

```
ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo),DISABLE
ITEM,SEPARATOR
ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut),DISABLE
ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy),DISABLE
ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste),DISABLE
END
MENU('Window'),STD(STD:WindowList),MSG('Arrange or Select Window'),LAST
ITEM('Tile'),STD(STD:TileWindow)
ITEM('Cascade'),STD(STD:CascadeWindow)
ITEM('Arrange Icons'),STD(STD:Arrangelcons)
ITEM,SEPARATOR
END
MENU('Help'),USE(?HelpMenu),LAST,RIGHT
ITEM('Contents'),USE(?HelpContents),STD(STD:HelpIndex)
ITEM('Search for Help On...'),USE(?HelpSearch),STD(STD:HelpSearch)
ITEM('How to Use Help'),USE(?HelpOnHelp),STD(STD:HelpOnHelp)
ITEM('About MyApp...'),USE(?HelpAbout),MSG('Copyright Info'),LAST
END
END
END
```

## **Атрибуты объектов в меню**

### **CHECK (объявить пункт меню - переключатель)**

#### **CHECK**

Атрибут CHECK объявляет, что пункт меню может иметь или значение “включено”, или “выключено”. Когда пункт меню включен, слева от него выводится отметка о включении, а в USE-переменную заносится единица. Когда пункт выключен, отметка о включении слева от него не выводится, а в USE-переменную заносится ноль.

### **DISABLE (установить, что при открытии окна объект затушеван)**

#### **DISABLE**

Атрибут DISABLE указывает, что, когда окно WINDOW или APPLICATION раскрывается экранный объект неактивен и отображается затушеванным. Неактивный объект можно активизировать оператором ENABLE.

**FIRST, LAST (установить положение окна-меню или пункта)****FIRST  
LAST**

Атрибуты FIRST и LAST задают положение пункта меню на линейке глобального меню, когда линейка окна объединяется с глобальным меню. Шкала приоритетов:

1. Пункты глобального меню с атрибутом FIRST.
2. Пункты локального меню с атрибутом FIRST.
3. Пункты глобального меню без атрибутов FIRST или LAST.
4. Пункты локального меню без атрибутов FIRST или LAST.
5. Пункты глобального меню с атрибутом LAST.
6. Пункты локального меню с атрибутом LAST.

**HLP (установить идентификатор справочной системы)****HLP(идентификатор)**

HLP	Задаёт идентификатор справочной системы
идентификатор	Строковая константа задающая ключ, который используется для доступа к в справочную систему. Это может быть или ключевое слово справочной системы или “строка контекста”.

Атрибут HLP задаёт идентификатор справочной системы для экранного объекта. Как только пользователь нажимает клавишу F1, Windows автоматически высвечивает справку, если та существует. Если пользователь нажимает F1, запрашивая справку, когда на поле переключен фокус, библиотечная процедура использует идентификатор, заданный для этого поля, для поиска файла справочной системы раздела с заданным идентификатором.

Идентификатор может содержать ключевое слово справочной системы или “строку контекста”. Ключевое слово справочной системы представляет собой слово или фразу, которая высвечивается в окне Help Search справочной системы. Если при нажатии пользователем клавиши F1, это ключевое слово определяет только один раздел справки, то файл раскрывается на этом разделе. Если ключевое слово определяет несколько разделов справки, то пользователю раскрывается окно поиска.

“Строка контекста” отличается в идентификаторе знаком тильды (~) спереди, за которым следует уникальный идентификатор (без пробелов), который связан точно с одним справочным разделом. При нажатии пользователем клавиши F1, файл справки раскрывается на конкретном разделе, связанном с данным контекстом. Если тильда отсутствует, то подразумевается, что идентификатор содержит ключевое слово справочной системы.

**Пример:**

```

MainWin APPLICATION('My Application'),SYSTEM,MAX,ICON('Mylcon.ICO'),STATUS
  MENUBAR
  MENU('&File'),USE(?FileMenu)
    ITEM('&Open...'),USE(?OpenFile),HLP('~OpenFileHelp')
    ITEM('&Close'),USE(?CloseFile),DISABLE,HLP('Close open file')
    ITEM(),SEPARATOR
    ITEM('E&xit'),USE(?MainExit),MSG('Exit the program')
  END
  MENU('&Help'),USE(?HelpMenu),RIGHT
    ITEM('&Contents'),USE(?HelpContents),STD(STD:HelpIndex)
    ITEM('&Search...'),USE(?HelpSearch),STD(STD:HelpSearch)
    ITEM('&How to Use Help'),USE(?HelpOnHelp),STD(STD:HelpOnHelp)
    ITEM('&About MyApp...'),USE(?HelpAbout)
  END
END
END

```

**KEY (клавиша быстрого выполнения меню или пункта)****KEY (код клавиши)**

KEY	Задаёт горячую клавишу.
код клавиши	Код клавиши принятый в Clarion, или метка соответствия кода клавиши.

Атрибут KEY задаёт горячую клавишу, предназначенную для того, чтобы немедленно переключит фокус и выполнить связанное с пунктом или меню действие.

**Пример:**

```

MainWin APPLICATION('My Application'),SYSTEM,MAX,ICON('Mylcon.ICO'),STATUS
  MENUBAR
  MENU('&Edit'),USE(?EditMenu)
    ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut),DISABLE
    ITEM('&Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy),DISABLE
    ITEM('&Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste),DISABLE
  END
  MENU('&Window'),STD(STD:WindowList),LAST
    ITEM('&Tile'),STD(STD:TileWindow)
    ITEM('&Cascade'),STD(STD:CascadeWindow)
    ITEM('&Arrange Icons'),STD(STD:ArrangeIcons)
  END
END
END

```

## MSG (установить сообщение для меню или пункта)

MSG(текст)

**MSG**                                Задаёт текст, который должен выводиться в строке сообщений.  
 текст                                Строковая константа, содержащая сообщение, которое должно выводиться в строке состояния.

Атрибут MSG указывает текст, подлежащий выводу в первой области строки состояния, когда на поле переключен фокус ввода.

### Пример:

```
MainWin APPLICATION('My Application'),SYSTEM,MAX,ICON('Mylcon.ICO'),STATUS
  MENUBAR
    MENU('&File'),USE(?FileMenu)
      ITEM('&Open...'),USE(?OpenFile),MSG('Open a file')
      ITEM('&Close'),USE(?CloseFile),DISABLE,MSG('Close the open file')
      ITEM(),SEPARATOR
      ITEM('E&xit'),USE(?MainExit),MSG('Exit the program')
    END
    MENU('&Edit'),USE(?EditMenu)
      ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut),DISABLE
      ITEM('&Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy),DISABLE
      ITEM('&Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste),DISABLE
    END
    MENU('&Window'),STD(STD:WindowList),LAST
      ITEM('&Tile'),STD(STD:TileWindow)
        ITEM('&Cascade'),STD(STD:CascadeWindow)
      ITEM('&Arrange Icons'),STD(STD:Arrangelcons)
    END
    MENU('&Help'),USE(?HelpMenu),RIGHT
      ITEM('&Contents'),USE(?HelpContents),STD(STD:HelpIndex)
      ITEM('&Search...'),USE(?HelpSearch),STD(STD:HelpSearch)
      ITEM('&How to Use Help'),USE(?HelpOnHelp),STD(STD:HelpOnHelp)
      ITEM('&About MyApp...'),USE(?HelpAbout)
    END
  END
END
END
```

**RIGHT (установить положение меню)****RIGHT**

Атрибут RIGHT указывает, что меню помещается на правый край линейки меню.

**SEPARATOR (установить в меню разделительную линию)****SEPARATOR**

Атрибут SEPARATOR указывает, что пункт меню является разделительной линией, которая служит для группировки пунктов внутри меню. Для данного пункта не может задаваться никаких других атрибутов.

**STD (установить стандартное действие для меню или пункта)****STD (действие)**

<b>STD</b>	Задаёт стандартное в Windows действие.
действие	Целочисленная константа или мнемоническое имя, указывающее идентификатор стандартного действия в Windows.

Атрибут STD указывает, что поле выполняет некоторое стандартное в Windows действие. Это действие автоматически выполняется библиотечной процедурой и для этого поля не генерируется никаких событий.

Операторы EQUATE для мнемонических имен, обозначающих стандартные в Windows действия, содержатся в файле EQUATES.CLW. Пример этих операторов (полный список см. в файле EQUATES.CLW) :

STD:WindowList	Список открытых MDI окон
STD:TileWindow	Расположить окна рядом
STD:CascadeWindow	Расположить окна каскадом
STD:ArrangeIcons	Упорядочить пиктограммы
STD:HelpIndex	Оглавление справочной системы
STD:HelpSearch	Окно поиска в справочной системе

**Пример:**

```
MDIChild WINDOW('Child One'),MDI,SYSTEM,MAX
  MENUBAR
  MENU('Edit'),USE(?EditMenu)
  ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo)
  ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut)
  ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy)
```

```

    ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste)
  END
END
TOOLBAR
  BUTTON('Cut'),USE(?CutButton),ICON(ICON:Cut),STD(STD:Cut)
  BUTTON('Copy'),USE(?CopyButton),ICON(ICON:Copy),STD(STD:Copy)
  BUTTON('Paste'),USE(?PasteButton),ICON(ICON:Paste),STD(STD:Paste)
END
END

```

## USE (задать для меню или пункта метку соответствия или переменную)

```

USE( [ метка | [,номер] [,метка соответствия] ]
    | переменная | )

```

<b>USE</b>	Задаёт переменную или метку соответствия поля.
метка	Мнемоническая метка соответствия, предназначенная для того, чтобы ссылаться на это поле в исполняемых операторах.
переменная	Переменная, принимающая введенное в поле значение. Если не задан параметр метка соответствия, то метка этой переменной (со знаком вопроса спереди) становится меткой соответствия для экранного объекта.
номер	Целочисленная константа, которая указывает номер присваиваемый компилятором метке соответствия для этого поля.
метка соответствия	Мнемоническая метка соответствия объекта, которая служит для того, чтобы в исполняемых операторах ссылаться на данный объект, когда имя переменной уже использовалось в этой экранной структуре. Дополнительная метка соответствия обеспечивает уникальность меток соответствия полей, тогда как использование USE-переменных этого не обеспечивает.

Атрибут USE задает для экранного поля переменную или метку соответствия. Атрибут с параметром метка просто обеспечивает способ, с помощью которого на это поле можно ссылаться в исполняемых операторах. Для некоторых полей в качестве параметра атрибута USE допускается только метка соответствия, но не имя переменной. Это такие поля как: PROMPT, IMAGE, LINE, BOX, ELLIPSE, GROUP, RADIO, REGION, MENU, и BUTTON. Атрибут USE обеспечивает для поля переменную, в которую заносятся введенные пользователем в поле данные. Это относится к следующим полям: ITEM с атрибутом CHECK, и полям ENTRY, OPTION, SPIN, TEXT, LIST, COMBO, CHECK, и CUSTOM.

Всем полям в окне APPLICATION или WINDOW компилятором автоматически присваиваются номера. Для полей на линейке меню окна APPLICATION эти номера начинаются с минус единицы (-1) и уменьшаются на единицу для каждого поля MENU или ITEM на линейке. В окне WINDOW нумерация начинается с 1 и увеличивается с шагом 1 для каждого следующего поля.



Параметр номер атрибута USE позволяет задать номер, назначаемый компилятором экранному объекту. Это число используется также в качестве начальной точки отсчета в дальнейшей нумерации полей, не имеющих в атрибуте USE параметра номер. Последующие поля не имеющие в атрибуте USE параметра номер нумеруются с приращением (или уменьшением) относительно последнего назначенного номера.

Для двух или более полей с одинаковым значением атрибута USE в одном окне APPLICATION или WINDOW создадутся одинаковые метки соответствия полей; когда компилятор обнаруживает такую ситуацию, он удаляет для всех этих полей метки соответствия. Предупреждая возникновение неоднозначности относительно того, на какое поле в действительности делается ссылка, компилятор делает невозможными ссылки на эти поля в исполняемых операторах. Однако такой подход позволяет сознательно создавать такую ситуацию для того, чтобы вывести содержимое одной переменной в нескольких полях с разными шаблонами.

### Пример:

```
MainWin APPLICATION('My Application'),SYSTEM,MAX,ICON('Mylcon.ICO'),STATUS
    MENUBAR
        MENU(' &File'),USE(?FileMenu)
            ITEM(' &Open... '),USE(?OpenFile)
            ITEM(' &Close'),USE(?CloseFile),DISABLE
            ITEM('E&xit'),USE(?MainExit)
        END
        MENU(' &Edit'),USE(?EditMenu)
            ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut),DISABLE
            ITEM(' &Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy),DISABLE
            ITEM(' &Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste),DISABLE
        END
        MENU(' &Window'),STD(STD:WindowList),LAST
            ITEM(' &Tile'),STD(STD:TileWindow)
            ITEM(' &Cascade'),STD(STD:CascadeWindow)
            ITEM(' &Arrange Icons'),STD(STD:Arrangelcons)
        END
        MENU(' &Help'),USE(?HelpMenu)
            ITEM(' &Contents'),USE(?HelpContents),STD(STD:HelpIndex)
            ITEM(' &Search... '),USE(?HelpSearch),STD(STD:HelpSearch)
            ITEM(' &How to Use Help'),USE(?HelpOnHelp),STD(STD:HelpOnHelp)
            ITEM(' &About MyApp... '),USE(?HelpAbout)
        END
    END
END
```



## Глава 7 - Управляющие объекты в окнах

### Объекты структур **TOOLBAR** и **WINDOW**

#### **BOX** (объявить прямоугольную область внутри окна)

```
BOX,AT()[,USE()[,DISABLE][,COLOR()[,FILL()[,ROUND][,FULL]
[,SCROLL][HIDE][,LINEWIDTH( )]
```

<b>BOX</b>	Размещает в окне прямоугольную область.
<b>AT</b>	Задаёт первоначальные размер и расположение поля. Если атрибут опущен, то значения по умолчанию выбираются библиотечной подпрограммой во время выполнения.
<b>USE</b>	Метка соответствия поля, служащая для того, чтобы осуществить ссылку на это поле в исполняемых операторах.
<b>DISABLE</b>	Указывает, что когда в окно <b>WINDOW</b> (или <b>APPLICATION</b> ) раскрывается впервые, данное поле выглядит затухеванным.
<b>COLOR</b>	Задаёт цвет рамки для поля. Если этот атрибут опущен, то рамки нет.
<b>FILL</b>	Указывает цвет внутренней области поля. По умолчанию внутренняя область поля не окрашена.
<b>ROUND</b>	Задаёт, что углы прямоугольника скругленные. По умолчанию углы прямые.
<b>FULL</b>	Указывает, что поле занимает всю протяженность окна по любому из опущенных в атрибуте <b>AT</b> измерений - длине или высоте.
<b>SCROLL</b>	Указывает, что поле подвергается скроллингу вместе с окном.
<b>HIDE</b>	Задаёт, что когда окно <b>WINDOW</b> или <b>APPLICATION</b> раскрывается в первый раз, данное поле не прорисовывается. Для того, чтобы отобразить это поле нужно использовать оператор <b>UNHIDE</b> .
<b>LINEWIDTH</b>	Указывает толщину рамки прямоугольника.

Управляющее поле **BOX** представляет в структуре **WINDOW** или **TOOLBAR** прямоугольную область в заданном атрибуте **AT** месте и заданного размера. На это поле не может переключаться фокус ввода и оно не может генерировать события.

#### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
BOX,AT(0,0,20,20) !Незакрашенный, с черной рамкой
BOX,AT(0,20,20,20),USE(?Box1),DISABLE
!Незакрашенный, с черной рамкой, затухеванный
BOX,AT(20,20,20,20),ROUND
!Незакрашенный, с черной рамкой и скругленными углами
```

```

BOX,AT(40,40,20,20),FILL(COLOR:ACTIVEBORDER)
!Закрашенный, с черной рамкой
BOX,AT(60,60,20,20),COLOR(COLOR:ACTIVEBORDER)
!Незакрашенный, с рамкой цвета рамки активного окна
BOX,AT(480,180,20,20),SCROLL !Подлежащий скроллингу
END

```

## BUTTON (объявить кнопку)

```

BUTTON(текст),AT(),CURSOR(),USE(),DISABLE(),KEY(),MSG()
[,HLP][,SKIP][,STD][,FONT][,ICON(),DEFAULT][,IMM][,REQ]
[,FULL][,SCROLL][,ALRT()][,HIDE][,DROPID] [,TIP( )] [,LEFT]
|LEFT|

```

<b>BUTTON</b> текст	Помещает командную кнопку в окне WINDOW или на панели инструментов. Строковая константа, содержащая надпись, которая должна выводиться на кнопке (наряду с заданной пиктограммой) Для того, чтобы указать “горячую букву” (клавишу ускоренного доступа) для кнопки, константа может содержать символ амперсанд (&).
<b>AT</b>	Задаёт первоначальные размеры и расположение кнопки. Если атрибут опущен, то значения по умолчанию устанавливаются библиотечной подпрограммой во время выполнения.
<b>CURSOR</b>	Задаёт форму, которую должен принимать курсор мыши, при попадании на кнопку. Если этот атрибут кнопки опущен, то используется значение атрибут CURSOR структуры WINDOW, если же и тот не указан, то курсор имеет форму, используемую в среде Windows по умолчанию.
<b>USE</b>	Метка соответствия поля, служащая для того, чтобы осуществить ссылку на это поле в исполняемых операторах.
<b>DISABLE</b>	Указывает, что когда в окно WINDOW (или APPLICATION) раскрывается впервые, данная кнопка выглядит затупеванной.
<b>KEY</b>	Задаёт целочисленную константу или мнемоническое имя кода клавиши, которая вызывает немедленное переключение фокуса на данную кнопку..
<b>MSG</b>	Задаёт строковую константу, содержащую текст, который следует вывести в строке состояния, когда на данную кнопку переключен фокус.
<b>HLP</b>	Задаёт строковую константу, содержащую идентификатор системы справки для данной кнопки.
<b>SKIP</b>	Указывает, что на кнопку не переключается фокус ввода, и доступ к ней может быть осуществлен только посредством мыши или клавиши ускоренного доступа.
<b>STD</b>	Целочисленная константа или мнемоническое имя, которое определяет для данной кнопки “стандартное в системе Windows действие”.
<b>FONT</b>	Задаёт шрифт, используемый для данной кнопки.
<b>ICON</b>	Задаёт имя файла с расширением ICO или имя стандартной пиктограммы,

	которая должна высвечиваться на поверхности данной кнопки.
<b>DEFAULT</b>	Указывает, что при нажатии пользователем клавиши ENTER кнопка автоматически “нажимается”.
<b>IMM</b>	Указывает, что для данного поля непрерывно генерируется событие, начиная с момента нажатия левой кнопки мыши и до момента ее отпускания. Если же этот атрибут опущен, то события генерируются только в момент нажатия и отпускания левой кнопки мыши.
<b>REQ</b>	Задаст, что при нажатии данной кнопки, библиотечная процедура автоматически проверяет все поля для ввода данных с атрибутом REQ в этом окне на предмет наличия в них данных отличных от пробелов и нулей.
<b>FULL</b>	Указывает, что поле занимает всю протяженность окна по любому из опущенных в атрибуте AT измерений - длине или высоте.
<b>SCROLL</b>	Указывает, что поле подвергается скроллингу вместе с окном.
<b>ALRT</b>	Задаст активные горячие клавиши для данного поля.
<b>HIDE</b>	Задаст, что когда окно WINDOW или APPLICATION раскрывается в первый раз, данное поле не прорисовывается. Для того, чтобы отобразить это поле нужно использовать оператор UNHIDE.
<b>DROPID</b>	Указывает, что данное управляющее поле может служить областью в которой происходит вторая часть операции “потащить и отпустить” - отпускание объекта.
<b>TIP</b>	Задаст текст, который выводится как “возникающая” подсказка, когда курсор мыши задерживается на экранном объекте.
<b>LEFT</b>	Указывает, что пиктограмма выводится слева от текста.
<b>RIGHT</b>	Указывает, что пиктограмма выводится справа от текста.

Поле BUTTON представляет собой прямоугольную кнопку в окне WINDOW (или на панели инструментов, местоположение которой и размеры задаются атрибутом AT).

Кнопка с атрибутом IMM генерирует события с того, момента как только пользователь, указав на кнопку курсором, нажимает левую кнопку мыши и генерация продолжается до тех пор, пока кнопка не будет отпущена. Это позволяет соответствующие кнопке BUTTON операторы по обработке события выполнять непрерывно до тех пор, пока кнопка мыши не будет отпущена. Для кнопки без атрибута IMM события генерируются только в момент нажатия и отпускания левой кнопки мыши.

Кнопка BUTTON с атрибутом REQ является кнопкой “проверки обязательных полей”. Заполнение полей типа TEXT и ENTRY с атрибутом REQ не проверяется до тех пор, пока не будет нажата кнопка BUTTON с атрибутом REQ или не будет выполнена функция IN-COMplete. Фокус ввода переключается на первое незаполненное поле из числа обязательно заполняемых.

На поверхности кнопки с атрибутом ICON в дополнение к значению параметра текст выводится пиктограмма (по умолчанию текст выводится под ней). Кроме того, значение

параметра текст служит для определения клавиши ускоренного доступа.

### Генерируемые события:

EVENT:Selected	На управляющее поле переключен фокус ввода
EVENT:Accepted	Пользователь нажал на кнопку
EVENT:PreAlertKey	Пользователь нажал горячую клавишу определенную атрибутом ALRT
EVENT:AlertKey	Пользователь нажал горячую клавишу определенную атрибутом ALRT.
EVENT:Drop	Успешная операция “потящить и отпустить” в данное поле.

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
  BUTTON('1'),AT(0,0,20,20),USE(?B1)
  BUTTON('2'),AT(20,0,20,20),USE(?B2),KEY(F10Key)
  BUTTON('3'),AT(40,0,20,20),USE(?B3),MSG('Button 3')
  BUTTON('4'),AT(60,0,20,20),USE(?B4),HLP('Button4Help')
  BUTTON('5'),AT(80,0,20,20),USE(?B5),STD(STD:Cut)
  BUTTON('6'),AT(100,0,20,20),USE(?B6),FONT('Arial',12)
  BUTTON('7'),AT(120,0,20,20),USE(?B7),ICON(ICON:Question)
  BUTTON('8'),AT(140,0,20,20),USE(?B8),DEFAULT
  BUTTON('9'),AT(160,0,20,20),USE(?B9),IMM
  BUTTON('10'),AT(180,0,20,20),USE(?B10),CURSOR(CURSOR:Wait)
  BUTTON('11'),AT(200,0,20,20),USE(?B11),REQ
  BUTTON('12'),AT(220,0,20,20),USE(?B12),ALRT(F10Key)
  BUTTON('13'),AT(240,0,20,20),USE(?B13),SCROLL
END
CODE
OPEN(MDIChild)
ACCEPT
  CASE ACCEPTED()
  OF ?B1
    !Выполнить некоторые действия
  END
END
```

**CHECK (объявить кнопку с независимой фиксацией)**

```
CHECK(текст) ,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,KEY( )]
[,MSG( )][,HLP( )] [,SKIP] [,FONT( )] [,ICON( )] [,FULL]
[,SCROLL] [,ALRT( )] [,HIDE] [DROPID( )] TIP( )
[, LEFT] [,VALUE( )] [,TRN] [,COLOR( )] RIGHT
```

<b>CHECK</b>	Помещает кнопку с независимой фиксацией в окно WINDOW или на панель инструментов.
текст	Строковая константа, содержащая надпись, которая должна выводиться для данной кнопки. Для того, чтобы указать “горячую букву” (клавишу ускоренного доступа) для кнопки, константа может содержать символ амперсанд (&).
<b>AT</b>	Задаёт первоначальные размеры и расположение кнопки. Если атрибут опущен, то значения по умолчанию выбираются библиотечной подпрограммой во время выполнения.
<b>CURSOR</b>	Задаёт форму, которую должен принимать курсор мыши, при попадании на кнопку. Если этот атрибут кнопки опущен, то используется значение атрибут CURSOR структуры WINDOW, если же и тот не указан, то курсор имеет форму, используемую в среде Windows по умолчанию.
<b>USE</b>	Метка переменной, которая должна принимать значения, соответствующие состоянию кнопки. Если не указан атрибут VALUE, то ноль (0) означает выключено, а единица (1) - включено.
<b>DISABLE</b>	Указывает, что когда в окно WINDOW (или APPLICATION) раскрывается впервые, данная кнопка выглядит затусневанной.
<b>KEY</b>	Задаёт целочисленную константу или мнемоническое имя кода клавиши, которая вызывает немедленное переключение фокуса на данную кнопку и переключение ее состояния.
<b>MSG</b>	Задаёт строковую константу, содержащую текст, который следует вывести в строке состояния, когда на данную кнопку переключен фокус.
<b>HLP</b>	Задаёт строковую константу, содержащую идентификатор системы справки для данной кнопки.
<b>SKIP</b>	Указывает, что на кнопку не переключается фокус ввода, и доступ к ней может быть осуществлен только посредством мыши или клавиши ускоренного доступа.
<b>FONT</b>	Задаёт шрифт, используемый для данной кнопки.
<b>ICON</b>	Задаёт имя файла с расширением ICO или имя стандартной пиктограммы, которая должна высвечиваться на поверхности включенной кнопки.
<b>FULL</b>	Указывает, что поле занимает всю протяженность окна по любому из опущенных в атрибуте AT измерений - длине или высоте.
<b>SCROLL</b>	Указывает, что поле подвергается скроллингу вместе с окном.
<b>ALRT</b>	Задаёт активные горячие клавиши для данного поля.
<b>HIDE</b>	Задаёт, что когда окно WINDOW или APPLICATION раскрывается в первый

	раз, данное поле не прорисовывается. Для того, чтобы отобразить это поле нужно использовать оператор UNHIDE.
<b>DROPID</b>	Указывает, что данное управляющее поле может служить областью в которой происходит вторая часть операции “потащить и отпустить” - отпускание объекта.
<b>TIP</b>	Задаёт текст, который выводится как “возникающая” подсказка, когда курсор мыши задерживается на экранном объекте.
<b>LEFT</b>	Указывает, что надпись выводится слева от кнопки с независимой фиксацией.
<b>RIGHT</b>	Указывает, что надпись выводится справа от кнопки с независимой фиксацией (положение по умолчанию).
<b>TRN</b>	Указывает, что объект выводится на “прозрачном” фоне.
<b>COLOR</b>	Указывает цвет фона для текста объекта.

С помощью объекта CHECK в окно или панель инструментов в позицию, заданную атрибутом AT, помещается кнопка-переключатель. Когда кнопка выключена, USE-переменная принимает значение ноль(0); когда включена то USE-переменная принимает значение единица (1).

Кнопка CHECK с атрибутом ICON выглядит как фиксирующаяся кнопка с нарисованной на ней пиктограммой. Когда кнопка отжата, то переключатель выключен, а когда кнопка нажата, то переключатель включен.

Для автоматической установки USE-переменной к отличному от нуля значению и нулевого значения можно использовать свойства PROP:TrueValue and PROP:FalseValue

### Генерируемые события:

EVENT:Selected	На кнопку переключен фокус
EVENT:Accepted	Пользователь переключил состояние кнопки
EVENT:PreAlertKey	Пользователь нажал заданную атрибутом ALRT горячую клавишу
EVENT:AlertKey	Пользователь нажал заданную атрибутом ALRT горячую клавишу
EVENT:Drop	Успешная операция переноса объекта в данную кнопку

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
CHECK('1'),AT(0,0,20,20),USE(C1)
CHECK('2'),AT(0,20,20,20),USE(C2),VALUE('T','F')
END
CODE
OPEN(MDIChild)
ACCEPT
CASE ACCEPTED()
```



```

OF ?C1
  IF C1 = 1 THEN DO C1Routine.
OF ?C2
  IF C2 = 'T' THEN DO C2Routine.
END
END

```

## COMBO (объявить комбинированное окно списка)

```

COMBO(шаблон) ,FROM( ) ,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE]
[,KEY( )] [,MSG( )] [,HLP( )] [,SKIP][,FONT( )][,FORMAT( )]
[,DROP] [,COLUMN] [,VCR][,FULL][,GRID( )][,SCROLL] [,ALRT( )]
[,HIDE] [,READONLY] [,REQ] [,NOBAR][DROPID( )][,TIP( )]
[,TRN][,COLOR( )] [,MARK( )] [,HSCROLL][,LEFT][,
|INS|[, |UPR| | |
|IMM      |VSCROLL |RIGHT      |OVR |CAP|
|          |HVSCROLL|CENTER      |
|          |DECIMAL  |

```

<b>COMBO</b>	Помещает в окно или панель инструментов поле для ввода данных вместе со связанным с ним окном списком элементов данных.
шаблон	Шаблон для отображения данных, который указывает формат вводимых в это поле данных.
<b>FROM</b>	Задаёт источник для выводимых в списке элементов данных.
<b>AT</b>	Задаёт первоначальные размеры и расположение комбинированного окна списка.
<b>CURSOR</b>	Задаёт форму, которую должен принимать курсор мыши, при попадании в это поле. Если этот атрибут опущен, то используется значение атрибута CURSOR структуры WINDOW, если же и тот не указан, то курсор имеет форму, используемую в среде Windows по умолчанию.
<b>USE</b>	Метка соответствия поля, которая служит для ссылок на данное поле в исполняемых операторах, или метка переменной, которая получает значение выбранного или введенного элемента данных.
<b>DISABLE</b>	Указывает, что когда в окно WINDOW (или APPLICATION) раскрывается впервые, данное поле выглядит затухеванным.
<b>KEY</b>	Задаёт целочисленную константу или мнемонический код клавиши (или комбинации клавиш), с помощью которой фокус немедленно переключается на это комбинированное окно списка.
<b>MSG</b>	Задаёт строковую константу, содержащую текст, который следует вывести в строке состояния, когда на данное поле переключен фокус.
<b>HLP</b>	Задаёт строковую константу, содержащую идентификатор системы справки для данного поля.
<b>SKIP</b>	Указывает, что для ввода данных в это поле фокус ввода переключается только посредством мыши или клавиши ускоренного доступа, и не

	задерживается на этом поле после завершения ввода данных.
<b>FONT</b>	Задает шрифт для вывода текста в комбинированном окне списка.
<b>FORMAT</b>	Задает формат для отображения данных в окне списка.
<b>DROP</b>	Задает выпадающее окно списка и число элементов данных, которое содержит выпадающая часть.
<b>COLUMN</b>	В многоколоночном окне списка задает длину выделенной полосы-курсора.
<b>VCR</b>	При наличии линейки горизонтального скроллинга задает наличие слева от линейки кнопок, подобных клавишам управления на видеомагнитофоне.
<b>FULL</b>	Указывает, что по любому из опущенных параметров атрибута AT (ширина или высота) окно расширяется таким образом, чтобы занимать всю протяженность окна.
<b>GRID</b>	Задает цвет линий сетки между колонками в списке.
<b>SCROLL</b>	Указывает, что поле подвергается скроллингу вместе с содержимым окна.
<b>ALRT</b>	Задает активные горячие клавиши для данного поля.
<b>HIDE</b>	Задает, что когда окно WINDOW или APPLICATION раскрывается в первый раз, данное поле не прорисовывается. Для того, чтобы отобразить это поле нужно использовать оператор UNHIDE.
<b>READONLY</b>	Указывает, что в поле нельзя ввести произвольные данные, а можно только выбрать значение из окна списка.
<b>NOBAR</b>	Задает, что выделенная полоса курсор отображается, только когда фокус переключается на окно списка.
<b>DROPID</b>	Указывает, что данное управляющее поле может служить областью в которой происходит вторая часть операции “потящить и отпустить” - отпускание объекта.
<b>TIP</b>	Задает текст, который выводится как “возникающая” подсказка, когда курсор мыши задерживается на экранном объекте.
<b>TRN</b>	Указывает, что объект выводится на “прозрачном” фоне.
<b>COLOR</b>	Задает цвет фона и цвета активизированного объекта.
<b>REQ</b>	Указывает, что данное поле не может остаться пустым (или равным нулю) после завершения обработки окна.
<b>MARK</b>	Задает режим выбора нескольких элементов одновременно.
<b>IMM</b>	Задает генерацию события при нажатии пользователем любой клавиши
<b>HSCROLL</b>	Указывает, что когда какая-либо часть данных по горизонтали не помещается в окне списка, к нему автоматически присоединяется линейка горизонтального скроллинга.
<b>VSCROLL</b>	Указывает, что когда какая-либо часть данных по вертикали не помещается в окне списка, к нему автоматически присоединяется линейка вертикального скроллинга.
<b>HVSCROLL</b>	Указывает, что когда какая-либо часть данных не помещается в окне списка, к нему автоматически присоединяются линейки и вертикального и горизонтального скроллинга.
<b>LEFT</b>	Задает, что данные в окне списка выравниваются влево.
<b>RIGHT</b>	Задает, что данные в окне списка выравниваются право.

<b>CENTER</b>	Задаёт, что данные в окне списка выравниваются по центру окна.
<b>DECIMAL</b>	Задаёт, что данные в окне списка выравниваются по десятичной точке.
<b>INS/OVR</b>	Задаёт при вводе данных режим вставки или замещения (допустимо только в окнах, имеющих атрибут MASK).
<b>UPR/CAP</b>	Указывает, что данные вводятся прописными буквами или по типу имен собственных (Первая Буква В Каждом Слове Прописная).

С помощью поля COMBO в заданном атрибуте AT месте окна (или панели инструментов) помещается поле для ввода и связанный с ним список элементов данных (комбинация полей типа ENTRY и LIST - комбинированное окно списка). Пользователь может набрать данные на клавиатуре или выбрать элемент из списка. Введенные с клавиатуры данные автоматически не проверяются на соответствие одному из элементов списка. Поле ввода данных комбинированного окна действует как “инкрементальный локатор” в окне списка - по мере ввода символов пользователем выделенная полоса-курсор в окне устанавливается на наиболее соответствующий элемент списка.

В поле COMBO с атрибутом DROP до тех пор, пока на поле не переключится фокус и пользователь не нажмет клавишу стрелка вниз или не щелкнет на пиктограмме справа от поля, отображается только выбранный в данный момент элемент. А как только пользователь нажмет клавишу стрелка вниз или щелкнет на пиктограмме, появится список возможных вариантов (“выпадет”), позволяя пользователю выбрать элемент.

Поле COMBO с атрибутом IMM генерирует события каждый раз, когда пользователь перемещает выделенную полосу-курсор на другой элемент списка, или нажимает клавишу, вызывающую скроллинг высвеченных элементов списка. Тем самым программе предоставляется удобная возможность повторно заполнить высвечиваемую очередь или получить выделенную в данный момент запись для того, чтобы вывести на экран другие поля этой записи. У COMBO-поля с атрибутом VCR имеются слева от линейки горизонтального скроллинга (если она есть) кнопки управления скроллингом, подобные кнопкам управления видеомagnитофоном. Эти кнопки позволяют пользователю использовать мышь для прокрутки списка.

### Генерируемые события:

EVENT:Selected	На кнопку переключен фокус
EVENT:Accepted	Пользователь выбрал элемент списка
EVENT:Rejected	Пользователь ввел несоответствующее шаблону значение.
EVENT:NewSelection	Выбран другой элемент списка (полоса-курсор переместилась вниз или вверх)
EVENT:PreAlertKey	Пользователь нажал заданную атрибутом ALRT горячую клавишу
EVENT:AlertKey	Пользователь нажал заданную атрибутом ALRT горячую клавишу
EVENT:Drop	Успешная операция переноса объекта в данное поле.

EVENT:ScrollUp	Пользователь нажал стрелку вверх (только для окон с атрибутом IMM).
EVENT:ScrollDown	Пользователь нажал стрелку вниз (только для окон с атрибутом IMM).
EVENT:PageUp	Пользователь нажал клавишу PgUp (только для окон с атрибутом IMM).
EVENT:PageDown	Пользователь нажал клавишу PgDn (только для окон с атрибутом IMM).
EVENT:ScrollTop	Пользователь нажал клавишу Ctrl-PgUp (только для окон с атрибутом IMM).
EVENT:ScrollBottom	Пользователь нажал клавишу Ctrl-PgDn (только для окон с атрибутом IMM).
EVENT:PreAlertKey	Пользователь нажал клавишу с печатным символом (только для окон с атрибутом IMM) или заданную атрибутом ALRT горячую клавишу.
EVENT:AlertKey	Пользователь нажал клавишу с печатным символом (только для окон с атрибутом IMM) или заданную атрибутом ALRT горячую клавишу.
EVENT:Locate	Пользователь нажал видеомagnитофонную клавишу поиска (только для окон с атрибутом IMM).
EVENT:ScrollDrag	Пользователь передвинул бегунок на линейке скроллинга и его новое положение отражено в PROP:VScrollPos (только для окон с атрибутом IMM).
EVENT:ScrollTrack	Пользователь передвигает бегунок на линейке скроллинга и его новое положение отражено в PROP:VScrollPos (только для окон с атрибутом IMM).
EVENT:DroppingDown	Пользователь нажал кнопку “стрелка вниз” (только для окон с атрибутом DROP)
EVENT:DroppedDown	Список раскрыт (только для окон с атрибутом DROP).

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
COMBO(@S8),AT(0,0,20,20),USE(C1),FROM(Que)
COMBO(@S8),AT(20,0,20,20),USE(C2),FROM(Que),KEY(F10Key)
COMBO(@S8),AT(40,0,20,20),USE(C3),FROM(Que),MSG('Button 3')
COMBO(@S8),AT(60,0,20,20),USE(C4),FROM(Que),HLP('Check4Help')
COMBO(@S8),AT(80,0,20,20),USE(C5),FROM(Q) |
,FORMAT('5C~List~15L~Box~'),COLUMN
COMBO(@S8),AT(100,0,20,20),USE(C6),FROM(Que),FONT('Arial',12)
COMBO(@S8),AT(120,0,20,20),USE(C7),FROM(Que),DROP(8)
COMBO(@S8),AT(140,0,20,20),USE(C8),FROM(Que),HVSCROLL,VCR
COMBO(@S8),AT(160,0,20,20),USE(C9),FROM(Que),IMM
COMBO(@S8),AT(180,0,20,20),USE(C10),FROM(Que),CURSOR(CURSOR:Wait)
```

```

COMBO(@S8),AT(200,0,20,20),USE(C11),FROM(Que),ALRT(F10Key)
COMBO(@S8),AT(220,0,20,20),USE(C12),FROM(Que),LEFT
COMBO(@S8),AT(240,0,20,20),USE(C13),FROM(Que),RIGHT
COMBO(@S8),AT(260,0,20,20),USE(C14),FROM(Que),CENTER
COMBO(@N8.2),AT(280,0,20,20),USE(C15),FROM(Que),DECIMAL
COMBO(@S8),AT(300,0,20,20),USE(C16),FROM('Apples|Peaches|Pumpkin|Pie')
COMBO(@S8),AT(320,0,20,20),USE(C17),FROM('TBA')
END
CODE
OPEN(MDICHild)
?C17{PROP:From} = 'Live|Long|And|Prosper'    !Присвоение атрибуту FROM
ACCEPT
CASE ACCEPTED()
OF ?C1
  LOOP X# = 1 to RECORDS(Que)    !Проверить введенное значение на
                                ! наличие в очереди
    GET(Que,X#)
    IF C1 = Que THEN BREAK.      !Прервать цикл если значение совпало
  END
  IF X# > RECORDS(Que)           !Проверить есть ли в очереди значение ?
    Que = C1                     ! и добавить его, если не было
    ADD(Que)
  END
END
END
END

```

Смотри также: LIST, ENTRY

## ELLIPSE (объявить экранный объект эллиптической формы)

```

ELLIPSE ,AT( ) [,USE( )] [,DISABLE] [,COLOR( )] [,FILL( )] [,FULL]
[,SCROLL] [,HIDE] [,LINEWIDTH]

```

<b>ELLIPSE</b>	Помещает в окно или панель инструментов “круглый” экранный объект.
<b>AT</b>	Задаёт первоначальные размеры и расположение данного объекта. Если этот атрибут опущен, то значения по умолчанию выбираются библиотечной процедурой.
<b>USE</b>	Метка соответствия поля, служащая для того, чтобы осуществить ссылку на это поле в исполняемых операторах.
<b>DISABLE</b>	Указывает, что когда в окно WINDOW (или APPLICATION) раскрывается впервые, данный объект выглядит затухеванным.
<b>COLOR</b>	Указывает цвет контура эллипса. Если этот атрибут опущен, то эллипс не имеет линии контура.
<b>FILL</b>	Задаёт цвет области внутри объекта. Если этот атрибут опущен, то

область внутри объекта не закрашивается.

**FULL**

Указывает, что по любому из опущенных параметров атрибута AT (ширина или высота) объект расширяется таким образом, чтобы занимать всю протяженность окна.

**SCROLL**

Указывает, что поле подвергается скроллингу вместе с содержимым окна.

**HIDE**

Задаёт, что когда окно WINDOW или APPLICATION раскрывается в первый раз, данное поле не прорисовывается. Для того, чтобы отобразить это поле нужно использовать оператор UNHIDE.

**LINEWIDTH**

Задаёт толщину линии контура эллипса.

Оператор **ELLIPSE** помещает в окно или панель инструментов “круглый” экраный объект в позиции, заданной его атрибутом AT. Эллипс рисуется внутри ограничивающего прямоугольника, определенного параметрами x, y, ширина и высота атрибута AT. Параметры x и y задают начальную точку, а ширина и высота задают горизонтальный и вертикальный размер ограничивающего прямоугольника. На этот объект не может переключаться фокус и для него не генерируются никакие события.

**Пример:**

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
          ELLIPSE,FILL(COLOR:MENU),FULL !Закрашенный, с черным контуром, на
                                           !весь экран
          ELLIPSE,AT(0,0,20,20)           !Незакрашенный, с черным контуром
          ELLIPSE,AT(0,20,20,20),USE(?Box1),DISABLE !Затуманенный
          ELLIPSE,AT(20,20,20,20),ROUND
          ELLIPSE,AT(40,40,20,20),FILL(COLOR:ACTIVEBORDER)
          !Закрашенный, с черным контуром
          ELLIPSE,AT(60,60,20,20),COLOR(COLOR:ACTIVEBORDER)
          !Закрашенный, с контуром текущего цвета рамки
          ELLIPSE,AT(480,180,20,20),SCROLL !Прокручиваемый вместе с экраном
END
```

**ENTRY (объявить поле для ввода данных)**

```
ENTRY(шаблон),AT() [,CURSOR() [,USE() [,DISABLE] [,KEY() [,MSG()
[,HLP() [,SKIP] [,FONT() [,IMM] [,PASSWORD] [,REQ] [,FULL]
[,SCROLL][,ALRT() [,HIDE] [,TIP] [,READONLY] [DROPID( )] [,TRN]
[, |INS      |] [, |CAP  |] [, |LEFT |]
|OVR      |UPR |      |RIGHT | [,COLOR( )]
|          |   |      |CENTER |
|          |   |      |DECIMAL|
```

<b>ENTRY</b> шаблон	Помещает в окно или панель инструментов поле для ввода данных. Шаблон для отображения данных, который указывает формат вводимых в это поле данных.
<b>AT</b> <b>CURSOR</b>	Задаст первоначальные размеры и расположение вводного поля. Задаст форму, которую должен принимать курсор мыши, при попадании в это поле. Если этот атрибут опущен, то используется значение атрибута CURSOR структуры WINDOW, если же и тот не указан, то курсор имеет форму, используемую в среде Windows по умолчанию.
<b>USE</b> <b>DISABLE</b>	Метка переменной, которая получает значение введенное в данное поле. Указывает, что когда в окно WINDOW (или APPLICATION) раскрывается впервые, данное поле выглядит затухеванным.
<b>KEY</b>	Задаст целочисленную константу или мнемонический код клавиши (или комбинации клавиш), с помощью которой фокус немедленно переключается на это поле.
<b>MSG</b>	Задаст строковую константу, содержащую текст, который следует вывести в строке состояния, когда на данное поле переключен фокус.
<b>HLP</b>	Задаст строковую константу, содержащую идентификатор системы справки для данного поля.
<b>SKIP</b>	Указывает, что для ввода данных в это поле фокус ввода переключается только посредством мыши или клавиши ускоренного доступа, и не задерживается на этом поле после завершения ввода данных.
<b>FONT</b> <b>IMM</b>	Задаст шрифт для вывода текста в поле для ввода данных. Указывает, что как только пользователь нажмет любую клавишу, немедленно происходит генерирование события.
<b>PASSWORD</b> <b>REQ</b> <b>FULL</b>	Отменяет вывод на экран вводимых данных (режим ввода пароля). Задаст, что поле не может оставаться пустым или нулевым. Указывает, что по любому из опущенных параметров атрибута AT (ширина или высота) поле расширяется таким образом, чтобы занимать всю протяженность окна.
<b>SCROLL</b> <b>ALRT</b> <b>HIDE</b>	Указывает, что поле подвергается скроллингу вместе с содержимым окна. Задаст активные горячие клавиши для данного поля. Задаст, что когда окно WINDOW или APPLICATION раскрывается в первый раз, данное поле не прорисовывается. Для того, чтобы отобразить это поле нужно использовать оператор UNHIDE.

<b>TIP</b>	Задает текст, который выводится как “возникающая” подсказка, когда курсор мыши задерживается на экранном объекте.
<b>TRN</b>	Указывает, что объект выводится на “прозрачном” фоне.
<b>READONLY</b>	Указывает, что в поле нельзя ввести данные.
<b>DROPID</b>	Указывает, что данное управляющее поле может служить областью в которой происходит вторая часть операции “потащить и отпустить” - отпусkanie объекта.
<b>INS/OVR</b>	Задает при вводе данных режим вставки или замещения (допустимо только в окнах, имеющих атрибут MASK).
<b>UPR/CAP</b>	Указывает, что данные вводятся прописными буквами или по типу имен собственных (Первая Буква В Каждом Слове Прописная).
<b>LEFT</b>	Задает, что данные в окне списка выравниваются влево.
<b>RIGHT</b>	Задает, что данные в окне списка выравниваются право.
<b>CENTER</b>	Задает, что данные в окне списка выравниваются по центру окна.
<b>DECIMAL</b>	Задает, что данные в окне списка выравниваются по десятичной точке.
<b>COLOR</b>	Указывает, что цвет фона объекта и цвета объекта в активизированном состоянии.

Оператор **ENTRY** помещает в окно или панель инструментов, в заданную атрибутом **AT** позицию поле для ввода данных. Вводимые данные форматируются в соответствии с шаблоном и, когда пользователь завершит ввод данных и переключится на другое поле, заносятся в указанную атрибутом **USE** переменную. Вводимые данные прокручиваются в горизонтальном направлении, чтобы позволить пользователю вводить данные во всю длину **USE**-переменной. Поэтому клавиши стрелок влево и вправо позволяют перемещаться введенные данные в поле типа **ENTRY**.

Когда пользователь вводит данные в поле **ENTRY** с атрибутом **PASSWORD**, в нем вместо вводимых данных отображаются звездочки. Такой режим ввода применяется для ввода конфиденциальных данных типа паролей. Для часто пропускаемых вводных полей используется оператор **ENTRY** с атрибутом **SKIP**.

Поля, служащие только для вывода значения переменной на экран, объявляются оператором **ENTRY** с атрибутом **READONLY**.

Атрибут **MASK** структуры **WINDOW** указывает для всех вводных поле в окне режим ввода данных по шаблону. Это означает, что по мере ввода пользователем каждый символ автоматически проверяется на соответствии шаблону для данного поля, чтобы обеспечить правильность введенных данных (только цифры для числового поля и т.д.). Этот режим ввода вынуждает пользователя вводить данные в формате, указанном в шаблоне для данного поля. Если атрибут **MASK** опущен, Windows позволяет вводить в поле произвольные данные. Это используемый по умолчанию в Windows режим ввода. Режим произвольного ввода означает, что данные форматируются в соответствии с шаблоном поля только после завершения ввода. Если пользователь ввел данные в формате, отличающемся от заданного шаблоном для поля, библиотечная процедура пытается определить использованный



пользователем формат и преобразовать данные в формат, заданный для отображения данных в этом поле. Например, если пользователь ввел “January 1, 1995” в поле с шаблоном @D1, библиотечная процедура отформатирует введенные им данные как “1/1/95”. Это произойдет только после того, как пользователь завершит ввод данных в это поле и перейдет к другому полю. Если библиотечная процедура не может определить, какой формат использовал пользователь, то значение USE-переменной не изменяется. Выдается звуковой сигнал и пользователь возвращается в предыдущее поле, чтобы повторить ввод данных в него.

### Генерируемые события:

EVENT:Selected	В данное поле переключен фокус.
EVENT:Accepted	Пользователь завершил ввод данных в поле.
EVENT:Rejected	Пользователь ввел несоответствующее шаблону значение.
EVENT:PreAlertKey	Пользователь нажал заданную атрибутом ALRT горячую клавишу.
EVENT:AlertKey	Пользователь нажал заданную атрибутом ALRT горячую клавишу.
EVENT:Drop	Успешная операция переноса объекта в данное поле.

Поле **ENTRY** с атрибутом **IMM** дополнительно генерирует события:

EVENT:NewSelection	Пользователь ввел символ.
--------------------	---------------------------

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
ENTRY(@S8),AT(0,0,20,20),USE(E1)
ENTRY(@S8),AT(20,0,20,20),USE(E2),KEY(F10Key)
ENTRY(@S8),AT(40,0,20,20),USE(E3),MSG('Button 3')
ENTRY(@S8),AT(60,0,20,20),USE(E4),HLP('Entry4Help')
ENTRY(@S8),AT(80,0,20,20),USE(E5),DISABLE
ENTRY(@S8),AT(100,0,20,20),USE(E6),FONT('Arial',12)
ENTRY(@S8),AT(120,0,20,20),USE(E7),REQ,INS,CAP
ENTRY(@S8),AT(140,0,20,20),USE(E8),SCROLL,OVR,UPR
ENTRY(@S8),AT(160,0,20,20),USE(E9),IMM
ENTRY(@S8),AT(180,0,20,20),USE(E10),CURSOR(CURSOR:Wait)
ENTRY(@S8),AT(200,0,20,20),USE(E11),ALRT(F10Key)
ENTRY(@S8),AT(240,0,20,20),USE(E13),RIGHT
ENTRY(@S8),AT(260,0,20,20),USE(E14),CENTER
ENTRY(@N8.2),AT(280,0,20,20),USE(E15),DECIMAL
END
```

**GROUP (объявить группу экранных объектов)**

```

GROUP(текст) ,AT() [,CURSOR()] [,USE()] [,DISABLE] [,KEY()] [,MSG()]
    [,HLP()] [,FONT()] [,BOXED] [,FULL] [,SCROLL] [,HIDE]
    [,ALRT( )] [,SKIP] [,TIP( )] [,DROPID( )] [,COLOR()][,BEVEL()]
    объекты
END

```

<b>GROUP</b>	Объявляет группу объектов, на которые будут ссылки как на единое целое.
<b>текст</b>	Строчковая константа, содержащая подсказку для группы объектов. Константа может содержать амперсанд (&), обозначающий горячую “букву” в подсказке. Текст подсказки отображается на экране, только если дополнительно указан атрибут <b>BOXED</b> .
<b>AT</b>	Задает первоначальные размеры и расположение группы. Если этот атрибут опущен, то по умолчанию параметры выбираются библиотечной процедурой.
<b>CURSOR</b>	Задает форму, которую должен принимать курсор мыши, при попадании в пределы группы. Если этот атрибут опущен, то используется значение атрибута <b>CURSOR</b> структуры <b>WINDOW</b> , если же и тот не указан, то курсор имеет форму, используемую в среде Windows по умолчанию.
<b>USE</b>	Метка соответствия поля, служащая для того, чтобы осуществить ссылку на эту группу в исполняемых операторах.
<b>DISABLE</b>	Указывает, что когда в окно <b>WINDOW</b> (или <b>APPLICATION</b> ) раскрывается впервые, данная группа выглядит затуманенной.
<b>KEY</b>	Задает целочисленную константу или мнемонический код клавиши (или комбинации клавиш), с помощью которой фокус немедленно переключается на первый объект этой группы.
<b>MSG</b>	Задает строковую константу, содержащую текст, который следует вывести в строке состояния, когда на любой объект из данной группы переключен фокус.
<b>HLP</b>	Задает строковую константу, содержащую идентификатор системы справки используемой по умолчанию для любого объекта из данной группы.
<b>FONT</b>	Задает шрифт для вывода текста в данной группе и шрифт, используемый по умолчанию для любого объекта данной группы.
<b>BOXED</b>	Задает наличие вокруг группы рамки, проведенной одинарной линией, и заголовка в верхней части рамки.
<b>FULL</b>	Указывает, что по любому из опущенных параметров атрибута <b>AT</b> (ширина или высота) группа расширяется таким образом, чтобы занимать всю протяженность окна.
<b>SCROLL</b>	Указывает, что группа подвергается скроллингу вместе с содержимым окна.
<b>HIDE</b>	Задает, что когда окно <b>WINDOW</b> или <b>APPLICATION</b> раскрывается в первый раз, сам объект - группа и все объекты в нем не прорисовываются.

Для того, чтобы отобразить группу и объекты в ней нужно использовать оператор UNHIDE.

#### ALRT SKIP

Задаёт активные горячие клавиши для объектов данной группы.

Указывает, что на объекты данной группы фокус ввода переключается только посредством мыши или клавиши ускоренного доступа.

#### TIP

Задаёт текст, который выводится как “возникающая” подсказка, когда курсор мыши задерживается на экранном объекте.

#### DROPID

Указывает, что данное окно может служить областью, в которой происходит вторая часть операции “перетащить и отпустить” - отпуская объект.

#### COLOR

Задаёт цвет фона по умолчанию и цвета переднего плана и фона для активизированных объектов в группе.

#### BEVEL

Задаёт эффект объёмности границ объекта.

#### объекты

Объявления объектов на которые можно ссылаться по групповому имени.

Оператор **GROUP** объявляет группу объектов, на которые будут ссылки как на единое целое. Объединение в группу позволяет пользователю для перемещения между объектами группы использовать вместо клавиши TAB использовать клавиши стрелок и обеспечивает общие атрибуты MSG и HLP для всех объектов группы. На группу как объект не переключается фокус.

### Генерируемые события:

EVENT:Drop Успешная операция “перетащить и отпустить” в данный объект.

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
GROUP('Group 1'),USE(?G1),KEY(F10Key)
ENTRY(@S8),AT(0,0,20,20),USE(?E1)
ENTRY(@S8),AT(20,0,20,20),USE(?E2)
END
GROUP('Group 2'),USE(?G2),MSG('Group 2'),CURSOR(CURSOR:Wait)
ENTRY(@S8),AT(40,0,20,20),USE(?E3)
ENTRY(@S8),AT(60,0,20,20),USE(?E4)
END
GROUP('Group 3'),USE(?G3),AT(80,0,20,20),BOXED
ENTRY(@S8),AT(80,0,20,20),USE(?E5)
ENTRY(@S8),AT(100,0,20,20),USE(?E6)
END
GROUP('Group 4'),USE(?G4),FONT('Arial',12)
ENTRY(@S8),AT(120,0,20,20),USE(?E7)
ENTRY(@S8),AT(140,0,20,20),USE(?E8)
END
END
```

**IMAGE (объявить экранное поле, содержащее изображение)**

```

IMAGE(имя_файла),AT( ) [,USE( )] [,DISABLE] [,FULL] [,SCROLL] [,HIDE]
[,|HSCROLL|]
|VSCROLL|
|HVSCROLL|

```

<b>IMAGE</b> имя_файла	Помещает графическое изображение в окно или панель инструментов. Строковая константа, содержащая имя файла, подлежащего выводу в этом поле.
<b>AT</b>	Задаёт первоначальные размеры и расположение изображения. Если этот атрибут опущен, то по умолчанию параметры выбираются библиотечной процедурой.
<b>USE</b>	Метка соответствия поля, служащая для того, чтобы осуществить ссылку на этот объект в исполняемых операторах.
<b>DISABLE</b>	Указывает, что когда в окно WINDOW (или APPLICATION) раскрывается впервые, данный объект выглядит затухеванным.
<b>FULL</b>	Указывает, что по любому из опущенных параметров атрибута AT (ширина или высота) объект расширяется таким образом, чтобы занимать всю протяженность окна.
<b>SCROLL</b>	Указывает, что объект подвергается скроллингу вместе с содержимым окна.
<b>HIDE</b>	Задаёт, что когда окно WINDOW или APPLICATION раскрывается в первый раз, объект не прорисовывается. Для того, чтобы отобразить объект нужно использовать оператор UNHIDE.
<b>HSCROLL</b>	Указывает, что когда изображение шире области, выделенной для его отображения, к нему автоматически присоединяется линейка горизонтального скроллинга.
<b>VSCROLL</b>	Указывает, что когда изображение выше области, выделенной для его отображения, к нему автоматически присоединяется линейка вертикального скроллинга.
<b>HVSCROLL</b>	Указывает, что когда изображение шире и выше области, выделенной для его отображения к нему автоматически присоединяются линейки и вертикального и горизонтального скроллинга.

Оператор **IMAGE** помещает графическое изображение в окно или панель инструментов; место и размеры изображения заданы атрибутом AT. Изображение может быть следующего формата: .BMP, .ICO, .PCX, .GIF, .GPG или .WMF. На объект такого типа не переключается фокус и для него не генерируются события.

**Пример:**

```

MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
        IMAGE('PIC.BMP'),AT(0,0,20,20),USE(?I1)
        IMAGE('PIC.WMF'),AT(40,0,20,20),USE(?I3),SCROLL
END

```

## LINE (экранный объект - прямая линия)

LINE, AT( ) [,USE( )] [,DISABLE] [,COLOR( )] [,FULL] [,SCROLL] [,HIDE]  
[,LINEWIDTH( )]

<b>LINE</b>	Помещает в окно или панель инструментов прямую линию
<b>AT</b>	Задаёт первоначальные размеры и расположение объекта. Если этот атрибут опущен, то по умолчанию параметры выбираются библиотечной процедурой.
<b>USE</b>	Метка соответствия поля, служащая для того, чтобы осуществить ссылку на этот объект в исполняемых операторах.
<b>DISABLE</b>	Указывает, что когда в окно WINDOW (или APPLICATION) раскрывается впервые, данный объект выглядит затухеванным.
<b>COLOR</b>	Задаёт цвет линии. Если этот атрибут опущен, то цвет линии - чёрный.
<b>FULL</b>	Указывает, что по любому из опущенных параметров атрибута AT (ширина или высота) объект расширяется таким образом, чтобы занимать всю протяженность окна.
<b>SCROLL</b>	Указывает, что объект подвергается скроллингу вместе с содержимым окна.
<b>HIDE</b>	Задаёт, что когда окно WINDOW или APPLICATION раскрывается в первый раз, объект не прорисовывается. Для того, чтобы отобразить объект нужно использовать оператор UNHIDE.
<b>LINEWIDTH</b>	Задаёт толщину линии.

Оператор **LINE** помещает в окно или панель инструментов, в заданную атрибутом AT позицию отрезок прямой линии. Параметры x и y атрибута AT задают координаты начальной точки отрезка. А параметры ширина и высота задают горизонтальное и вертикальное смещение конечной точки отрезка. Если оба эти параметра положительные, то линия идет вправо и вниз от начальной точки. Если параметр ширина отрицательный, то линия идет влево. Если отрицательным является параметр высота, то линия идет выше начальной точки. Если или ширина, или высота равны 0, то линия вертикальная или горизонтальная. На объект такого типа не переключается фокус и для него не генерируются события.

### Ширина

положительная  
отрицательная  
положительная  
отрицательная  
нуль  
нуль  
положительная  
отрицательная

### Высота

положительная  
положительная  
отрицательная  
отрицательная  
положительная  
отрицательная  
нуль  
нуль

### Результат

вправо и вниз от начальной точки  
влево и вниз от начальной точки  
вправо и вверх от начальной точки  
влево и вверх от начальной точки  
вертикальная, вниз от начальной точки  
вертикальная, вверх от начальной точки  
горизонтальная, вправо от начальной точки  
горизонтальная, влево от начальной точки

Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
        LINE,AT(60,60,20,20),COLOR(COLOR:ACTIVEBORDER)      !Цвет рамки
        LINE,AT(480,180,20,20),SCROLL                        !Прокрутка вместе с экраном
END
```

LIST (объявить окно списка)

LIST ,FROM( ) ,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,KEY( )]			
[,MSG( )] [,HLP( )] [,SKIP] [,FONT( )] [,FORMAT( )] [,DROP]			
[,COLUMN] [,VCR] [,FULL] [,SCROLL] [,NOBAR] [,GRID( )] [,TRN]			
[,ALRT( )] [,HIDE] [,DRAGID( )] [,DROPID( )] [,TIP( )] [,COLOR( )]			
[, MARK( ) ] [,	HSCROLL	] [,	LEFT
IMM	VSCROLL		RIGHT
	HVSCROLL		CENTER
			DECIMAL

LIST	Помещает в окно или панель инструментов прокручиваемый список элементов данных.
FROM	Задает источник для выводимых в списке элементов данных.
AT	Задает первоначальные размеры и расположение окна списка. Если этот атрибут опущен, то по умолчанию параметры выбираются библиотечной процедурой.
CURSOR	Задает форму, которую должен принимать курсор мыши, при попадании в этот объект. Если этот атрибут опущен, то используется значение атрибута CURSOR структуры WINDOW, если же и тот не указан, то курсор имеет форму, используемую в среде Windows по умолчанию.
USE	Метка соответствия поля, которая служит для ссылок на данное поле в исполняемых операторах, или метка переменной, которая получает значение выбранного элемента данных.
DISABLE	Указывает, что когда в окно WINDOW (или APPLICATION) раскрывается впервые, данное поле выглядит затухеванным.
KEY	Задает целочисленную константу или мнемонический код клавиши (или комбинации клавиш), с помощью которой фокус немедленно переключается на это окно списка.
MSG	Задает строковую константу, содержащую текст, который следует вывести в строке состояния, когда на данное поле переключен фокус.
HLP	Задает строковую константу, содержащую идентификатор системы справки для данного поля.фирмами.
SKIP	Указывает, что на это поле фокус не переключается, доступ к нему может быть осуществлен только посредством мыши или клавиш ускоренного доступа.
FONT	Задает шрифт для вывода текста в окне списка.

<b>FORMAT</b>	Задает формат для отображения данных в окне списка. Форматирование может включать в себя наличие пиктограммы, экранного объекта, представляющего древовидную структуру и цвета окна списка.
<b>DROP</b>	Задает выпадающее окно списка и число элементов данных, которое содержит выпадающая часть.
<b>COLUMN</b>	В многоколоночном окне списка задает последовательность выделенных ячеек.
<b>VCR</b>	При наличии линейки горизонтального скроллинга задает наличие слева от линейки кнопок, подобных клавишам управления видеомagneитофоном.
<b>FULL</b>	Указывает, что по любому из опущенных параметров атрибута AT (ширина или высота) окно расширяется таким образом, чтобы занимать всю протяженность окна.
<b>SCROLL</b>	Указывает, что поле подвергается скроллингу вместе с содержимым окна.
<b>NOBAR</b>	Задает, что выделенная полоса курсор отображается, только когда фокус переключается на окно списка.
<b>ALRT</b>	Задает активные горячие клавиши для данного поля.
<b>HIDE</b>	Задает, что когда окно WINDOW или APPLICATION раскрывается в первый раз, данное поле не прорисовывается. Для того, чтобы отобразить это поле нужно использовать оператор UNHIDE.
<b>DRAGID</b>	Указывает, что данный объект может служить областью - источником данных или объектов для операции “потащить и отпустить”.
<b>DROPID</b>	Указывает, что данное управляющее поле может служить областью в которой происходит вторая часть операции “потащить и отпустить” - отпускание объекта.
<b>TIP</b>	Задает текст, который выводится как “возникающая” подсказка, когда курсор мыши задерживается на экранном объекте.
<b>GRID</b>	Задает цвет линий сетки между колонками в списке.
<b>TRN</b>	Указывает, что объект выводится на “прозрачном” фоне.
<b>COLOR</b>	Задает цвет фона по умолчанию и цвета переднего плана и фона для активизированного объекта.
<b>MARK</b>	Задает режим выбора нескольких элементов одновременно.
<b>IMM</b>	Задает генерацию события при нажатии пользователем любой клавиши
<b>HSCROLL</b>	Указывает, что когда какая-либо часть данных по горизонтали не помещается в окне списка, к нему автоматически присоединяется линейка горизонтального скроллинга.
<b>VSCROLL</b>	Указывает, что когда какая-либо часть данных по вертикали не помещается в окне списка, к нему автоматически присоединяется линейка вертикального скроллинга.
<b>HVSCROLL</b>	Указывает, что когда какая-либо часть данных не помещается в окне списка, к нему автоматически присоединяются линейки и вертикального

и горизонтального скроллинга.

<b>LEFT</b>	Задает, что данные в окне списка выравниваются влево.
<b>RIGHT</b>	Задает, что данные в окне списка выравниваются право.
<b>CENTER</b>	Задает, что данные в окне списка выравниваются по центру окна.
<b>DECIMAL</b>	Задает, что данные в окне списка выравниваются по десятичной точке.

С помощью поля **LIST** в заданном атрибуте AT месте окна (или панели инструментов) помещается прокручиваемый список элементов данных. Выводимые в окне списка элементы данных берутся из очереди или строковой переменной, указанной атрибутом FROM и форматируются на основании параметров, указанных в атрибуте FORMAT (форматирование может включать в себя наличие пиктограммы, экранного объекта, представляющего древовидную структуру и цвета окна списка).

Когда для окна списка сгенерировалось событие EVENT:Accepted, функция CHOICE возвращает номер элемента очереди (значение возвращаемое функцией POINTER(очередь)) выбранного пользователем элемент. Независимо от того, имеется ли атрибут AUTO или нет, выводимые в окне списка данные обновляются при каждом выполнении цикла ACCEPT.

Пока на окно списка, имеющее атрибут DROP, не переключен фокус и пользователь не нажал клавишу “стрелка вниз” или не щелкнул по пиктограмме справа от выводимых данных, в нем выводится только выбранный в данный момент элемент данных. Когда же он выполнит одно из этих действий, появляется (“выпадает”) список элементов данных, позволяющий пользователю выбрать один из них.

Поле LIST с атрибутом IMM генерирует события каждый раз, когда пользователь переместил выделенную полосу-курсор на другой элемент списка, или нажимал клавишу, вызывающую скроллинг высвеченных элементов списка. Тем самым программе предоставляется удобная возможность повторно заполнить высвечиваемую очередь или получить выделенную в данный момент запись для того, чтобы вывести на экран другие поля этой записи. Если кроме того, имеется атрибут VSCROLL, то всегда высвечивается линейка вертикального скроллинга, и, когда пользователь щелкнет мышью, указывая на линейку, то генерируется событие, но перемещения элементов в списке не происходит (это должна сделать программа). Чтобы определить положение скроллинга списка, можно прочитать значение свойства PROP:VscrollPos (диапазон от 0 - начало списка, до 255 - конец списка).

У окна списка с атрибутом VCR имеются слева от линейки горизонтального скроллинга (если она есть) кнопки управления скроллингом, подобные кнопкам управления видеомagnитофоном. Эти кнопки позволяют пользователю использовать мышь для прокрутки списка.

Окно списка с атрибутом DRAGID может служить источником в операции “потащить и



отпустить”, обеспечивая данные, которые должны быть перемещены или скопированы в другой экраный объект. Окно списка с атрибутом DROPID может служить областью в которой происходит вторая часть операции “потащить и отпустить” - отпускание объекта, принимая данные от другого объекта. Вместе эти атрибуты служат для того, чтобы задать “ярлыки” для операции “потащить и отпустить”, которые определяют допустимую цель операции. Для выполнения обмена данными вместе с процедурой SETDROPID используются функции DRAGID() и DROPID().

### Генерируемые события:

EVENT:Selected	На объект переключен фокус ввода.
EVENT:Accepted	Пользователь выбрал элемент из списка.
EVENT:NewSelection	Изменился выбор элемента из списка (полоса-курсор переместилась вверх или вниз)
EVENT:ScrollUp	Пользователь нажал клавишу “стрелка вверх” (только для окна с атрибутом IMM).
EVENT:ScrollDown	Пользователь нажал клавишу “стрелка вниз” (только для окна с атрибутом IMM).
EVENT:PageUp	Пользователь нажал PgUp (только для окна с атрибутом IMM).
EVENT:PageDown	Пользователь нажал PgDn (только для окна с атрибутом IMM).
EVENT:ScrollTop	Пользователь нажал Ctrl-PgUp (только для окна с атрибутом IMM).
EVENT:ScrollBottom	Пользователь нажал Ctrl-PgDn (только для окна с атрибутом IMM).
EVENT:Locate	Пользователь нажал видеомagnитофонную клавишу поиска (только для окна с атрибутом IMM)
EVENT:ScrollDrag	Пользователь передвинул бегунок на линейке скроллинга и его новое положение отражено в PROP:VScrollPos (только для окон с атрибутом IMM).
EVENT:ScrollTrack	Пользователь передвигает бегунок на линейке скроллинга и его новое положение отражено в PROP:VScrollPos (только для окон с атрибутом IMM).
EVENT:AlertKey	Пользователь нажал клавишу отображаемого символа (только для окна с атрибутом IMM) или заданную атрибутом ALRT горячую клавишу
EVENT:Dragging	Курсор мыши находится над потенциальной целью операции “потащить и отпустить” (только для окна с атрибутом DRAGID)
EVENT:Drag	Курсор мыши отпущен над целью операции “потащить и отпустить” (только для окна с атрибутом DRAGID).

EVENT:Drop	Курсор мыши отпущен над целью операции “поташить и отпустить” (только для окна с атрибутом DRAGID).
EVENT:DroppingDown	Пользователь затребовал раскрытие окна списка (только для окна с атрибутом DROP). Оператор CYCLE прерывает раскрытие окна списка.
EVENT:DroppedDown	Пользователь раскрыл окно списка (только для окна с атрибутом DROP).
EVENT:Expanding	Пользователь щелкнул на квадратике раскрытия древовидной структуры (только при наличии формата T в строке атрибута FORMAT). Оператор CYCLE прерывает раскрытие древовидной структуры.
EVENT:Expanded	Пользователь щелкнул на квадратике раскрытия древовидной структуры (только при наличии формата T в строке атрибута FORMAT).
EVENT:Contracting	Пользователь щелкнул на квадратике закрытия древовидной структуры (только при наличии формата T в строке атрибута FORMAT). Оператор CYCLE прерывает закрытие древовидной структуры.
EVENT:Contracted	Пользователь щелкнул на квадратике закрытия древовидной структуры (только при наличии формата T в строке атрибута FORMAT).

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
LIST,AT(0,0,20,20),USE(?L1),FROM(Que),IMM
LIST,AT(20,0,20,20),USE(?L2),FROM(Que),KEY(F10Key)
LIST,AT(40,0,20,20),USE(?L3),FROM(Que),MSG('Button 3')
LIST,AT(60,0,20,20),USE(?L4),FROM(Que),HLP('Check4Help')
LIST,AT(80,0,20,20),USE(?L5),FROM(Q),FORMAT('5C~List~15L~Box~'),COLUMN
LIST,AT(100,0,20,20),USE(?L6),FROM(Que),FONT('Arial',12)
LIST,AT(120,0,20,20),USE(?L7),FROM(Que),DROP(6)
LIST,AT(140,0,20,20),USE(?L8),FROM(Que),HVSCROLL,VCR
LIST,AT(180,0,20,20),USE(?L10),FROM(Que),CURSOR(CURSOR:Wait)
LIST,AT(200,0,20,20),USE(?L11),FROM(Que),ALRT(F10Key)
LIST,AT(220,0,20,20),USE(?L12),FROM(Que),LEFT
LIST,AT(240,0,20,20),USE(?L13),FROM(Que),RIGHT
LIST,AT(260,0,20,20),USE(?L14),FROM(Que),CENTER
LIST,AT(280,0,20,20),USE(?L15),FROM(Que),DECIMAL
LIST,AT(300,0,20,20),USE(?L16),FROM('Apples|Peaches|Pumpkin|Pie')
LIST,AT(320,0,20,20),USE(?L17),FROM('TBA')
END
CODE

OPEN(MDIChild)
?L17{PROP:From} = 'Live|Long|And|Prosper'
```

Смотри также: COMBO, DRAGID, DROPID, SETDROPID

### OLE (объявить объект, содержащий объект OLE or .OCX)

```
OLE ,AT( ) [,CURSOR( )] [,USE()] [,DISABLE] [,KEY()] [,MSG()]
[,HLP()] [,SKIP] [,FULL] [,TIP()] [,SCROLL] [,ALRT( )] [,HIDE]
[,FONT( )] [,DROPID( )] [,COMPATIBILITY( )]
      [, CREATE( )] | [, CLIP ] | [,свойство( значение )]
      | OPEN( ) | AUTOSIZE |
      | LINK( ) | STRETCH |
      | DOCUMENT( ) | ZOOM |
[ MENUBAR
  объявления меню и/или пунктов
END ]
END
```

- OLE** Поместить в окно или панель инструментов поле OLE (Object Linking and Embedding) или .OCX.
- AT** Задаёт первоначальные размеры и положение экранного объекта. Если параметр опущен, то значения по умолчанию выбираются самим объектом.
- CURSOR** Указывает форму курсора мыши при попадании его в пределы данного объекта. Если атрибут опущен то используется значение атрибута CURSOR структуры WINDOW, если и он отсутствует, то используется курсор принятый в Windows по умолчанию.
- USE** Метка соответствия или имя переменной в которую заносится “значение” экранного объекта.
- DISABLE** Указывает, что когда в окно WINDOW (или APPLICATION) раскрывается впервые, данное поле выглядит затухеванным.
- KEY** Задаёт целочисленную константу или мнемонический код клавиши (или комбинации клавиш), с помощью которой фокус немедленно переключается на этот объект.
- MSG** Задаёт строковую константу, содержащую текст, который следует вывести в строке состояния, когда на данное поле переключен фокус.
- HLP** Задаёт строковую константу, содержащую идентификатор системы справки для данного поля.
- SKIP** Указывает, что на это поле фокус не переключается, доступ к нему может быть осуществлен только посредством мыши или клавиш ускоренного доступа.
- FULL** Указывает, что по любому из опущенных параметров атрибута AT (ширина или высота) объект расширяется таким образом, чтобы занимать

	всю протяженность окна.
<b>TIP</b>	Задает текст, который выводится как “возникающая” подсказка, когда курсор мыши задерживается на экранном объекте.
<b>SCROLL</b>	Указывает, что поле подвергается скроллингу вместе с содержимым окна.
<b>ALRT</b>	Задает активные горячие клавиши для данного поля.
<b>HIDE</b>	Задает, что когда окно WINDOW или APPLICATION раскрывается в первый раз, данное поле не прорисовывается. Для того, чтобы отобразить это поле нужно использовать оператор UNHIDE.
<b>FONT</b>	Задает шрифт для вывода текста в объекте
<b>DROPID</b>	Указывает, что данное управляющее поле может служить областью в которой происходит вторая часть операции “потащить и отпустить” - отпускание объекта.
<b>COMPATIBILITY</b>	Задает режим совместимости для некоторых объектов OLE и .OCX, которые того требуют.
<b>CREATE</b>	Указывает, что в поле создается новый объект OLE или .OCX.
<b>OPEN</b>	Указывает, данный экранный объект открывает объект из специального файла (OLE Compound Storage file). При открытии объекта загружается сохраненный вариант значений свойств контейнера (клиента), поэтому не нужно задавать их заново.
<b>LINK</b>	Указывает, что данный OLE-объект представляет собой связь с объектом из некоего файла, например, таблицы Excel.
<b>DOCUMENT</b>	Указывает, что данный OLE-объект представляет собой объект из некоего файла, например, таблицы Excel.
<b>CLIP</b>	Указывает, что для внедряемого объекта высвечивается только та его часть, которая помещается рамки, определенные атрибутом AT экранного объекта- контейнера. Если внедряемый объект больше чем определенное для него экранное поле OLE, то отображается только его верхний левый угол.
<b>AUTOSIZE</b>	Указывает, что когда параметры атрибута AT поля-контейнера во время выполнения программы изменяются (PROP:At), внедряемый объект автоматически изменяет свои размеры.
<b>STRETCH</b>	Указывает, что OLE-объект растягивается таким образом, чтобы заполнить весь OLE-контейнер, размеры которого заданы атрибутом AT.
<b>ZOOM</b>	Указывает, что размеры OLE-объекта увеличиваются таким образом, чтобы заполнить весь OLE-контейнер, размеры которого заданы атрибутом AT, не нарушая пропорциональности объекта.
<i>свойство</i>	Строковая константа, содержащая имя пользовательского (не системного) свойства, устанавливаемого для данного экранного объекта.
<i>значение</i>	Строковая константа, содержащая значение свойства или метку соответствия для свойства.
<b>MENUBAR</b>	Определяет структуру меню для объекта. Это точно такая же структура как MENUBAR в окне APPLICATION или WINDOW и оно

вливается в меню прикладной программы.

**меню и/или пункты** Объявления меню и/или пунктов меню, которые описывают возможный в нем выбор.

Структура **OLE** размещает в окне (или панели инструментов) объект OLE или .OCX, положение и размеры которого задаются атрибутом AT. Атрибут свойство позволяет указать значения дополнительных свойств, которые могут требоваться объектом OLE или .OCX. Это свойства, задание которых обеспечивает правильную работу объекта OLE или .OCX и не являющиеся стандартными для Clation свойствами, такими как AT, CURSOR или USE. Пользовательский экранный объект воспринимает только те свойства, которые для него определены. Допустимые свойства и их значения должны быть описаны в документации на эти пользовательские экранные объекты. Для одного объекта OLE можно указать несколько атрибутов свойство.

### Генерируемые события:

EVENT:Selected	На объект переключен фокус .
EVENT:Accepted	Пользователь завершил работу с объектом
EVENT:PreAlertKey	Пользователь нажал определенную атрибутом ALRT горячую клавишу.
EVENT:AlertKey	Пользователь нажал определенную атрибутом ALRT горячую клавишу.
EVENT:Drop	Успешная операция “поташить и отпустить” в область данного объекта.

### Пример:

```

PROGRAM
MAP
  INCLUDE('OCX.CLW')
END
W WINDOW('OCX Controls'),AT(,200,200),RESIZE,STATUS(-1,-1),SYSTEM
  MENUBAR
    ITEM('E&xit!'),USE(?Exit)
    ITEM('&About!'),USE(?About)
    ITEM('&Properties!'),USE(?Property)
  END
  OLE,AT(0,0,0,0),USE(?oc1),HIDE,CREATE('COMCTL.ImagelistCtrl.1')
  END
  OLE,AT(0,0,150,20),USE(?oc2),CREATE('TOOLBAR.ToolbarCtrl.1')
  END
END
CODE
OPEN(W)
?OC1{'ListImages.Add(1,xyz,' & ocxloadimage('IRCLOCK.BMP') & ')}
```

```

?OC1{'ListImages.Add(2,abc,' & ocxloadimage('IRCLOCK2.BMP') & '')}
?oc2{'ImageList'} = ?oc1{PROP:Object}
LOOP X# = 1 TO 3
  ?oc2{'Buttons.Add(,,,1)'}
  ?oc2{'Buttons.Add(,,,2)'}
END
ACCEPT
CASE EVENT()
  OF EVENT:Accepted
    CASE FIELD()
      OF ?Exit
        BREAK
      OF ?About
        ?oc1{'AboutBox'}      !Вывести окно AboutBox объекта OCX
      OF ?Property
        ?oc1{PROP:DoVerb} = -7 !Вывести диалоговое окно свойств объекта OCX
    END
  END
END
END

```

## OPTION (объявить группу кнопок с зависимой фиксацией)

```

OPTION(текст) ,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,KEY( )
[,MSG( )] [,HLP( )] [,BOXED] [,FULL] [,SCROLL] [,HIDE] [,COLOR( )
[,FONT( )] [,ALRT( )] [,SKIP] [,DROPID( )] [,TIP( )] [,TRN] [,BEVEL( )
  кнопки
END

```

### OPTION

*текст*

объявляет группу кнопок с зависимой фиксацией

Строковая константа, содержащая подсказку для группы объектов. Константа может содержать амперсанд (&), обозначающий горячую “букву” в подсказке. Текст подсказки отображается на экране, только если дополнительно указан атрибут BOXED.

### AT

Задаёт первоначальные размеры и расположение группы. Если этот атрибут опущен, то по умолчанию параметры выбираются библиотечной процедурой.

### CURSOR

Задаёт форму, которую должен принимать курсор мыши, при попадании в пределы группы. Если этот атрибут опущен, то используется значение атрибута CURSOR структуры WINDOW, если же и тот не указан, то курсор имеет форму, используемую в среде Windows по умолчанию.

### USE

Метка строковой переменной, в которую заносится значение текста для выбранной пользователем кнопки (без амперсанда, указывающего клавишу ускоренного доступа).

### DISABLE

Указывает, что когда в окно WINDOW (или APPLICATION) раскрывается впервые, данная группа выглядит затухеванной.

<b>KEY</b>	Задает целочисленную константу или мнемонический код клавиши (или комбинации клавиш), с помощью которой фокус немедленно переключается на включенную в данный момент кнопку этой группы.
<b>MSG</b>	Задает строковую константу, содержащую текст, который по умолчанию следует вывести в строке состояния, когда на любую кнопку из данной группы переключен фокус.
<b>HLP</b>	Задает строковую константу, содержащую идентификатор системы справки используемой по умолчанию для любой кнопки из данной группы.
<b>FONT</b>	Задает шрифт для вывода текста в данной группе и шрифт, используемый по умолчанию для любой кнопки данной группы.
<b>BOXED</b>	Задает наличие вокруг группы рамки, проведенной одинарной линией, и заголовка в верхней части рамки.
<b>FULL</b>	Указывает, что по любому из опущенных параметров атрибута AT (ширина или высота) группа расширяется таким образом, чтобы занимать всю протяженность окна.
<b>SCROLL</b>	Указывает, что группа подвергается скроллингу вместе с содержимым окна.
<b>HIDE</b>	Задает, что когда окно WINDOW или APPLICATION раскрывается в первый раз, сам объект - группа и все объекты в нем не прорисовываются. Для того, чтобы отобразить группу и объекты в ней нужно использовать оператор UNHIDE.
<b>ALRT</b>	Задает активные горячие клавиши для объектов данной группы.
<b>SKIP</b>	Указывает, что на объекты данной группы фокус ввода не переключается и доступ к ним может быть осуществлен только посредством мыши или клавиши ускоренного доступа.
<b>DROPID</b>	Указывает, что данная группа может служить областью в которой происходит вторая часть операции “потащить и отпустить” - отпуская объект.
<b>TIP</b>	Задает текст, который выводится как “возникающая” подсказка, когда курсор мыши задерживается на экранном объекте.
<b>TRN</b>	Указывает, что объект выводится на “прозрачном” фоне.
<b>COLOR</b>	Задает цвет фона для объекта.
<b>BEVEL</b>	Задает эффект объемности границ объекта.
<i>кнопки</i>	Несколько объявлений кнопок с зависимой фиксацией.

Оператор **OPTION** объявляет группу кнопок с зависимой фиксацией, которые предлагают пользователю список возможных альтернатив. Несколько объектов типа RADIO в структуре OPTION объявляют альтернативы, предлагаемые пользователю.

Фокус ввода переключается между кнопками с независимой фиксацией, с помощью которых указывается, что только одна отдельная кнопка включена. Это означает, что события, генерируемые когда пользователь переключает фокус внутри структуры OPTION, являются характерными для поля, на которое производится воздействие, а не для структуры

OPTION, в которой они содержатся.

В строковую переменную, использованную в качестве атрибута USE структуры OPTION заносится текст, соответствующий выбранной пользователем кнопке. Функция CHOICE(?Option) возвращает номер выбранного поля типа RADIO. Если содержимое атрибута USE структуры OPTION является числовой величиной, то в нее заносится номер выбранной пользователем кнопки (значение, возвращаемое функцией CHOICE).

Ситуация, когда не выбрано ни одно поле RADIO является нормальной, и случается только когда USE-переменная структуры OPTION содержит значение, не совпадающее ни с одним ее компонентом - полем типа RADIO. Эта ситуация длится только до тех пор, пока пользователь не выбрал одно из полей RADIO.

### Генерируемые события:

EVENT:Selected	На одно из полей RADIO переключился фокус.
EVENT:Accepted	Одно из полей RADIO выбрано пользователем
EVENT:PreAlertKey	Пользователь нажал заданную атрибутом ALRT горячую клавишу.
EVENT:AlertKey	Пользователь нажал заданную атрибутом ALRT горячую клавишу.
EVENT:Drop	Кнопка мыши отпущена над объектом - целью операции "потащить и отпустить"

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
    OPTION('Option 1'),USE(OptVar1),KEY(F10Key),HLP('Option1Help')
    RADIO('Radio 1'),AT(0,0,20,20),USE(?R1)
    RADIO('Radio 2'),AT(20,0,20,20),USE(?R2)
END
OPTION('Option 2'),USE(OptVar2),MSG('Option 2'),SCROLL
    RADIO('Radio 3'),AT(40,0,20,20),USE(?R3)
    RADIO('Radio 4'),AT(60,0,20,20),USE(?R4)
END
OPTION('Option 3'),USE(OptVar3),AT(80,0,20,20),BOXED
    RADIO('Radio 5'),AT(80,0,20,20),USE(?R5)
    RADIO('Radio 6'),AT(100,0,20,20),USE(?R6)
END
OPTION('Option 4'),USE(OptVar4),FONT('Arial',12),CURSOR(CURSOR:Wait)
    RADIO('Radio 7'),AT(120,0,20,20),USE(?R7)
    RADIO('Radio 8'),AT(140,0,20,20),USE(?R8)
END
END
```

**Смотри также:** RADIO



**PANEL (объявить область-панель в окне))**

**PANEL ,AT( ) [,USE( )] [,DISABLE] [,FULL] [,FILL( )] [,SCROLL] [,HIDE]  
[,BEVEL( )]**

<b>PANEL</b>	Объявить область в окне или на панели инструментов.
<b>AT</b>	Задаёт первоначальные размеры и расположение группы. Если этот атрибут опущен, то по умолчанию параметры выбираются библиотечной процедурой.
<b>USE</b>	Мнемоническая метка соответствия поля служащая для ссылок на него в исполняемых операторах.
<b>DISABLE</b>	Указывает, что когда в окно WINDOW (или APPLICATION) раскрывается впервые, данная группа выглядит затушеванной.
<b>FULL</b>	Указывает, что по любому из опущенных параметров атрибута AT (ширина или высота) объект расширяется таким образом, чтобы занимать всю протяженность окна.
<b>FILL</b>	Задаёт цвет области внутри объекта. Если этот атрибут опущен, то область внутри объекта не закрашивается.
<b>SCROLL</b>	Указывает, что объект подвергается скроллингу вместе с содержимым окна.
<b>HIDE</b>	Задаёт, что когда окно WINDOW или APPLICATION раскрывается в первый раз, сам объект не прорисовываются. Для того, чтобы отобразить группу и объекты в ней нужно использовать оператор UNHIDE.
<b>BEVEL</b>	Задаёт эффект объёмности границ объекта.

Объектом **PANEL** определяется область на экране, размеры и положение которой описываются атрибутом AT. На этот объект не может переключаться фокус и для него не генерируется никаких событий.

**Пример:**

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
PANEL,AT(10,100,20,20),USE(?P1),BEVEL(-2,2)
END
```

**PROMPT (объявить поле - подсказку)**

**PROMPT(текст) ,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,FONT( )]  
[,FULL] [,SCROLL][,HIDE] [,LEFT |] [,DROPID( )] [,TRN]  
| RIGHT | [,COLOR( )]  
| CENTER|**

**PROMPT** Объявляет поле - подсказку для следующего за ним в структуре WINDOW или TOOLBAR поля.

<i>текст</i>	Строковая константа, содержащая текст подсказки. Она может включать амперсанд, указывающий горячую клавишу для этой подсказки.
<b>AT</b>	Задает первоначальные размеры и расположение окна списка. Если этот атрибут опущен, то по умолчанию параметры выбираются библиотечной процедурой.
<b>CURSOR</b>	Задает форму, которую должен принимать курсор мыши, при попадании в этот объект. Если этот атрибут опущен, то используется значение атрибута CURSOR структуры WINDOW, если же и тот не указан, то курсор имеет форму, используемую в среде Windows по умолчанию.
<b>USE</b>	Метка соответствия поля, которая служит для ссылок на данное поле в исполняемых операторах.
<b>DISABLE</b>	Указывает, что когда в окно WINDOW (или APPLICATION) раскрывается впервые, данное поле выглядит затухеванным.
<b>FONT</b>	Задает шрифт для вывода текста в подсказке.
<b>FULL</b>	Указывает, что по любому из опущенных параметров атрибута AT (ширина или высота) подсказка расширяется таким образом, чтобы занимать всю протяженность окна.
<b>SCROLL</b>	Указывает, что поле подвергается скроллингу вместе с содержимым окна.
<b>TRN</b>	Задает, что объект выводится на “прозрачном” фоне.
<b>HIDE</b>	Задает, что когда окно WINDOW или APPLICATION раскрывается в первый раз, данное поле не прорисовывается. Для того, чтобы отобразить это поле нужно использовать оператор UNHIDE.
<b>LEFT</b>	Задает, что текст подсказки выравниваются влево.
<b>RIGHT</b>	Задает, что текст подсказки выравниваются право.
<b>CENTER</b>	Задает, что текст подсказки выравниваются по центру.
<b>DROPID</b>	Указывает, что данный объект может служить областью, в которой происходит вторая часть операции “перетащить и отпустить” - отпускание объекта.
<b>COLOR</b>	Задает цвет фона объекта.

Оператор **PROMPT** объявляет поле - подсказку для следующего за ним в структуре WINDOW или TOOLBAR поля. Положение и размеры текста подсказки задаются атрибутом AT.

Текст подсказки может включать амперсанд, указывающий, что следующая непосредственно за ним буква, является горячей клавишей. По умолчанию “горячая” буква высвечивается подчеркнутой, чтобы указать ее специальное назначение. Эта горячая буква будучи нажатой в сочетании с клавишей Alt переключает фокус на следующее за ним в структуре WINDOW или TOOLBAR поле, на которое возможно его переключение.

Выключение или неотображение (с помощью атрибутов DISABLE или HIDE) следующего за подсказкой не делает того же автоматически и с полем - подсказкой; его

надо выключать или скрывать явно, в противном случае подсказка будет относиться к активному объекту, следующему в структуре WINDOW или TOOLBAR за скрытым или выключенным полем. Это позволяет поместить в окно подсказку, которая будет относиться к любому из нескольких объектов (если только один будет активен в любой момент времени). Если следующий активный объект является кнопкой (BUTTON), то, когда пользователь нажимает горячую клавишу для подсказки, кнопка нажимается.

Для того, чтобы включить амперсанд как часть текста подсказки, в строку поместите два амперсанда подряд, а высвечиваться будет только один.

На этот объект не может переключаться фокус.

### Генерируемые события:

EVENT:Drop                      Успешная операция “перетащить и отпустить” в данный объект.

### Пример:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           PROMPT('Enter Data:'),AT(10,100,20,20),USE(?P1),CURSOR(CURSOR:Wait)
           ENTRY(@S8),AT(100,100,20,20),USE(E1)
           PROMPT('EnterMore Data:'),AT(10,200,20,20),USE(?P2),CURSOR(CURSOR:Wait)
           ENTRY(@S8),AT(100,200,20,20),USE(E2)
           ENTRY(@D1),AT(100,200,20,20),USE(E3)
           END
           CODE
           OPEN(MDIChild)
           IF некое_условие
               HIDE(?E2)                      !Подсказка будет относиться к полю E3
           ELSE
               HIDE(?E3)                      !Подсказка будет относиться к полю E2
           END
```

## PROGRESS (объявить индикатор степени выполнения)

**PROGRESS, AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,FULL] [,SCROLL]  
[,HIDE] [,DROPID( )] [,RANGE( )]**

**PROGRESS**                      Помещает в окно или панель инструментов объект, в котором отображается текущая степень выполнения некоего длительного действия.

**AT**                                  Задает первоначальные размеры и местоположение объекта. Если этот параметр опущен, то значения по умолчанию выбираются в библиотеке поддержки.

**CURSOR**

Указывает форму курсора мыши, в которой тот отображается, когда он находится в пределах данного экранного объекта. Если этот атрибут опущен, то используется атрибут **CURSOR** структуры **WINDOW**, если он не указан и там, то используется курсор по умолчанию.

**USE**

Имя переменной, содержащей значение степени выполнения процесса или метка соответствия, которая служит для обращения к данному объекту в исполняемых операторах.

**DISABLE**

Указывает, что при первоначальном открытии окна данный объект отображается затухеванным.

**FULL**

Указывает, что для любого пропущенного параметра атрибута **AT** объект распространяется по всей протяженности окна.

**SCROLL**

Задаст, что объект прокручивается вместе с окном.

**HIDE**

Указывает, что при первом открытии структуры **WINDOW** или **APPLICATION** объект не высвечивается. Для того, чтобы его отобразить, нужно использовать оператор **UNHIDE**.

**DROPID**

Указывает, что данный объект может служить областью, в которой происходит вторая часть операции “перетащить и отпустить” - отпускание кнопки мыши.

**RANGE**

Задаст диапазон значений диаграммы степени выполнения процесса (максимум до 32767). По умолчанию диапазон значений от 0 до 100.

Экранный объект **PROGRESS** высвечивает динамическую линейную диаграмму. Обычно она отображает текущий процент выполнения некоего длительного действия.

Если атрибутом **USE** указана переменная, то при изменении значения этой переменной автоматически обновляется линейка диаграммы. Если же **USE**-переменная представляет собой метку соответствия, то присваивая свойству **PROP:progress** (необъявляемое свойство - смотри раздел **Необъявляемые свойства**) значение, лежащее в определенном атрибутом **RANGE** диапазоне, можно непосредственно обновлять индикатор.

Этот объект не может принимать фокус ввода.

**Генерируемые события:**

**EVENT:Drop** Успешная операция “перетащить и отпустить” в данный объект.

**Пример**

```
BackgroundProcess      PROCEDURE      !Фоновый процесс пакетной
обработки
ProgressVariable      LONG
Win      WINDOW('Batch Processing...'),AT(,400,400),TIMER(1),MDI,CENTER
      PROGRESS,AT(100,100,200,20),USE(ProgressVariable),RANGE(0,200)
      PROGRESS,AT(100,140,200,20),USE(?ProgressBar),RANGE(0,200)
      BUTTON('Cancel'),AT(190,300,20,20),STD(STD:Close)
END
```

```

CODE
OPEN(Win)
OPEN(File)
?ProgressVariable{PROP:rangehigh} = RECORDS(File)
?ProgressBar{PROP:rangehigh} = RECORDS(File)
SET(File)                                !Установить пакетную обработку
ACCEPT
CASE EVENT()
OF EVENT:CloseWindow
    BREAK
OF EVENT:Timer                            !Обрабатывать записи, когда таймерные события
    позволяют
    ProgressVariable += 3                !Автоматическое обновление 1-го индикатора
LOOP 3 TIMES
NEXT(File)
IF ERRORCODE() THEN BREAK.
    ?ProgressBar{PROP:progress} += 1    !Ручное обновление
2-го индикатора
    !выполнить некоторые операции по пакетной обработке
...
CLOSE(File)
    
```

## RADIO (объявить кнопку с зависимой фиксацией)

```

RADIO(текст) ,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,KEY( )]
    [,MSG( )] [,HLP( )] [,SKIP] [,FONT( )] [,ICON( )] [,FULL] [,SCROLL]
    [,HIDE] [,ALRT( )] [,DROPID( )] [,LEFT      ] [,TIP( )] [,VALUE( )]
    [,RIGHT      ] [,TRN] [,COLOR( )]
    
```

**RADIO** объявляет кнопку с зависимой фиксацией в структуре WINDOW или TOOLBAR.

*текст* Строковая константа, содержащая текст, который должен выводиться для кнопки с зависимой фиксацией. Константа может содержать амперсанд (&), обозначающий горячую клавишу для кнопки.

**AT** Задаёт первоначальные размеры и расположение кнопки. Если этот атрибут опущен, то по умолчанию параметры выбираются библиотечной процедурой.

**CURSOR** Задаёт форму, которую должен принимать курсор мыши, при попадании в пределы кнопки. Если этот атрибут опущен, то используется значение атрибута CURSOR структуры WINDOW, если же и тот не указан, то курсор имеет форму, используемую в среде Windows по умолчанию.

**USE** Метка соответствия поля, которая служит для ссылок на данное поле в исполняемых операторах.

**DISABLE** Указывает, что когда в окно WINDOW (или APPLICATION)

	раскрывается впервые, данная кнопка выглядит затушеванной.
<b>KEY</b>	Задаёт целочисленную константу или мнемонический код клавиши (или комбинации клавиш), с помощью которой немедленно выбирается данная кнопка.
<b>MSG</b>	Задаёт строковую константу, содержащую текст, который по умолчанию следует вывести в строке состояния, когда на данную кнопку переключается фокус.
<b>HLP</b>	Задаёт строковую константу, содержащую идентификатор системы справки для данной кнопки.
<b>SKIP</b>	Указывает, что на данную кнопку фокус ввода не переключается и доступ к ней может быть осуществлён только посредством мыши или клавиши ускоренного доступа.
<b>FONT</b>	Задаёт шрифт для вывода текста для данной кнопки.
<b>ICON</b>	Задаёт имя файла .ICO или стандартной пиктограммы, которая должна высвечиваться на поверхности нажимной кнопки.
<b>FULL</b>	Указывает, что по любому из опущенных параметров атрибута AT (ширина или высота) кнопка расширяется таким образом, чтобы занимать всю протяжённость окна.
<b>SCROLL</b>	Указывает, что кнопка подвергается скроллингу вместе с содержимым окна.
<b>HIDE</b>	Задаёт, что когда окно WINDOW или APPLICATION раскрывается в первый раз, сам объект - кнопка не прорисовываются. Для того, чтобы отобразить кнопку нужно использовать оператор UNHIDE.
<b>LEFT</b>	Указывает, что текст выводится слева от кнопки.
<b>RIGHT</b>	Указывает, что текст выводится справа от кнопки.
<b>ALRT</b>	Задаёт активные горячие клавиши для данной кнопки.
<b>DROPID</b>	Указывает, что данная кнопка может служить областью в которой происходит вторая часть операции “потащить и отпустить” - отпускание объекта.
<b>TIP</b>	Задаёт текст, который выводится как “возникающая” подсказка, когда курсор мыши задерживается на экранном объекте.
<b>TRN</b>	Задаёт, что объект выводится на “прозрачном” фоне.
<b>COLOR</b>	Задаёт цвет фона объекта
<b>VALUE</b>	Указывает, какое значение принимает USE-переменная структуры OPTION, когда данная кнопка радио выбирается пользователем.

Оператор **RADIO** объявляет кнопку с зависимой фиксацией в структуре WINDOW или TOOLBAR, место и размеры которой заданы его атрибутом AT. Объект RADIO может помещаться только в структуре OPTION. Если не использован атрибут VALUE, то текст, соответствующий выбранной пользователем кнопке, (без обозначавшего горячую клавишу амперсанда) помещается в USE-переменную структуры

OPTION.

Поле RADIO с атрибутом ICON изображается в виде фиксирующейся кнопки с

пиктограммой на поверхности. Когда кнопка выглядит отжатой, она выключена, когда кнопка выглядит нажатой, то она включена, а USE-переменная принимает значение параметра текст выбранной кнопки (если не указан атрибут VALUE).

### Генерируемые события:

EVENT:Selected	На кнопку переключен фокус ввода.
EVENT:Accepted	Кнопка выбрана пользователем.
EVENT:PreAlertKey	Пользователь нажал заданную атрибутом ALRT горячую клавишу.
EVENT:AlertKey	Пользователь нажал заданную атрибутом ALRT горячую клавишу.
EVENT:Drop	Кнопка мыши отпущена над объектом - целью операции "поташить и отпустить"

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
  OPTION('Option 1'),USE(OptVar1)
    RADIO('Radio 1'),AT(0,0,20,20),USE(?R1),KEY(F10Key)
    RADIO('Radio 2'),AT(20,0,20,20),USE(?R2),MSG('Radio 2')
  END
  OPTION('Option 2'),USE(OptVar2)
    RADIO('Radio 3'),AT(40,0,20,20),USE(?R3),FONT('Arial',12)
    RADIO('Radio 4'),AT(60,0,20,20),USE(?R4),CURSOR(CURSOR:Wait)
  END
  OPTION('Option 3'),USE(OptVar3)
    RADIO('Radio 5'),AT(80,0,20,20),USE(?R5),HLP('Radio5Help')
    RADIO('Radio 6'),AT(100,0,20,20),USE(?R6)
  END
  OPTION('Option 4'),USE(OptVar4)
    RADIO('Radio 7'),AT(120,0,20,20),USE(?R7),ICON('Radio1.ICO')
    RADIO('Radio 8'),AT(140,0,20,20),USE(?R8),ICON('Radio2.ICO')
  END
  OPTION('Option 5'),USE(OptVar5)
    RADIO('Radio 9'),AT(100,20,20,20),USE(?R9),LEFT
    RADIO('Radio 10'),AT(120,20,20,20),USE(?R10),LEFT
  END
  OPTION('Option 6'),USE(OptVar6),SCROLL
    RADIO('Radio 11'),AT(200,0,20,20),USE(?R11),SCROLL
    RADIO('Radio 12'),AT(220,0,20,20),USE(?R12),SCROLL
  END
END
```

**Смотри также:** OPTION

**REGION (объявить объект - область в окне)**

```

REGION,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,FILL] [,COLOR( )]
[,IMM]
[,FULL][,SCROLL] [,HIDE] [,DRAGID( )] [,DROPID( )] [,TRN] [,BEVEL( )]

```

<b>REGION</b>	Объявляет область в структуре WINDOW или TOOLBAR.
<b>AT</b>	Задаёт первоначальные размеры и расположение области. Если этот атрибут опущен, то по умолчанию параметры выбираются библиотечной процедурой.
<b>CURSOR</b>	Задаёт форму, которую должен принимать курсор мыши, при попадании в пределы области. Если этот атрибут опущен, то используется значение атрибута CURSOR структуры WINDOW, если же и тот не указан, то курсор имеет форму, используемую в среде Windows по умолчанию.
<b>USE</b>	Метка соответствия поля, которая служит для ссылок на данное поле в исполняемых операторах.
<b>DISABLE</b>	Указывает, что когда в окно WINDOW (или APPLICATION) раскрывается впервые, данная область выглядит затуманенной.
<b>FILL</b>	Задаёт числовые величины компонент красного, зеленого и синего цвета, которые составляют цвет закрашивания области.
<b>COLOR</b>	Указывает цвет рамки области. Если атрибут опущен, то рамки нет вообще.
<b>FULL</b>	Указывает, что по любому из опущенных параметров атрибута AT (ширина или высота) область расширяется таким образом, чтобы занимать всю протяженность окна.
<b>TRN</b>	Задаёт, что объект выводится на “прозрачном” фоне.
<b>SCROLL</b>	Указывает, что кнопка подвергается скроллингу вместе с содержимым окна.
<b>HIDE</b>	Задаёт, что когда окно WINDOW или APPLICATION раскрывается в первый раз, область не прорисовывается. Для того, чтобы отобразить область нужно использовать оператор UNHIDE.
<b>DRAGID</b>	Указывает, что данный объект может служить областью - источником данных или объектов для операции “потянуть и отпустить”.
<b>DROPID</b>	Указывает, что в данной области может происходить вторая часть операции “потянуть и отпустить” - отпуская объект.
<b>BEVEL</b>	Задаёт эффект объёмности границ объекта.

Оператор **REGION** определяет область на экране, место и размеры которой задаются его атрибутом AT. Обычно, причиной описания области является необходимость обнаружения положения курсора мыши в данном месте. Когда событие попадания курсора в заданную область происходит, для определения точных координат курсора можно использовать функции MOUSEX и MOUSEY. Использование атрибута IMM вызывает



включение некоторого избыточного программного кода и дополнительные затраты времени во время выполнения программы, так что этот атрибут следует использовать только когда это действительно необходимо. На область определенную оператором REGION фокус ввода переключаться не может.

Область, описанная с атрибутом DRAGID может служить для операции “поташить и отпустить” источником, обеспечивающим данные, которые должны переноситься или копироваться в другой объект. В области с атрибутом DROPID может служить для операции “поташить и отпустить” “целью” - областью,

принимающей данные. Эти атрибуты функционируют совместно, чтобы задать “ярлыки” для операции “поташить и отпустить”, которые определяет допустимую цель операции. Для выполнения переноса данных, наряду с процедурой SETDROPID используются функции DRAGID() и DROPID(). Поскольку оператором REGION можно описать область поверх любого другого объекта, можно написать программу обмена данными операцией “поташить и отпустить” между любыми двумя экранными объектами. Просто определите области операторами REGION, чтобы управлять необходимыми операциями “поташить и отпустить”.

### Генерируемые события:

EVENT:Accepted                      Пользователь щелкнул мышью, указывая в пределы области.

Для области с атрибутом IMM дополнительно генерируется:

EVENT:MouseIn	Курсор мыши вошел в пределы области.
EVENT:MouseOut	Курсор мыши вышел за пределы области.
EVENT:MouseMove	Курсор мыши переместился в пределах области.

Для области с атрибутом DRAGID дополнительно генерируется:

EVENT:Dragging	Курсор мыши находится над объектом - потенциальной целью.
EVENT:Drag	Кнопка мыши отпущена над объектом - потенциальной целью.

Для области с атрибутом DROPID дополнительно генерируется:

EVENT:Drop	Кнопка мыши отпущена над объектом - потенциальной целью.
------------	--

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
    REGION,AT(10,100,20,20),USE(?R1),BEVEL(-2,2)
    REGION,AT(100,100,20,20),USE(?R2),CURSOR(CURSOR:Wait)
    REGION,AT(10,200,20,20),USE(?R3),IMM
    REGION,AT(100,200,20,20),USE(?R4),COLOR(COLOR:ACTIVEBORDER)
    REGION,AT(10,300,20,20),USE(?R4),FILL(COLOR:ACTIVEBORDER)
END
```

**SHEET (объявить группу листов-закладок)**

```

SHEET ,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,KEY( )] [,FULL]
[,SCROLL] [,HIDE] [,FONT( )] [,DROPID( )] [,WIZARD] [,SPREAD]
[,SKIP] [,HSCROLL][[,JOIN][,NOSHEET][,COLOR( )]
[,UP ] [,DOWN ] [,LEFT  ]
                |RIGHT
                |ABOVE
                |BELOW
лист
END

```

- SHEET**                      Объявляет группу объектов типа TAB.
- AT**                              Задает первоначальные размеры и расположение объекта. Если этот атрибут опущен, то значение по умолчанию выбирается библиотечной процедурой.
- CURSOR**                      Указывает форму курсора мыши, в которой тот отображается, когда он находится в пределах данного экранного объекта. Если этот атрибут опущен, то используется атрибут CURSOR структуры WINDOW, если он не указан и там, то используется курсор принятый по умолчанию в среде Windows.
- USE**                              Метка переменной, в которую заносится выбор пользователя. Если это строковая переменная, то в нее заносится значение строки-параметра структуры TAB (исключая амперсанд, обозначающий клавишу ускоренного доступа). Если же это числовая переменная, то в нее заносится номер выбранного в данный момент пользователем листа (число, возвращаемое функцией CHOICE()).
- DISABLE**                      Указывает, что при первом раскрытии окна WINDOW или APPLICATION набор листов выводится “затушеванным”.
- KEY**                              Задает целочисленную константу или мнемоническую метку соответствия клавиши, с помощью которой фокус немедленно переключается на выбранный лист в данном наборе.
- FULL**                              Указывает, что набор листов занимает всю протяженность окна по любому из опущенных в атрибуте AT измерений - длине или высоте.
- SCROLL**                              Указывает, что набор листов подвергается скроллингу вместе с окном.
- HIDE**                              Задает, что когда окно WINDOW или APPLICATION раскрывается в первый раз, данное поле не прорисовывается. Для того, чтобы отобразить это поле нужно использовать оператор UNHIDE.
- FONT**                              Задает шрифт для вывода текста в данном объекте и шрифт, используемый по умолчанию для отдельных листов в данном наборе.
- DROPID**                              Указывает, что данная группа листов может служить областью, в которой происходит вторая часть операции “потащить и отпустить” - отпускание объекта.
- WIZARD**                              Указывает что все листы данного набора сразу не выводятся.

Управление переходом пользователя от листа к листу происходит программно (обычно с помощью кнопок “Next” и “Previous”).

**SPREAD**

Указывает, что листы располагаются через равный промежуток друг от друга.

**SKIP**

Указывает, что фокус не может переключаться с листа на лист в этом наборе с помощью клавиши Tab, а переключается только посредством мыши или клавиши ускоренного доступа.

**HSCROLL**

Указывает, что листы выводятся в один ряд, а не в несколько, не взирая на то, сколько листов всего. Для перелистывания на каждом краю листа присоединяются кнопки скроллинга вправо и влево (или вверх и вниз).

**JOIN**

Указывает, что листы выводятся в один ряд, а не в несколько, не взирая на то, сколько листов всего. Для перелистывания на правом (или нижнем) краю листа присоединяются кнопки скроллинга вправо и влево (или вверх и вниз).

**NOSHEET**

Указывает, что листы выводятся без зрительного образа листа.

**COLOR**

Задает цвет фона объекта.

**UP**

Задает, что значение параметра текст на листах выводится вертикально для чтения снизу вверх.

**DOWN**

Задает, что значение параметра текст на листах выводится вертикально для чтения сверху вниз

**LEFT**

Указывает, что листы выводятся один за другим со сдвигом влево.

**RIGHT**

Указывает, что листы выводятся один за другим со сдвигом вправо.

**ABOVE**

Указывает, что листы выводятся один за другим со сдвигом вверх (способ используемый по умолчанию).

**BELOW**

Указывает, что листы выводятся один за другим со сдвигом вниз.

*лист*

Несколько объявлений объектов типа лист-закладка

Управляющий объект **SHEET** объявляет набор (группу) листов-закладок, которые часто используются для предоставления пользователю нескольких “страниц” с объектами на одном экране. Несколько объектов TAB в структуре SHEET, объявляют “страницы”, выводимые пользователю.

Переключение фокуса между листами в структуре SHEET означает, что воздействовать можно только на объекты данного отдельного листа. Это значит, что события, генерируемые при переключении фокуса внутри структуры SHEET относятся к объектам текущего листа, а не к структуре SHEET, в которой они содержатся.

В строковую переменную, использованную в качестве параметра атрибута USE, заносится текст - параметр выбранной пользователем структуры Tab, а функция CHOICE(?Option) возвращает номер выбранной структуры Tab. Если USE-переменной структуры SHEET является числовая переменная, то она принимает номер выбранной пользователем структуры Tab (то же самое значение возвращает функция CHOICE).

**Генерируемые события:**

EVENT:Selected	На один из листов набора переключен фокус.
EVENT:Accepted	Один из листов набора выбран пользователем.
EVENT:PreAlertKey	Пользователь нажал заданную атрибутом ALRT горячую клавишу
EVENT:AlertKey	Пользователь нажал заданную атрибутом ALRT горячую клавишу
EVENT:Drop	Успешная операция “перетащить и отпустить” в данный объект.
EVENT:TabChanging	Фокус переключается на другой лист.

**Пример:**

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
SHEET,AT(0,0,320,175),USE(SelectedTab)
TAB('Tab One'),USE(?TabOne)
OPTION('Option 1'),USE(OptVar1),KEY(F10Key),HLP('Option1Help')
RADIO('Radio 1'),AT(20,0,20,20),USE(?R1)
RADIO('Radio 2'),AT(40,0,20,20),USE(?R2)
END
OPTION('Option 2'),USE(OptVar2),MSG('Option 2')
RADIO('Radio 3'),AT(60,0,20,20),USE(?R3)
RADIO('Radio 4'),AT(80,0,20,20),USE(?R4)
END
PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P1)
ENTRY(@S8),AT(100,140,32,20),USE(E1)
PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P2)
ENTRY(@S8),AT(100,240,32,20),USE(E2)
END
TAB('Tab Two'),USE(?TabTwo)
OPTION('Option 3'),USE(OptVar3)
RADIO('Radio 1'),AT(20,0,20,20),USE(?R5)
RADIO('Radio 2'),AT(40,0,20,20),USE(?R6)
END
OPTION('Option 4'),USE(OptVar4)
RADIO('Radio 3'),AT(60,0,20,20),USE(?R7)
RADIO('Radio 4'),AT(80,0,20,20),USE(?R8)
END
PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P3)
ENTRY(@S8),AT(100,140,32,20),USE(E3)
PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P4)
ENTRY(@S8),AT(100,240,32,20),USE(E4)
END
END
```

```
BUTTON('Ok'),AT(100,180,20,20),USE(?Ok)
BUTTON('Cancel'),AT(200,180,20,20),USE(?Cancel)
END
```

Смотри также: TAB

## SPIN (объявить вращающийся список)

```
SPIN(шаблон),AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,KEY( )] [,MSG( )]
[,HLP( )] [,SKIP] [,FONT( )] [,FULL] [,SCROLL] [,ALRT( )] [,HIDE]
[,READONLY] [,REQ] [,IMM] [DROPID( )] [,TIP( )] [,TRN] [,COLOR( )]
[, |LEFT| ] [, |INS| ], |RANGE( )| [,STEP] |[,UPR|
|RIGHT| |OVR| |FROM( )| |CAP|
|CENTER|
|DECIMAL|
```

- SPIN** Помещает в окно или панель инструментов “вращающийся список элементов данных.
- шаблон* Шаблон для отображения данных, который указывает формат данных, выводимых в списке.
- AT** Задаёт первоначальные размеры и расположение поля. Если этот атрибут опущен, то значения выбираются библиотечной процедурой.
- CURSOR** Задаёт форму, которую должен принимать курсор мыши, при попадании в это поле. Если этот атрибут опущен, то используется значение атрибута CURSOR структуры WINDOW, если же и тот не указан, то курсор имеет форму, используемую в среде Windows по умолчанию.
- USE** Метка соответствия, которая служит для того, чтобы указывать на это поле в исполняемых операторах или имя переменной, которая получает значение, введенное в данное поле.
- DISABLE** Указывает, что когда в окно WINDOW (или APPLICATION) раскрывается впервые, данное поле выглядит затухеванным.
- KEY** Задаёт целочисленную константу или мнемонический код клавиши (или комбинации клавиш), с помощью которой фокус немедленно переключается на это поле.
- MSG** Задаёт строковую константу, содержащую текст, который следует вывести в строке состояния, когда на данное поле переключен фокус.
- HLP** Задаёт строковую константу, содержащую идентификатор системы справки для данного поля.
- SKIP** Указывает, что для ввода данных в это поле фокус ввода переключается только посредством мыши или клавиши ускоренного доступа, и не задерживается на этом поле после завершения ввода данных.
- FONT** Задаёт шрифт для вывода данных в этом поле.
- FULL** Указывает, что по любому из опущенных параметров атрибута AT

	(ширина или высота) поле расширяется таким образом, чтобы занимать всю протяженность окна.
<b>SCROLL</b>	Указывает, что поле подвергается скроллингу вместе с содержимым окна.
<b>ALRT</b>	Задаёт активные горячие клавиши для данного поля.
<b>HIDE</b>	Задаёт, что когда окно <b>WINDOW</b> или <b>APPLICATION</b> раскрывается в первый раз, данное поле не прорисовывается. Для того, чтобы отобразить это поле нужно использовать оператор <b>UNHIDE</b> .
<b>READONLY</b>	Указывает, что в поле нельзя ввести данные.
<b>REQ</b>	Задаёт, что поле не может оставаться пустым или нулевым.
<b>IMM</b>	Указывает, что как только пользователь нажмет любую клавишу, немедленно происходит генерирование события.
<b>TIP</b>	Задаёт текст, который выводится как “возникающая” подсказка, когда курсор мыши задерживается на экранном объекте.
<b>TRN</b>	Задаёт, что объект выводится на “прозрачном” фоне.
<b>DROPID</b>	Указывает, что данное управляющее поле может служить областью в которой происходит вторая часть операции “потащить и отпустить” - отпуская объект.
<b>LEFT</b>	Задаёт, что данные в области, заданной атрибутом <b>AT</b> , выравниваются влево.
<b>RIGHT</b>	Задаёт, что данные в области, заданной атрибутом <b>AT</b> , выравниваются право.
<b>CENTER</b>	Задаёт, что данные в области, заданной атрибутом <b>AT</b> , выравниваются по центру.
<b>DECIMAL</b>	Задаёт, что данные в области, заданной атрибутом <b>AT</b> , выравниваются по десятичной точке.
<b>INS/OVR</b>	Задаёт при вводе данных режим вставки или замещения (допустимо только в окнах, имеющих атрибут <b>MASK</b> ).
<b>RANGE</b>	Задаёт диапазон допустимых значений, которые пользователь может выбирать.
<b>STEP</b>	Задаёт величину приращения/уменьшения значения в диапазоне, заданном атрибутом <b>RANGE</b> . По умолчанию атрибут <b>STEP</b> равен 1.0 .
<b>FROM</b>	Указывает источник высвечиваемых пользователю вариантов выбора.
<b>UPR/CAP</b>	Указывает, что данные вводятся прописными буквами или по типу имен собственных (Первая Буква В Каждом Слове Прописная).
<b>HSCROLL</b>	Задаёт кнопки вращающегося списка располагаются рядом, указывая налево и направо.
<b>VSCROLL</b>	Задаёт кнопки вращающегося списка располагаются одна над другой, указывая налево и направо.
<b>HVSCROLL</b>	Задаёт кнопки вращающегося списка располагаются рядом, указывая вверх и вниз.
<b>COLOR</b>	Задаёт цвет фона по умолчанию и цвета переднего плана и фона для активизированного объекта.

Объект **SPIN** представляет собой “вращающийся” список данных в окне или на панели

инструментов. Местоположение и размеры списка заданы атрибутом AT. Вращающийся список высвечивает только текущее значение для выбора и пару кнопок справа от него, позволяющих пользователю прокручивать возможные значения (подобно колесу игрового автомата).

Если поле SPIN предлагает пользователю равномерную шкалу числовых значений для выбора, то атрибут RANGE указывает допустимый диапазон значений, из которого пользователь может выбирать. В этом случае совместно с атрибутом RANGE используется атрибут STEP, который служит для того, чтобы задать величину приращения/уменьшения текущего значения. Атрибут FROM задает для объекта SPIN список возможных значений из очереди в памяти или строковой переменной. Используя атрибут FROM можно предоставить пользователю в поле SPIN любые варианты для выбора.

Пользователь может выбрать элемент из списка или ввести желаемое значение, так что это поле может также функционировать как поле для ввода.

### Генерируемые события:

EVENT:Selected	На данное поле переключен фокус.
EVENT:Accepted	Пользователь выбрал значение в данном поле.
EVENT:Rejected	Пользователь ввел несоответствующее шаблону значение.
EVENT:NewSelection	Пользователь изменил значение высвечиваемое в данном поле.
EVENT:PreAlertKey	Пользователь нажал заданную атрибутом ALRT горячую клавишу.
EVENT:AlertKey	Пользователь нажал заданную атрибутом ALRT горячую клавишу.
EVENT:Drop	Кнопка мыши отпущена над объектом - целью операции "потащить и отпустить"

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
    SPIN(@S8),AT(0,0,20,20),USE(SpinVar1),FROM(Que)
    SPIN(@N3),AT(20,0,20,20),USE(SpinVar2),RANGE(1,999),KEY(F10Key)
    SPIN(@N3),AT(40,0,20,20),USE(SpinVar3),RANGE(5,995),STEP(5)
    SPIN(@S8),AT(60,0,20,20),USE(SpinVar4),FROM(Que),HLP('Check4Help')
    SPIN(@S8),AT(80,0,20,20),USE(SpinVar5),FROM(Que),MSG('Button 3')
    SPIN(@S8),AT(100,0,20,20),USE(SpinVar6),FROM(Que),FONT('Arial',12)
    SPIN(@S8),AT(120,0,20,20),USE(SpinVar7),FROM(Que),DROP
    SPIN(@S8),AT(140,0,20,20),USE(SpinVar8),FROM(Que),HVSCROLL
    SPIN(@S8),AT(160,0,20,20),USE(SpinVar9),FROM(Que),IMM
    SPIN(@S8),AT(180,0,20,20),USE(SpinVar10),FROM(Que),CURSOR(CURSOR:Wait)
    SPIN(@S8),AT(200,0,20,20),USE(SpinVar11),FROM(Que),ALRT(F10Key)
    SPIN(@S8),AT(220,0,20,20),USE(SpinVar12),FROM('Mr|Mrs|Ms'),LEFT
```

```

SPIN(@S8),AT(240,0,20,20),USE(SpinVar13),FROM(Que),RIGHT
SPIN(@S8),AT(260,0,20,20),USE(SpinVar14),FROM(Que),CENTER
SPIN(@N8.2),AT(280,0,20,20),USE(SpinVar15),FROM(Que),DECIMAL
END

```

## STRING (объявить экранное строковое поле)

```

STRING(текст) ,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,FONT( )]
[,FULL] [,SCROLL] [,HIDE] [,TRN] [, |LEFT|] [,DROPID( )]
[,COLOR( )] [,ANGLE( )] |RIGHT|
|CENTER|
|DECIMAL|

```

<b>STRING</b> <i>текст</i>	Помещает в окно или панель инструментов заданный текст. Строковая константа, содержащая подлежащий выводу текст, или шаблон для форматирования значения переменной, указанной атрибутом USE.
<b>AT</b>	Задаёт первоначальные размеры и расположение поля. Если этот атрибут опущен, то значения выбираются библиотечной процедурой.
<b>CURSOR</b>	Задаёт форму, которую должен принимать курсор мыши, при попадании в это поле. Если этот атрибут опущен, то используется значение атрибута CURSOR структуры WINDOW, если же и тот не указан, то курсор имеет форму, используемую в среде Windows по умолчанию.
<b>USE</b>	Метка соответствия, которая служит для того, чтобы указывать на это поле в исполняемых операторах или имя переменной, чье содержимое высвечивается в формате, указанном шаблоном, объявленным вместо текста параметром текст .
<b>DISABLE</b>	Указывает, что когда в окно WINDOW (или APPLICATION) раскрывается впервые, данное поле выглядит затухеванным.
<b>FONT</b>	Задаёт шрифт для вывода текста в этом поле.
<b>FULL</b>	Указывает, что по любому из опущенных параметров атрибута AT (ширина или высота) поле расширяется таким образом, чтобы занимать всю протяженность окна.
<b>SCROLL</b>	Указывает, что поле подвергается скроллингу вместе с содержимым окна.
<b>HIDE</b>	Задаёт, что когда окно WINDOW или APPLICATION раскрывается в первый раз, данное поле не прорисовывается. Для того, чтобы отобразить это поле нужно использовать оператор UNHIDE.
<b>TRN</b>	Указывает, что текст или значение USE-переменной высвечиваются прозрачно на предыдущем фоне.
<b>DROPID</b>	Указывает, что данное окно может служить областью, в которой происходит вторая часть операции “перетащить и отпустить” - отпускание объекта.
<b>LEFT</b>	Задаёт, что текст в области, заданной атрибутом AT, выравнивается влево.
<b>RIGHT</b>	Задаёт, что текст в области, заданной атрибутом AT, выравнивается право.
<b>CENTER</b>	Задаёт, что текст в области, заданной атрибутом AT, выравнивается по центру.



<b>DECIMAL</b>	Задаёт, что текст в области, заданной атрибутом AT, выравнивается по десятичной точке.
<b>COLOR</b>	Задаёт цвет фона объекта.
<b>ANGLE</b>	Задаёт вывод текста под заданным углом, отсчитываемым против хода часовой стрелки от горизонтального направления.

Оператор **STRING** помещает в окно или панель инструментов заданный текст. Местоположение и размеры текста задаются атрибутом AT.

Если параметр текст представляет собой шаблон форматирования вместо текстовой константы, то по этому шаблону форматируется значение переменной, указанной атрибутом USE. Таким образом поле STRING с атрибутом USE становится выводным полем для переменной. Данные, отображаемые в поле STRING, автоматически обновляются при каждом выполнении цикла ACCEPT, независимо от того, имеется атрибут AUTO или нет.

Между амперсандами, используемыми в поле STRING и в поле PROMPT, существует разница. Амперсанд в поле STRING высвечивается как часть текста, в то время как амперсанд в поле PROMPT определяет горячую клавишу для поля подсказки.

В поле STRING с атрибутом TRN текст высвечивается прозрачно, на предыдущем фоне. Это значит, что на экран выводятся только пикселы, составляющие каждый символ. Это позволяет поместить строку поверх поля IMAGE, не разрушая фоновое изображение.

На это поле не переключается фокус.

#### Генерируемые события:

EVENT:Drop Успешная операция “перетащить и отпустить” в данный объект.

#### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
          STRING('String Constant'),AT(10,0,20,20),USE(?S1)
          STRING(@S30),AT(10,20,20,20),USE(StringVar1)
          STRING(@S30),AT(10,20,20,20),USE(StringVar2),CURSOR(CURSOR:Wait)
          STRING(@S30),AT(10,20,20,20),USE(StringVar3),FONT('Arial',12)
END
```

## TAB (объявить лист в структуре SHEET)

```
TAB( текст ) [,CURSOR( )] [,USE( )] [,KEY( )] [,MSG( )] [,HLP( )] [,REQ]
      [DROPID( )] [,TIP( )] [,COLOR( )]
      управляющие поля
END
```

<b>TAB</b>	Объявляет группу управляющих полей, которые составляют один из нескольких листов - “страниц”, содержащихся в структуре SHEET.
<i>текст</i>	Строковая константа, содержащая выводимый для листа текст.
<b>CURSOR</b>	Указывает форму курсора мыши, в которой тот отображается, когда он находится в пределах данного экранного объекта. Если этот атрибут опущен, то используется атрибут CURSOR структуры WINDOW, если он не указан и там, то используется курсор по умолчанию.
<b>USE</b>	Задаёт метку соответствия, для осуществления ссылок в исполняемых операторах.
<b>KEY</b>	Задаёт целочисленную константу или мнемоническую метку соответствия клавиши, с помощью которой фокус немедленно переключается на данный лист.
<b>MSG</b>	Задаёт строковую константу, содержащую текст, который следует вывести в строке состояния, когда фокус переключен на любое управляющее поле на данном листе.
<b>HLP</b>	Задаёт строковую константу, содержащую идентификатор системы справки для любого управляющего поля на данном листе.
<b>REQ</b>	Задаёт, что когда выбирается другой лист, библиотека исполняющей системы автоматически проверяет, REQ все поля типа ENTRY в этой структуре TAB, имеющей атрибут REQ, содержат ненулевые и не пробельные данные.
<b>DROPID</b>	Указывает, что данное окно может служить областью, в которой происходит вторая часть операции “перетащить и отпустить” - отпускание объекта.
<b>TIP</b>	Задаёт текст, который выводится как “возникающая” подсказка, когда курсор мыши задерживается на экранном объекте.
<b>COLOR</b>	Задаёт цвет фона по умолчанию и цвета переднего плана и фона для активизированного объекта на листе.
<i>управляющие поля</i>	Несколько объявлений управляющих полей.

В структуре **TAB** объявляется группа полей, которые составляют один из нескольких листов, содержащихся в структуре SHEET. Несколько объектов TAB в структуре SHEET определяют несколько листов, выводимых на экран. В переменную - атрибут USE структуры SHEET заносится текст выбранной пользователем структуры TAB.

Переключение фокуса между листами в структуре SHEET означает, что воздействовать можно только на объекты данного отдельного листа. Это значит, что события, генерируемые

при переключении фокуса внутри структуры SHEET относятся к объектам текущего листа, а не к структуре SHEET, в которой они содержатся.

### Генерируемые события:

EVENT:Selected	На данный лист переключен фокус.
EVENT:Accepted	Данный лист набора выбран пользователем.
EVENT:Drop	Успешная операция “перетащить и отпустить” в данный объект.

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
SHEET,AT(0,0,320,175),USE(SelectedTab)
TAB('Tab One'),USE(?TabOne)
  OPTION('Option 1'),USE(OptVar1),KEY(F10Key),HLP('Option1Help')
  RADIO('Radio 1'),AT(20,0,20,20),USE(?R1)
  RADIO('Radio 2'),AT(40,0,20,20),USE(?R2)
END
  OPTION('Option 2'),USE(OptVar2),MSG('Option 2')
  RADIO('Radio 3'),AT(60,0,20,20),USE(?R3)
  RADIO('Radio 4'),AT(80,0,20,20),USE(?R4)
END
  PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P1)
  ENTRY(@S8),AT(100,140,32,20),USE(E1)
  PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P2)
  ENTRY(@S8),AT(100,240,32,20),USE(E2)
END
TAB('Tab Two'),USE(?TabTwo)
  OPTION('Option 3'),USE(OptVar3)
  RADIO('Radio 1'),AT(20,0,20,20),USE(?R5)
  RADIO('Radio 2'),AT(40,0,20,20),USE(?R6)
END
  OPTION('Option 4'),USE(OptVar4)
  RADIO('Radio 3'),AT(60,0,20,20),USE(?R7)
  RADIO('Radio 4'),AT(80,0,20,20),USE(?R8)
END
  PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P3)
  ENTRY(@S8),AT(100,140,32,20),USE(E3)
  PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P4)
  ENTRY(@S8),AT(100,240,32,20),USE(E4)
END
END
  BUTTON('Ok'),AT(100,180,20,20),USE(?Ok)
  BUTTON('Cancel'),AT(200,180,20,20),USE(?Cancel)
END
```

**Смотри также:** SHEET

**TEXT (объявить поле из нескольких строк для ввода данных)**

```

TEXT ,AT( ) [,CURSOR( )] [,USE( )] [,DISABLE] [,KEY( )] [,MSG( )] [,TRN]
[,HLP( )] [,SKIP] [,FONT( )] [,REQ] [,FULL] [,SCROLL] [,ALRT( )]
[,HIDE] [,READONLY] [,DROPID( )] [,UPR] [,TIP( )] [,COLOR( )]
[,SINGLE]
[, |INS      |] [, |HSCROLL |] [, |LEFT |]
  |OVR|      |VSCROLL |      |RIGHT |
                |HVSCROLL|      |CENTER|

```

<b>TEXT</b>	Помещает в окно или панель инструментов поле для ввода данных из нескольких строк.
<b>AT</b>	Задаёт первоначальные размеры и расположение поля. Если этот атрибут опущен, то значения выбираются библиотечной процедурой.
<b>CURSOR</b>	Задаёт форму, которую должен принимать курсор мыши, при попадании в это поле. Если этот атрибут опущен, то используется значение атрибута CURSOR структуры WINDOW, если же и тот не указан, то курсор имеет форму, используемую в среде Windows по умолчанию.
<b>USE</b>	Имя переменной, которая получает значение, введенное пользователем в данное поле.
<b>DISABLE</b>	Указывает, что когда в окно WINDOW (или APPLICATION) раскрывается впервые, данное поле выглядит затухеванным.
<b>KEY</b>	Задаёт целочисленную константу или мнемонический код клавиши (или комбинации клавиш), с помощью которой фокус немедленно переключается на это поле.
<b>MSG</b>	Задаёт строковую константу, содержащую текст, который следует вывести в строке состояния, когда на данное поле переключен фокус.
<b>HLP</b>	Задаёт строковую константу, содержащую идентификатор системы справки для данного поля.
<b>SKIP</b>	Указывает, что для ввода данных в это поле фокус ввода переключается только посредством мыши или клавиши ускоренного доступа, и не задерживается на этом поле после завершения ввода данных.
<b>FONT</b>	Задаёт шрифт для отображения данных в этом поле.
<b>REQ</b>	Задаёт, что поле не может оставаться пустым или нулевым.
<b>FULL</b>	Указывает, что по любому из опущенных параметров атрибута AT (ширина или высота) поле расширяется таким образом, чтобы занимать всю протяженность окна.
<b>SCROLL</b>	Указывает, что поле подвергается скроллингу вместе с содержимым окна.
<b>ALRT</b>	Задаёт активные горячие клавиши для данного поля.
<b>HIDE</b>	Задаёт, что когда окно WINDOW или APPLICATION раскрывается в первый раз, данное поле не прорисовывается. Для того, чтобы отобразить это поле нужно использовать оператор UNHIDE.

<b>READONLY</b>	Указывает, что в поле нельзя ввести данные.
<b>DROPID</b>	Указывает, что данное управляющее поле может служить областью в которой происходит вторая часть операции “потащить и отпустить” - отпускание объекта.
<b>TIP</b>	Задаёт текст, который выводится как “возникающая” подсказка, когда курсор мыши задерживается на экранном объекте.
<b>INS/OVR</b>	Задаёт при вводе данных режим вставки или замещения (допустимо только в окнах, имеющих атрибут MASK).
<b>UPR</b>	Указывает, что данные вводятся прописными буквами.
<b>HSCROLL</b>	Указывает, что когда какая-либо часть данных по горизонтали не помещается в этом поле, к нему автоматически присоединяется линейка горизонтального скроллинга.
<b>VSCROLL</b>	Указывает, что когда какая-либо часть данных по вертикали не помещается в этом поле, к нему автоматически присоединяется линейка вертикального скроллинга.
<b>HVSCROLL</b>	Указывает, что когда какая-либо часть данных не помещается в этом поле, к нему автоматически присоединяются линейки и вертикального и горизонтального скроллинга.
<b>LEFT</b>	Задаёт, что данные в области, заданной атрибутом AT, выравниваются влево.
<b>RIGHT</b>	Задаёт, что данные в области, заданной атрибутом AT, выравниваются право.
<b>CENTER</b>	Задаёт, что данные в области, заданной атрибутом AT, выравниваются по центру.
<b>COLOR</b>	Задаёт цвет фона объекта.
<b>SINGLE</b>	Указывает, что объект служит для ввода только одной строки. Это сделано специально для того, чтобы использовать поле TEXT вместо ENTRY для ввода данных на иврите и арабских языках.

Поле **TEXT** помещает в окно или панель инструментов для ввода данных из нескольких строк поле, местоположение и размеры которого заданы атрибутом AT. Когда пользователь завершил ввод данных и перешел к другому полю, введенные данные заносятся в переменную, указанную атрибутом USE. Введенные данные автоматически переносятся так, чтобы не разрывать слово.

### Генерируемые события:

EVENT:Selected	На данное поле переключен фокус ввода.
EVENT:Accepted	Пользователь завершил ввод данных в это поле.
EVENT:PreAlertKey	Пользователь нажал заданную атрибутом ALRT горячую клавишу.
EVENT:AlertKey	Пользователь нажал заданную атрибутом ALRT горячую клавишу.
EVENT:Drop	Кнопка мыши отпущена над объектом - целью операции “потащить и отпустить”

**Пример:**

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
TEXT,AT(0,0,40,40),USE(E1),ALRT(F10Key),CENTER
TEXT,AT(20,0,40,40),USE(E2),KEY(F10Key),HLP('Text4Help')
TEXT,AT(40,0,40,40),USE(E3),SCROLL,OVR,UPR
TEXT,AT(60,0,40,40),USE(E4),CURSOR(CURSOR:Wait),RIGHT
TEXT,AT(80,0,40,40),USE(E5),DISABLE,FONT('Arial',12)
TEXT,AT(100,0,40,40),USE(E6),HVSCROLL,LEFT
TEXT,AT(120,0,40,40),USE(E7),REQ,INS,CAP,MSG('Text Field 7')
END
```

**VBX (объявить окно пользовательского объекта .VBX)**

**VBX(текст) ,AT( ) [,CLASS( )] [,CURSOR( )] [,USE( )] [,DISABLE**  
**[,KEY( )] [,MSG( )] [,HLP( )] [,SKIP] [,FULL] [,SCROLL] [,ALRT( )**  
**[,HIDE] [,FONT( )] [,DROPID( )] [,TIP( )] [,свойство( значение)]**

<b>VBX</b>	Поместить объект, разработанный на Visual Basic (.VBX) в окно или инструментальную панель.
<i>текст</i>	Строковая константа, содержащая заголовок объекта.
<b>AT</b>	Задаёт первоначальные размеры и расположение объекта. Если этот атрибут опущен, то значения определяются самим объектом.
<b>CLASS</b>	Задаёт имя файла, в котором находится объект VBX и тип объекта.
<b>CURSOR</b>	Задаёт форму, которую должен принимать курсор мыши, при попадании в это поле. Если этот атрибут опущен, то используется значение атрибута CURSOR структуры WINDOW, если же и тот не указан, то курсор имеет форму, используемую в среде Windows по умолчанию.
<b>USE</b>	Метка переменной, принимающей значение объекта.
<b>DISABLE</b>	Указывает, что когда в окно WINDOW (или APPLICATION) раскрывается впервые, данное поле выглядит затухеванным.
<b>KEY</b>	Задаёт целочисленную константу или мнемонический код клавиши (или комбинации клавиш), с помощью которой фокус немедленно переключается на это поле.
<b>MSG</b>	Задаёт строковую константу, содержащую текст, который следует вывести в строке состояния, когда на данное поле переключен фокус.
<b>HLP</b>	Задаёт строковую константу, содержащую идентификатор системы справки для данного поля.
<b>SKIP</b>	Указывает, что фокус ввода на это поле не переключается и воздействовать на него можно только посредством мыши или клавиши ускоренного доступа.
<b>FULL</b>	Указывает, что по любому из опущенных параметров атрибута AT (ширина или высота) поле расширяется таким образом, чтобы занимать

	всю протяженность окна.
<b>SCROLL</b>	Указывает, что поле подвергается скроллингу вместе с содержимым окна.
<b>ALRT</b>	Задаёт активные горячие клавиши для данного поля.
<b>HIDE</b>	Задаёт, что когда окно WINDOW или APPLICATION раскрывается в первый раз, данное поле не прорисовывается. Для того, чтобы отобразить это поле нужно использовать оператор UNHIDE.
<b>FONT</b>	Задаёт шрифт для отображения данных в этом поле.
<b>DROPID</b>	Указывает, что данное управляющее поле может служить областью в которой происходит вторая часть операции “поташить и отпустить” - отпуская объект.
<b>TIP</b>	Задаёт текст, который выводится как “возникающая” подсказка, когда курсор мыши задерживается на экранном объекте.
<i>свойство</i>	Строковая константа, содержащая имя пользовательского (не системного) свойства, устанавливаемого для данного экранного объекта.
<i>значение</i>	Строковая константа, содержащая значение свойства или метку соответствия для свойства.

Оператором **VBX** объект, разработанный на Visual Basic (.VBX) помещается в окно или инструментальную панель. Размеры и местоположение этого экранного объекта определяются параметрами атрибута AT.

Атрибут свойство позволяет указать значения дополнительных свойств, которые могут требоваться объектом .VBX. Это свойства, задание которых обеспечивает правильную работу объекта .VBX и не являющиеся стандартными для Clarion свойствами, такими как AT, CURSOR или USE. Пользовательский экранный объект воспринимает только те свойства, которые для него определены. Допустимые свойства и их значения должны быть описаны в документации на эти пользовательские экранные объекты. Для одного объекта .VBX можно указать несколько атрибутов свойство.

Объекты .VBX поддерживают механизм, аналогичный оператору ON ERROR языка Visual Basic. В случае возникновения при выполнении объекта .VBX внутренней ошибки в программе возникнет событие EVENT:VBXevent, при котором свойство PROP:VBXEvent получит значение ‘&OnError’, а свойство PROP:VBXEventArgs будет содержать номер ошибки в своем первом аргументе и текст сообщения об ошибке во втором аргументе.

### Генерируемые события:

EVENT:VBXevent	Возникло событие, присущее объекту VBX. Исследуйте значения свойств PROP:VBXEvent и PROP:VBXEventArgs.
EVENT:Selected	На объект переключен фокус.
EVENT:Accepted	Пользователь завершил работу с данным объектом.
EVENT:PreAlertKey	Пользователь нажал определенную атрибутом ALRT горячую

клавишу.

EVENT:AlertKey

Пользователь нажал определенную атрибутом ALRT горячую

клавишу.

EVENT:Drop

Кнопка мыши отпущена над объектом - целью операции

“поташить и отпустить”.

### Пример:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           VBX,AT(0,0,120,320),USE(C1), |
           CLASS('graph.vbx','graph'),'graphstyle'('2')
           END
MsgNum    LONG
MsgTxt    STRING
CODE
OPEN(MDIChild)
ACCEPT
CASE EVENT()
  OF ?EVENT:VBXevent
    IF ?C1{PROP:VBXEvent} = '&OnError'          !Проверить условие ON ERROR
      MsgNum = ?C1{PROP:VBXArg,1}
      MsgTxt = ?C1{PROP:VBXArg,2}
      MESSAGE('VBX Error ' & MsgNum & ' ' & MsgTxt)
    END
  END
CASE ACCEPTED()
  OF ?C1
    ?C1{'graphstyle'} = '3' !Изменить “на ходу” значение свойства “graphstyle”
    ! используя синтаксис доступа к свойствам во время
    ! выполнения программы
  END
END
```



## Атрибуты экранных объектов

### ALRT (установить горячие клавиши для объекта)

**ALRT**(код\_клавиши)

**ALRT**            Задает горячую клавишу активную в то время, когда на данное поле переключен фокус.

код\_клавиши    Числовая константа кода клавиши или мнемоническая метка соответствия кода клавиши.

Атрибут **ALRT** задает горячую клавишу активную в то время, когда на данное поле переключен фокус. Когда пользователь нажимает горячую клавишу, заданную для экранного поля атрибутом **ALRT**, генерируются два зависящих от типа поля события: **EVENT:PreAlertKey** и **EVENT:AlertKey**. Если при обработке события **EVENT:PreAlertKey** в программе выполнить оператор **CYCLE**, то тем самым “отрубится” событие **EVENT:AlertKey**, предупреждая стандартную обработку библиотечной процедурой нажатия заданной атрибутом **ALRT** клавиши для экранного объекта.

Для одного объекта можно задать несколько атрибутов **ALRT**. Оператор **ALRT** и атрибут **ALRT** для окна или объекта - это совершенно разные вещи. Это означает, что отмена горячих клавиш оператором **ALRT** не влияет на горячие клавиши включенные атрибутами **ALRT**.

#### Пример:

```
WinOne    WINDOW,AT(0,0,160,400)
           ENTRY,AT(6,40),USE(SomeVar1),ALRT(F9Key)    !F9 включена для поля
           ENTRY,AT(60,40),USE(SomeVar2),ALRT(F10Key)    !F10 включена для
поля
           END
           CODE
           OPEN(WinOne)
           ACCEPT
           CASE FIELD()
           OF ?SomeVar1
           CASE EVENT()
           OF EVENT:PreAlertKey    !Предварительная проверка события нажатия гор.
клавиши
           IF NOT SomeVar1
           CYCLE    !Прервать обработку события для другого поля
           END
           OF EVENT:AlertKey    !Обработка события
           DO F9Routine
           END
           OF ?SomeVar2
```

```

CASE EVENT()
  OF EVENT:AlertKey      !Обработка нажатия горячей клавиши
    DO F10Routine
  END
END
END
END

```

## ANGLE (установить угол подкоторым выводится объект)

**ANGLE**( *размер* )

**ANGLE** определяет ориентацию экранного объекта STRING.  
*размер* Целочисленная константа или константное выражение, которое задает величину поворота в десятых долях градуса. Положительный угол отсчитывается в направлении против движения часовой стрелки от обычного горизонтального положения текста. Допустимый диапазон значений от 3600 до -3600.

Атрибут **ANGLE** указывает вывод экранного объекта STRING под заданным углом, измеряемым проив движения часовой стрелки от горизонтального направления вывода текста. Атот атрибут позволяет вывести текст помимо стандартной горизонтальной ориентации под любым углом. Шрифт для этой строки должен быть TrueType.

### Пример:

```

WinOne  WINDOW,AT(0,0,160,400),FONT('Arial')
        STRING('String Constant'),AT(6,40),USE(?String1)
        !Вывести текст горизонтально
        STRING('String Constant'),AT(6,40),USE(?String2),ANGLE(900)
        ! Вывести текст вертикально
        STRING('String Constant'),AT(6,40),USE(?String3),ANGLE(1800)
        ! Вывести текст "вверх ногами"
        END

```

## AT (установить положение и размеры поля в окне)

**AT**([*x*] [*y*] [*ширина*] [*высота*])

**AT** определяет положение и размеры объекта.  
*x* Целочисленная константа или константное выражение, которое задает горизонтальную координату левого верхнего угла объекта. Если этот параметр опущен, то по умолчанию библиотечной процедурой выбирается нуль.  
*y* Целочисленная константа или константное выражение, которое задает вертикальную координату левого верхнего угла объекта. Если этот параметр

<i>ширина</i>	опущен, то по умолчанию библиотечной процедурой выбирается ноль. Целочисленная константа или константное выражение, которое задает ширину объекта. Если этот параметр опущен, то значение по умолчанию выбирается библиотечной процедурой.
<i>высота</i>	Целочисленная константа или константное выражение, которое задает высоту объекта. Если этот параметр опущен, то значение по умолчанию выбирается библиотечной процедурой.

Атрибут **AT** определяет местоположение и размеры объекта. Если любой из параметров атрибута опущен, то значение по умолчанию выбирается библиотечной процедурой.

Значения параметров *x*, *y*, ширина и высота измеряются в условных единицах. Условная единица определяется как одна четвертая средней ширины символа в шрифте на одну восьмую средней высоты. Размер условной единицы зависит от размера шрифта, используемого по умолчанию для окна. Эта единица измерения базируется на размере шрифта, заданного для окна атрибутом **FONT** или шрифтом, установленным по умолчанию средой Windows.

### Пример:

```
!Размерность в условных единицах
WinOne    WINDOW,AT(0,0,160,400)
          ENTRY,AT(8,40,80,8) !Приблиз. 2 символ слева, 5 строк вниз, шириной - 20
          высотой 1 символ
          END
```

## AUTOSIZE (установить изменение размеров объекта OLE)

### AUTOSIZE

Атрибут **AUTOSIZE** указывает, что OLE-объект автоматически изменяет свои размеры, когда, используя синтаксис изменения свойств во время выполнения программы, изменением свойства **PROP:At** изменяются значения параметров атрибута **AT**.

## BEVEL (установить эффект объемности границ объекта)

### BEVEL( внешний [,внутренний] [,стиль] )

<b>BEVEL</b>	Установить эффект объемности границ объекта.
<b>внешний</b>	Целочисленная константа или константное выражение, задающая ширину внешней кромки границы. Если эта величина отрицательна, то внешняя кромка выглядит вдавленной; если положительна, то - выпуклой.
<b>внутренний</b>	Целочисленная константа или константное выражение, задающая ширину внутренней кромки границы. Если эта величина отрицательна, то

внешняя кромка выглядит вдавленной; если положительна, то - выпуклой. Если этот параметр опущен, то внутренней кромки границы нет вообще.

**стиль**

Целочисленная константа или константное выражение, задающее четкую кромку границы, не учитывающую знаки параметров внешний и внутренний.

Атрибут **BEVEL** объектов **PANEL**, **OPTION**, **GROUP** и **REGION** задает эффект объемности границ объектов. Знаки параметров внешний и внутренний определяют выглядит ли объект выпуклым или вдавленным. Параметр стиль позволяет сделать четкую границу объекта. Он представляет собой 16-ти битовую карту, в которой биты означают:

Bits:	15 - 12	11 - 08	07 - 04	03 - 00
Edge:	left	top	right	bottom

Каждая из этих 4-х битовых групп дальше делится на секции по 2 бита, которые управляют внешним видом внутренней и внешней частей границы. Младшие два бита каждой тетрады определяют внешнюю часть, тогда как старшие два бита отвечают за внутреннюю часть границы.

Binary:	00b	01b	10b	11b
Result:	нет ребра	выпуклое	вдавленное	серое

Сочетание этих комбинаций в тетрады дает обно ребро границы:

0110b	=	внутренняя часть выпуклая, внешняя вдавленная
1001b	=	внутренняя часть вдавленная, внешняя выпуклая

**Пример:**

```
WinOne  WINDOW,AT(0,0,160,400)
        PANEL,AT(25,15,50,50),USE(?Panel1),BEVEL(5,-5)
        !Выпуклая внешняя част, вдавленная внутренняя
        PANEL,AT(0,0,,),USE(?Panel2),FULL,BEVEL(2,2,1111010110101001b)
        !слева - все серое
        !верхняя граница = внутри выпуклая, извне выпуклая
        !справа= внутри вдавленная, извне вдавленная
        !снизу= внутри вдавленная, извне выпуклая
        REGION,AT(0,80,5,,),USE(?ResizeBar),FULL,IMM,BEVEL(2,2,0101000010100000b)
        !вертикальная шкала изменения размеров

END
```

**BOXED (установить рамку вокруг группы экранных объектов)****BOXED**

Атрибут **BOXED** задает одинарную рамку вокруг объектов в структуре **GROUP** или

OPTIONS. В разрыве верхней части рамки выводится значение параметра текст структуры GROUP или OPTIONS. Если атрибут BOXED опущен, то значение параметра текст структуры GROUP или OPTIONS на экран не выводится.

## CAP, UPR (установить регистр букв)

Атрибуты **CAP** и **UPR** задают автоматическое изменение регистра букв текста, вводимого в поля типа ENTRY или TEXT в окне с атрибутом MASK. Атрибут UPR указывает, что все буквы прописные.

Атрибут CAP указывает, что данные по Типу Имен Собственных, т. е. первая буква в каждом слове прописная, а остальные строчные. Пользователь может изменить такой режим ввода, нажав клавишу SHIFT, позволяющую ввести прописные буквы в середине слова (позволяя ввести имя типа “McDowell”) или нажав клавишу SHIFT при включенном режиме Caps-Lock включить режим строчных букв для первой буквы (чтобы ввести имя подобное “von Richtofen”).

## CLASS (установить класс объекта .VBX)

### CLASS(файл [,имя] )

<b>CLASS</b>	Задаст имя файла и тип пользовательского .VBX объекта.
<b>файл</b>	Строковая константа, содержащая имя файла .VBX (включая расширение .VBX), в котором реализован объект.
<b>имя</b>	Строковая константа, содержащая имя типа пользовательского объекта из .VBX файла. Если этот параметр опущен, то используется первый тип определенный в файле.

Атрибут **CLASS** задает имя файла и тип пользовательского .VBX -объекта. Параметр имя идентифицирует в файле .VBX, содержащем несколько объектов, конкретный объект, который следует использовать.

### Пример:

```
WinOne    WINDOW,AT(0,0,160,400)
          CUSTOM,AT(0,0,120,320),CLASS('graph.vbx','graph'),'graphstyle'('2')
          END
```

## CLIP (установить усечение OLE-объекта)

### CLIP

]Атрибут **CLIP** указывает, что для внедряемого объекта высвечивается только та его часть, которая помещается рамки, определенные атрибутом AT экранного объекта-контейнера. Если внедряемый объект больше чем определенное для него экранное поле OLE, то отображается только его верхний левый угол.

**COLOR (установить цвет экранного объекта)**

**COLOR( цвет [ , выбранный\_передн ] [ ,выбранный\_фон ] )**

**COLOR**

Задаёт цвет объекта.

**цвет**

Целочисленная константа типа LONG или ULONG, или задающая константу метка соответствия, содержащая в трех младших байтах красную, зеленую и синюю компоненты, составляющие цвет; или метка соответствия для стандартного в Windows значения цвета.

**выбранный\_передн**

Целочисленная константа типа LONG or ULONG, или задающая константу метка соответствия, содержащая в трех младших байтах (байты 0, 1, and 2), красную, зеленую и синюю компоненты, составляющие цвет; или метка соответствия для стандартного в Windows значения цвета. Этот параметр задает используемый по умолчанию цвет переднего плана для текста объекта, на который может переключаться фокус ввода.

**выбранный\_фон**

Целочисленная константа типа LONG или ULONG, или задающая константу етка соответствия, содержащая в трех младших байтах (байты 0, 1, and 2), красную, зеленую и синюю компоненты, составляющие цвет; или метка соответствия для стандартного в Windows значения цвета. Этот параметр задает используемый по умолчанию цвет фона для текста объекта, на который может переключаться фокус ввода.

Атрибут **COLOR** задает цвет высвечивания объекта тип LINE. Для объектов типа BOX, ELLIPSE и REGION параметр цвет указывает цвет рамки. Для всех других экранных объектов параметр цвет задает цвет фона объекта перебивающий стандартную цветовую гамму Windows для объектов такого типа. Для большинства из объектов, на которые может переключаться фокус параметры выбранный\_передн и выбранный\_фон задают цвет переднего плана и цвет фона выбранного (активизированного) объекта.

Мнемонические имена для стандартных цветов среды Windows содержатся в файле EQUATES.CLW. Для используемых во время выполнения программы аппаратных средств Windows автоматически подбирает цвет, наиболее соответствующий цвету, заданному параметром цвет. Стандартные цвета Windows могут переопределяться пользователем в окне Control Panel. Когда это происходит, все экранные объекты, для которых использовался стандартный цвет, автоматически перерисовываются.

**Пример:**

```
WinOne WINDOW,AT(0,0,160,400)
        BOX,AT(20,20,20,20),COLOR(COLOR:ACTIVEBORDER)
!Стандартный в Windows цвет рамки активного окна
        BOX,AT(100,100,20,20),COLOR(00FF0000h)      !Синий
        BOX,AT(140,140,20,20),COLOR(0000FF00h)      !Зеленый
        BOX,AT(180,180,20,20),COLOR(000000FFh)      !Красный
```

END

## **COLUMN (установить полосу-курсор в окне списка)**

**COLUMN**

Атрибут COLUMN задает размеры выделенной полосы-курсора в окне списка или комбинированном окне списка с несколькими столбцами выводимых данных.

## **COMPATIBILITY(установить режим совместимости объекта OLE)**

**COMPATIBILITY( режим )**

**COMPATIBILITY‘**      Задает установки совместимости для OLE - объекта.  
*режим*                      Целочисленная константа для значения установки.

Атрибут **COMPATIBILITY** задает режим совместимости для отдельных OLE и .OCX объектов, для которых это требуется. В общем случае режим следует устанавливать в ноль (0), однако некоторые OLE-объекты (такие как “битмэп” редактор Windows) не работают, если режим не установлен в единицу (1).

### **Пример:**

```
WinOne      WINDOW,AT(0,0,200,200)
OLE,AT(10,10,160,100), USE(?OLEObject), CREATE('Excel.Sheet.5'), COMPATIBILITY(0)
END
END
```

## **CREATE (создать объект элемента управления OLE)**

**CREATE( сервер [, объект ] )**

**CREATE**                      Указывает создание нового объекта для OLE.  
*сервер*                      Строковая константа, содержащая имя программы-сервера OLE, под  
                                  каким он зарегистрирован в операционной системе.  
*объект*                      Строковая константа, содержащая имя специального файла (OLE Compound  
                                  Storage file) и объекта в этом файле, который должен быть открыт.

Атрибут **CREATE** в данном поле OLE создается новый объект OLE или .OCX. Значение параметра сервер представляет собой имя объекта в том виде как объект регистрируется в операционной системе ( в Windows 95 эта информация доступна с помощью программы REGEDIT.EXE в разделе HKEY\_CLASSES\_ROOT, или в программе Microsoft System Information, поставляющейся вместе с Microsoft Office - MSINFO32.EXE).

Когда указан параметр объект, CREATE работает просто как атрибут OPEN, открывая сохраненный в специальном файле (OLE Compound Storage) (и игнорируя параметр сервер) объект для поля OLE. При открытии объекта загружается сохраненный вариант значений свойств контейнера, поэтому при открытии объекта не нужно указывать значения свойств. Синтаксис параметра объект должен иметь следующий вид: ИмяФайла\!ИмяОбъекта.

### Пример:

```
WinOne    WINDOW,AT(0,0,200,200)
          OLE,AT(10,10,160,100),USE(?OLEObject),CREATE('Excel.Sheet.5')
          END
          END
```

## CURSOR (установить форму курсора мыши для объекта)

### CURSOR(файл)

**CURSOR**      Задает форму курсора мыши для объекта.  
*файл*          Строковая константа, содержащая имя файла .CUR, или мнемоническое имя стандартной для Windows формы курсора мыши. Файл .CUR включается в исполняемый модуль в качестве ресурса.

Атрибут CURSOR задает форму курсора мыши в то время, когда тот находится в пределах данного объекта.

В файле EQUATES.CLW содержатся операторы EQUATE для стандартных в Windows форм курсора мыши.

**Пример этих операторов EQUATE** (полный список см. в файле EQUATES.CLW):

CURSOR:None	Нет курсора
CURSOR:Arrow	Обычный курсор в виде стрелки
CURSOR:IBeam	Курсор в виде заглавной буквы I похожий на двутавр
CURSOR:Wait	Песочные часы
CURSOR:Cross	Курсор в виде большого символа плюс
CURSOR:UpArrow	Курсор в виде стрелки направленной вверх
CURSOR:Size	Курсор в виде четырех стрелок, направленных в разные стороны.
CURSOR:Icon	Пиктограмма в рамке
CURSOR:SizeNWSE	Стрелки в направлении северо-запад - юго-восток
CURSOR:SizeNESW	Стрелки в направлении северо-восток - юго-запад
CURSOR:SizeWE	Стрелки в направлении запад - восток
CURSOR:SizeNS	Стрелки в направлении север - юг
CURSOR:DragWE	Стрелки в направлении запад - восток



**Пример:**

```
WinOne    WINDOW,AT(0,0,160,400)
          REGION,AT(20,20,20,20),CUSOR(CURSOR:IBeam)
          REGION,AT(100,100,20,20),CURSOR('Custom.CUR')
          END
```

**DEFAULT (установить экранную кнопку для клавиши ENTER)****DEFAULT**

Атрибут **DEFAULT** указывает экранную кнопку типа **BUTTON**, которая автоматически нажимается, когда пользователь нажимает клавишу **ENTER**. Этот атрибут должна иметь только одна активная кнопка **BUTTON**.

**DISABLE (при открытии окна элемент управления не действует)****DISABLE**

Атрибут **DISABLE** указывает, что при открытии окна **WINDOW** или **APPLICATION** данный элемент управления не работает и выглядит затухеванным. Такой элемент можно активизировать оператором **ENABLE**.

**DOCUMENT (создать объект из файла для элемента OLE)****DOCUMENT( имя\_файла )**

**DOCUMENT**            Задаёт создание из файла данных, характерного для прикладной программы сервера, объекта для элемента OLE  
*имя\_файла*            Строковая константа, содержащая имя файла.

Атрибут **DOCUMENT** указывает создание объекта для элемента OLE из файла данных, характерного для прикладной программы сервера. Параметр *имя\_файла* должен содержать полностью квалифицированное (содержащее путь) имя файла, если только файл не располагается в том же каталоге, что и прикладная программа, обрабатывающая объект OLE.

**Пример:**

```
WinOne    WINDOW,AT(0,0,200,200)
          OLE,AT(10,10,160,100),USE(?OLEObject),DOCUMENT('Book1.XLS')    !Таблица
Excel
          MENUBAR
          MENU('&Clarion App')
```

```

        ITEM('&Deactivate Object'),USE(?DeactOLE)
    END
END
END
END
END

```

## DROP (установить поведение окна списка)

**DROP**(*число*)

**DROP** Указывает, что список появляется, только когда пользователь нажимает клавишу стрелки или щелкает мышью на пиктограмме раскрытия списка.

*число* Целочисленная константа, которая задает число высвечиваемых элементов списка.

Атрибут **DROP** указывает, что список появляется, только когда пользователь нажимает клавишу стрелки или щелкает мышью на пиктограмме раскрытия списка справа от текущего выбранного элемента списка. Как только список раскрывается, в нем выводится заданное параметром число элементов списка. Если атрибут **DROP** опущен, то в окне списка или комбинированном окне списка, всегда отображается число элементов, определяемое параметром высота атрибута **AT** окна списка.

Для объектов в окне с атрибутом **MODAL** атрибут **DROP** не работает и не должен использоваться.

Для того, чтобы переопределить используемую по умолчанию для раскрытия списка пиктограмму “стрелка вниз”, можно присвоить свойству **PROP:Icon** имя другой пиктограммы.

### Пример:

```

WinOne    WINDOW,AT(0,0,160,400)
          LIST,AT(120,0,20,20),USE(?L7),FROM(Que1),DROP(6)
          COMBO(@S8),AT(120,120,20,20),USE(?C7),FROM(Que2),DROP(8)
          END

```

## DRAGID (установить идентификатор источника )

**DRAGID**( *идентификатор* [, *идентификатор*] )

**DRAGID** Указывает, что поле типа **LIST** или **REGION** может служить в качестве источника данных для операции “поташить и отпустить”.

*идентификатор* Строковая константа, содержащая идентификатор, используемый для того чтобы обозначить объекты, которые могут служить приемниками данных от данного объекта. Идентификаторы, которые начинаются

символом тильда (~) показывают, что данные также можно “перетащить” во внешнюю программу (на Clarion). Один атрибут DRAGID может содержать до 16-ти идентификаторов.

Атрибут **DRAGID** указывает, что поле типа LIST или REGION может служить в качестве источника данных для операции “поташить и отпустить”. Этот атрибут работает совместно с атрибутом DROPID другого объекта. Строки идентификаторов DRAGID (до 16-ти) определяют допустимые ключи, которые должны соответствовать параметрам идентификатор атрибута DROPID экранных полей - приемников данных в операции переноса. Тем самым обеспечивается идентификация объектов, в которые допускаются операции переноса данных.

Операция “поташить и отпустить” происходит, когда пользователь перетаскивает информацию из поля с атрибутом DRAGID в поле с атрибутом DROPID. Для того, чтобы операция была успешной, оба поля должны иметь по крайней мере одну одинаковую строку идентификатора в соответствующих атрибутах DRAGID и DROPID.

### Пример:

```
WinOne  WINDOW,AT(0,0,160,400)
        LIST,AT(120,0,20,20),USE(?List1),FROM(Que1),DRAGID('FromList1')
        !Допустимо брать данные, но не отпускать здесь
        LIST,AT(120,120,20,20),USE(?List2),FROM(Que2),DROPID('FromList1')
        !Допустимо переносить сюда из List1, но отсюда нельзя
        END
        CODE
        OPEN(WinOne)
        ACCEPT
        CASE EVENT()
        OF EVENT:Drag
            IF DRAGID()
                SETDROPID(Que1)
            END
            !Если происходит попытка переноса данных
            ! проверить допустимость переноса отсюда
            ! и взять данные для переноса
        OF EVENT:Drop
            Que2 = DROPID()
            ADD(Que2)
            !Когда событие отпускания успешное
            ! взять переносимые данные
            ! и добавить их в очередь
        END
        END
```

Смотри также: DROPID

### **DROPID (установить идентификатор поля-приемника данных)**

**DROPID( идентификатор [, идентификатор] )**

#### **DROPID**

Указывает, что поле типа LIST или REGION может служить в

качестве приемника данных в операции “поташить и отпустить”.

*идентификатор* Строковая константа, содержащая идентификатор, используемый для того чтобы обозначить объекты, которые могут служить источниками данных для данного объекта. Один атрибут DROPID может содержать до 16-ти идентификаторов. Идентификаторы, которые начинаются символом тильда (~) показывают, что данные также можно “переташить” из внешней программы (на Clarion). Идентификатор ‘~FILE’ показывает объект-приемник воспринимает разделенный запятыми список имен файлов перенесенный из окна File Manager Windows.

Атрибут DROPID указывает, что поле может служить в качестве приемника данных в операции “поташить и отпустить”. Этот атрибут работает совместно с атрибутом DRAGID другого объекта. Строки идентификаторов DROPID (до 16-ти) определяют допустимые ключи, которые должны соответствовать параметрам идентификатор атрибута DRAGID экранных полей - источников операции переноса. Тем самым обеспечивается идентификация объектов, из которых допускаются операции переноса данных.

Операция “поташить и отпустить” происходит, когда пользователь перетаскивает информацию из поля с атрибутом DRAGID в поле с атрибутом DROPID. Для того, чтобы операция была успешной, оба поля должны иметь по крайней мере одну одинаковую строку идентификатора в соответствующих атрибутах DRAGID и DROPID.

### Пример:

```
WinOne  WINDOW,AT(0,0,160,400)
        LIST,AT(120,0,20,20),USE(?List1),FROM(Que1),DRAGID('FromList1')
        !Допустимо брать данные, но не отпускать здесь
        LIST,AT(120,120,20,20),USE(?List2),FROM(Que2),DROPID('FromList1')
        !Допустимо переносить сюда из List1 и
        !Windows File Manager , но отсюда нельзя
        END
        CODE
        OPEN(WinOne)
        ACCEPT
        CASE EVENT()
        OF EVENT:Drag
            IF DRAGID()
                SETDROPID(Que1)
            END
            !Если происходит попытка переноса данных
            ! проверить допустимость переноса отсюда
            ! и взять данные для переноса
        OF EVENT:Drop
            Que2 = DROPID()
            ADD(Que2)
            !Когда событие отпускания успешное
            ! взять переносимые данные
            ! и добавить их в очередь
        END
        END
```

Смотри также: DRAGID

## FILL (установить цвет внутренней части объекта)

**FILL(кзс)**

### FILL

кзс

Задает цвет внутренней части объекта.

Целочисленная константа типа LONG или ULONG или мнемоническое имя константы такого типа, содержащая в трех младших байтах (байты 0, 1 и 2) величины красной, зеленой и синей компонент, из которых состоит цвет экранного объекта или мнемоническое имя стандартного для среды Windows цвета.

Атрибут **FILL** указывает цвет внутренней части полей BOX, ELLIPSE или REGION. Если этот атрибут опущен, то внутренняя часть этих полей не заполняется цветом.

### Пример:

```
WinOne WINDOW,AT(0,0,160,400)
        BOX,AT(20,20,20,20),FILL(COLOR:ACTIVEBORDER)
        !Цвет рамки активного окна Windows
        BOX,AT(100,100,20,20),FILL(00FF0000h)           !синий
        BOX,AT(140,140,20,20),FILL(0000FF00h)           !зеленый
        BOX,AT(180,180,20,20),FILL(000000FFh)           !красный
END
```

## FONT (установить шрифт для поля)

**FONT([начертание][,размер][,цвет][,стиль])**

### FONT

начертание

Задает шрифт, используемый для поля.

Строковая константа, содержащая название шрифта. Если этот параметр опущен, то используется шрифт по умолчанию.

размер

Целочисленная константа, содержащая размер шрифта в пунктах (1 пункт 1/72 дюйма). Если этот параметр опущен, то используется размер системного шрифта, используемого по умолчанию.

цвет

Целочисленная константа типа LONG, содержащая в младших трех байтах значения для красной, зеленой и голубой составляющих цвета или мнемоническое имя, задаваемое оператором EQUATE для стандартных в Windows значений, определяющих цвета.

стиль

Целочисленная константа или константное выражение или мнемоническая метка, задающая толщину штриха и стиль букв шрифта. Если этот параметр опущен, то используется стиль системного шрифта, используемого по умолчанию.

Атрибут **FONT** задает шрифт, используемый для поля, перекрывая любой шрифт, указанный в атрибуте FONT для окна.

Параметр начертание может быть именем любого шрифта, зарегистрированного в системе Windows. Файл EQUATES.CLW содержит операторы EQUATE для стандартных значение параметра стиль. Значения стиля в диапазоне от 0 до 1000 задают толщину штриха шрифта. Это значение можно прибавить к числам означающим курсив, подчеркнутый или перечеркнутый текст. Операторы EQUATE в файле EQUATES.CLW:

FONT:thin	EQUATE (100)
FONT:regular	EQUATE (400)
FONT:bold	EQUATE (700)
FONT:italic	EQUATE (01000h)
FONT:underline	EQUATE (02000h)
FONT:strikeout	EQUATE (04000h)

**Пример:**

```
WinOne WINDOW,AT(0,0,160,400)
LIST,AT(120,0,20,20),USE(?L7),FROM(Que1),FONT('Arial',14,0FFh)
!14 пунктов Arial, красный, нормальный
LIST,AT(120,120,20,20),USE(?C7),FROM(Que2),FONT('Arial',14,0,700)
!14 пунктов Arial, черный, полужирный
LIST,AT(120,240,20,20),USE(?C7),FROM(Que2),FONT('Arial',14,0,700+01000h)
!14 пунктов Arial, черный, полужирный курсив
END
```

**FORMAT (установить структуру окна LIST или COMBO)**

**FORMAT** (*строка формата*)

<b>FORMAT</b>	Задает структуру данных в окне LIST или COMBO.
<i>строка формата</i>	Строковая константа, задающая формат высвечиваемого списка.

Атрибут **FORMAT** задает структуру данных в окне списка или комбинированном окне списка. Строка формата содержит информацию для форматирования данных в окне списка в одну или несколько колонок.

В строке формата содержатся “спецификации полей”, которые устанавливают соответствие полей из выводимой в окне списка очереди. Для того, чтобы выводить несколько полей как единое целое, несколько “спецификаций” могут быть сгруппированы квадратными скобками в “группу полей”.

В списке окна выводятся только те поля из очереди, для которых есть спецификации в строке формата. Это означает, что если два поля указаны в строке формата, а очередь состоит

из трех, то в окне списка выводятся только два, указанные в строке формата.

### Формат спецификации поля: ширина выравнивание [ ( отступ ) ][ модификаторы ]

- ширина** Обязательный компонент. Целое число, задающее ширину поля. Указывается в условных единицах.
- выравнивание** Одна заглавная буква (L, R, C, или D), которая указывает выравнивание влево, вправо центрирование или выравнивание по десятичной точке. Обязательно должен быть указан один из типов выравнивания.
- отступ** Необязательно указываемое целое число, которое задает отступ от линии выравнивания. Может быть отрицательным. При выравнивании влево отступ определяет левую границу, при выравнивании вправо или на десятичную точку - правую границу, а при выравнивании по центру определяет смещение относительно середины поля (отрицательное число - смещение влево).
- модификаторы:** Необязательные специальные перечисленные ниже символы, которые служат для того, чтобы изменить формат вывода поля или группы полей. Для одного поля или группы можно использовать несколько модификаторов.
- \*** Звездочка означает, что информация о цвете для данного поля содержится в четырех полях типа LONG, идущих в структуре QUEUE (или строке, заданной атрибутом FROM) непосредственно следом за полем данных. Обычно это четыре цвета: цвет переднего плана, цвет фона, цвет переднего плана выбранного поля и цвет фона выбранного поля (в таком порядке).
- I** Символ I означает номер пиктограммы для данного поля содержится в поле типа LONG, идущем в структуре QUEUE (или строке, заданной атрибутом FROM) непосредственно следом за полем данных. Это поле типа LONG содержит число - индекс элемента в списке пиктограмм, связанном со полем LIST свойством PROP:IconList. Если еще задано и звездочка, означающая наличие информации о цвете, то поле - номер пиктограммы идет следом за полями несущими информацию о цвете.
- T [ (подавление) ]** Символ T означает окно списка представляет иерархическую структуру. Уровень ветвления содержится в поле типа LONG, которое следует в структуре QUEUE (или строке, заданной атрибутом FROM) непосредственно следом за полем данных. Если также указаны \* и I, то уровень ветвления следует за всеми соответствующими им полями типа LONG. Состояние данного уровня ветвления (раскрыто/свернуто) определяется знаком величины, содержащейся в поле типа LONG представляющем уровень ветвления (положительная = раскрыто, а отрицательная = свернуто). Необязательный параметр подавление может содержать 1 для того, чтобы обозначить тот факт, корневой уровень имеет

номер 1, а не 0, допускается -1, чтобы обозначит свернутый корневой уровень. Этот параметр может также содержать R, для подавления соединительных линий корневого уровня, символ L, чтобы указать подавление соединительных линий между уровнями иерархии, символ B, чтобы не выводить квадратик символизирующий расширение, и символ I, чтобы подавить отступ для уровней иерархии (что неявно подавляет и квадратики и соединительные линии).

**~заголовок~ [выравнивание[(отступ)]]** Строка заголовка заключенная в тильды, за которой следуют необязательные выравнивание и/или отступ, задает вывод заголовка в начале списка. Если не указаны выравнивание и отступ для заголовка, то используются то же самое выравнивание и отступ что и для поля.

**@шаблон@** Шаблон форматирования для отображения поля. Символ @ в конце нужен для указания конца шаблона, так что шаблон подобный @N12~Kг~ можно использовать не создавая неоднозначности в строке формата.

**?** Знак вопроса определяет локаторное поле в комбинированном окне списка с селекторным полем. Для выпадающего многоколоночного окна списка знак вопроса определяет поле, значение которого выводится в окне выбранного в данный момент значения.

**#номер#** Число заключенное между символами фунта (#) показывает номер поля из очереди, которое должно выводиться в окне списка. Следующие поля в строке формата, не имеющие явно заданного #номера# берутся по порядку начиная от предыдущего явно заданного поля. Например, #2# для первого поля в строке формата означает вывод в списке второго поля из очереди, пропуская первое. Если число полей указанных в строке формата больше или равно числу полей в очереди, то формат “зацикливается”, переходя на начало очереди.

**\_** Знак подчеркивания означает подчеркивание поля.

**/** Слэш приводит к тому, что следующее поле выводится в новой строке (используется только внутри групп полей).

**|** Вертикальная черта помещает вертикальную линию справа от поля.

**M** Символ M позволяет во время выполнения динамически изменять размер поля или группы полей. Пользователь может “поташить” вертикальную черту справа (если она есть) или правую границу области данных.

**F** Символ F создает фиксированную колонку, которая остается на экране при горизонтальном скроллинге. Фиксированные поля и группы должны быть в начале списка. Для поля внутри группы это модификатор игнорируется .

**S(целое число)** Символ S, за которым в скобках указано целое число добавляет к группе линейку скроллинга. Целое число определяет число условных единиц на которые происходит прокрутка. Тем самым становится возможным выводить большие поля в колонке небольшого размера. Для



поля внутри группы это модификатор игнорируется.

**Формат спецификации группы полей:***[несколько спецификаций полей] [(размер)] [модификаторы]*

**несколько спецификаций полей** Список спецификаций полей, заключенный в квадратные скобки, которые говорят о том, что эти поля выводятся как единое целое.

**размер** Необязательное целое число, заключенное в круглые скобки, которое задает ширину группы по умолчанию.

**модификаторы** Групповые модификаторы, действующие на всю группу полей. Это те же модификаторы, которые перечислены выше (за исключением \*, I, и T, которые не применимы к группам).

### Пример:

```
PROGRAM
MAP
  RandomAlphaData(*STRING)
END
```

TreeDemo	QUEUE,PRE()	!Очередь, содержащая данные для списка
FName	STRING(20)	
ColorNFG	LONG	!Цвет переднего плана для поля FName
ColorNBG	LONG	!Цвет фона для поля FName
ColorSFG	LONG	!Цвет переднего плана для выбранного поля
FName		
ColorSBG	LONG	!Цвет фона для выбранного поля FName
IconField	LONG	!Номер пиктограммы для поля FName
TreeLevel	LONG	!Уровень ветвления
LName	STRING(20)	
Init	STRING(4)	
	END	

```
Win WINDOW('List Boxes'),AT(0,0,366,181),SYSTEM,DOUBLE
  LIST,AT(0,34,366,146),FROM(TreeDemo),USE(?Show),HVSCROLL, |
  FORMAT('80L*IT~First Name~*80L~Last Name~16C~Initials~')
END
```

```
CODE
LOOP X# = 1 TO 20
  RandomAlphaData(FName)
  ColorNFG = COLOR:White    !Присвоить цвета для поля FNAME
  ColorNBG = COLOR:Maroon
  ColorSFG = COLOR:Yellow
  ColorSBG = COLOR:Blue
```

```

IconField = ((X#-1) % 4) + 1  !Присвоить номер пиктограммы
TreeLevel = ((X#-1) % 4) + 1  !Присвоить уровень иерархии
RandomAlphaData(LName)
RandomAlphaData(Init)
ADD(TD)
END
OPEN(Win)
?Show{PROP:iconlist,1} = ICON:VCRback      !Пиктограмма 1 = <
?Show{PROP:iconlist,2} = ICON:VCRrewind    !Пиктограмма 2 = <<
?Show{PROP:iconlist,3} = ICON:VCRplay      !Пиктограмма 3 = >
?Show{PROP:iconlist,4} = ICON:VCRfastforward !Пиктограмма 4 = >>
ACCEPT
END

```

```

RandomAlphaData PROCEDURE(Field)  !Прототип в MAP: RandomAlphaData(*STRING)
CODE
Z# = RANDOM(1,SIZE(Field))
LOOP Z# = 1 to Y#                  !Заполним каждый символ
    Field[Z#] = CHR(RANDOM(97,122))! произвольной маленькой буквой
END

```

## FROM (установить источник данных для окна списка)

**FROM**(*источник*)

**FROM**                      Задает источник данных для окна списка (LIST), комбинированного окна списка (COMBO) или вращающегося списка (SPIN).

*источник*                    Метка структуры QUEUE, поля внутри очереди или строковая константа, содержащая подлежащие выводу элементы данных.

Атрибут **FROM** задает источник элементов данных, выводимых в окне списка (LIST), комбинированном окне списка (COMBO) или вращающемся списке (SPIN).

Для вращающегося списка источник обычно будет полем из очереди или строкой. Если источник является очередью с несколькими полями, то в поле SPIN выводятся значения только первого поля.

В комбинированном или простом окне списка элементы данных форматируются для вывода в соответствии со значением атрибута FORMAT. Если в качестве источника указана метка структуры QUEUE, то выводятся все поля из очереди. Если в качестве источника задано имя одного поля из очереди, то только это поле и выводится.

Если в качестве источника указана строковая константа, то отдельные элементы данных в ней должны разделяться символом вертикальной черты (|). Для того, чтобы включить вертикальную черту в элемент данных укажите рядом две вертикальные черты, и только одна будет выводиться. Чтобы указать, что элемент пустой, поместите между двумя

разделительными вертикальными чертами хотя бы один пробел (| |).

### Пример:

```
Que1    QUEUE,PREF(Q1)
F1      LONG
F2      STRING(8)
        END
Win1    WINDOW,AT(0,0,160,400)

LIST,AT(120,0,20,20),USE(?L1),FROM(Que1),FORMAT('5C~List~15L~Box~'),COLUMN
        COMBO(@S8),AT(120,120,20,20),USE(?C1),FROM(Q1:F2)
        SPIN(@N8.2),AT(280,0,20,20),USE(SpinVar1),FROM(Q1:F1)
        SPIN(@S4),AT(280,0,20,20),USE(SpinVar2),FROM('Mr.|Mrs.|Ms.|Dr.')
        END
```

## FULL (установить расширение поля на весь экран)

### FULL

Атрибут **FULL** указывает, что по любому из опущенных параметров атрибута **AT** (ширина или высота) объект расширяется таким образом, чтобы занимать всю протяженность окна.

Это атрибут не может указываться для инструментальной панели (TOOLBAR).

## GRID (установить цвет разделительных линий сетки в списке)

### GRID(*кзс*)

#### GRID

*кзс*

Задаст цвет разделительных линий сетки в окне-списка.

Целочисленная константа типа **LONG** или **ULONG**, содержащая в трех младших байтах (байты 0, 1 и 2) значения красного, зеленого и синего компонентов цвета, или содержит метку соответствия стандартного для Windows цвета.

Атрибут **GRID** задает вывод цветной сетки в списке элемента **COMBO** или **LIST**. Операторы **EQUATE** для обозначения стандартных в Windows цветов содержатся в файле **EQUATES.CLW**. Windows автоматически подбирает для используемого во время выполнения программы видеорежима цвет, наиболее соответствующий заданной комбинации красного, зеленого и синего.

### Пример:

```
WinOne  WINDOW,AT(0,0,400,400)
```

```
LIST,AT(0,34,366,146),FROM(TreeDemo),USE(?Show),HVSCROLL,GRID(COLOR:Red) |  
    FORMAT('80L*IT~First Name~*80L~Last Name~16C~Initials~')  
END
```

## HIDE (установить, что поле не выводится при раскрытии окна)

### HIDE

Атрибут HIDE задает, что когда окно WINDOW или APPLICATION раскрывается в первый раз, данное поле не прорисовывается. Для того, чтобы отобразить это поле нужно использовать оператор UNHIDE.

## HLP (установить идентификатор справочной системы)

### HLP(идентификатор)

#### HLP

Задает идентификатор справочной системы

#### *идентификатор*

Строковая константа задающая ключ, который используется для доступа к справочной системе. Это может быть или ключевое слово справочной системы или “строка контекста”.

Атрибут **HLP** задает идентификатор справочной системы для экранного объекта. Как только пользователь нажимает клавишу F1, Windows автоматически высвечивает справку, если та существует. Если пользователь нажимает F1, запрашивая справку, когда на поле переключен фокус, библиотечная процедура использует идентификатор, заданный для этого поля, для поиска файла справочной системы раздела с заданным идентификатором.

Идентификатор может содержать ключевое слово справочной системы или “строку контекста”. Ключевое слово справочной системы представляет собой слово или фразу, которая высвечивается в окне Help Search справочной системы. Если при нажатии пользователем клавиши F1, это ключевое слово определяет только один раздел справки, то файл раскрывается на этом разделе. Если ключевое слово определяет несколько разделов справки, то пользователю раскрывается окно поиска.

“Строка контекста” отличается в идентификаторе знаком тильды (~) спереди, за которым следует уникальный идентификатор (без пробелов), который связан точно с одним справочным разделом. При нажатии пользователем клавиши F1, файл справки раскрывается на конкретном разделе, связанном с данным контекстом. Если тильда отсутствует, то подразумевается, что идентификатор содержит ключевое слово справочной системы.

#### Пример:

Win1            WINDOW

```
ENTRY(@s30),USE(SomeVariable),HLP('~Entry1Help')!Строка контекста в Help'e
ENTRY(@s30),USE(SomeVariable),HLP('Control Two Help')!Ключевое слово Help'a
END
```

## **HSCROLL, VSCROLL, HVSCROLL (наличие линеек скроллинга)**

**HSCROLL**  
**VSCROLL**  
**HVSCROLL**

Атрибуты **HSCROLL**, **VSCROLL**, **HVSCROLL** задают наличие линеек скроллинга у поля типа COMBO, LIST, IMAGE, или TEXT. **HSCROLL** добавляет к полю снизу линейку горизонтального скроллинга, **VSCROLL** добавляет к полю справа линейку вертикального скроллинга, а **HVSCROLL** - обе.

Атрибут **HSCROLL** также употребляется для элемента управления SHEET. Он определяет вывод листов в один ряд, а не в несколько рядов, не взирая на то, как много этих листов. Для перелистывания на каждом краю листа присоединяются кнопки скроллинга вправо и влево (или вверх и вниз).

Атрибуты **HSCROLL**, **VSCROLL** и **HVSCROLL** кроме того допустимы для элемента SPIN и задают отличающиеся от используемого по умолчанию расположения кнопок прокрутки списка: одна над другой, указывающие вверх или вниз. Атрибутом **HSCROLL** кнопки располагаются рядом, указывая влево и вправо. Атрибутом **VSCROLL** кнопки располагаются одна над другой, указывая вверх и вниз.

Линейка вертикального скроллинга позволяет с помощью мыши пролистывать содержимое поля вверх и вниз. Линейка горизонтального скроллинга позволяет с помощью мыши пролистывать содержимое поля влево и вправо. Как только какая-либо часть подлежащих скроллингу данных выходит за границы поля, у поля выводятся линейки скроллинга.

Когда линейка вертикального скроллинга добавляется к окну списка с атрибутом IMM, то она всегда присутствует на экране, даже если список не заполнен. Когда пользователь щелкает на линейке скроллинга, то генерируется событие, но содержимое списка не перемещается, (эту задачу должна выполнять сама прикладная программа. Для того, чтобы определить положение бегунка на линейке скроллинга, можно проверить значение свойства PROP:VscrollPos (от 0 - верх и до 100 - низ).

## **ICON (установить для поля пиктограмму)**

**ICON ( [файл] )**

**ICON**

Задает пиктограмму, которая должна выводиться для поля.

*файл*

Строковая константа или мнемоническое имя указывающее имя файла .ICO или стандартной пиктограммы Windows, подлежащей выводу. Файл .ICO автоматически включается в программу в качестве ресурса.

Атрибут **ICON** задает пиктограмму, которая должна выводиться для поля. Пиктограмма высвечивается на поверхности кнопки. Этот атрибут можно указать для полей типа **BUTTON**, **RADIO**, или **CHECK**. Для полей типа **RADIO** и **CHECK** атрибут **ICON** создает фиксирующиеся нажимные кнопки, которые выглядят нажатыми в состоянии “включено” и отпущенными в состоянии “выключено”.

В файле **EQUATES.C LW** содержатся операторы **EQUATE** для стандартных в Windows пиктограмм. Ниже представлены часть из них (полный список смотри в файле **EQUATES.C LW**).

<b>ICON:None</b>	нет пиктограммы
<b>ICON:Application</b>	
<b>ICON:Question</b>	?
<b>ICON:Exclamation</b>	!
<b>ICON:Asterisk</b>	*
<b>ICON:VCRtop</b>	>>
<b>ICON:VCRbottom</b>	<<
<b>ICON:VCRlocate</b>	?
<b>ICON:VCRrewind</b>	<<<
<b>ICON:VCRback</b>	<
<b>ICON:VCRplay</b>	>
<b>ICON:VCRfastforward</b>	>>>

Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
  OPTION('Option'),USE(OptVar)
  RADIO('Radio 1'),AT(120,0,20,20),USE(?R1),ICON('Radio1.ICO')
  RADIO('Radio 2'),AT(140,0,20,20),USE(?R2),ICON('Radio2.ICO')
END
CHECK('&A'),AT(0,120,20,20),USE(?C7),ICON(ICON:Asterisk)
BUTTON('&1'),AT(120,0,20,20),USE(?B7),ICON(ICON:Question)
END
```

## **IMM (установить немедленное генерирование события)**

### **IMM**

Атрибут **IMM** задает немедленную генерацию события.

Для объекта типа **REGION** с атрибутом **IMM** события генерируются как только курсор

мышь входит в пределы объекта, перемещается внутри или покидает пределы области, заданной атрибутом AT. Точное положение курсора мыши можно определить с помощью функций MOUSEX и MOUSEY.

Для поля BUUTTON атрибут IMM означает, что событие генерируется, когда нажимается левая кнопка мыши, а не когда она отпускается. Событие генерируется непрерывно на протяжении всего времени, в течение которого пользователь удерживает кнопку нажатой.

Для полей COMBO и LIST атрибут IMM указывает, что событие генерируется немедленно и всякий раз когда пользователь нажимает любую клавишу или комбинацию клавиш (обычно используется для повторного построения очереди). Когда пользователь нажимает печатный символ, генерируется событие EVENT:NewSelection. То же самое происходит для полей типа ENTRY или SPIN.

### INS, OVR (установить режим ввода)

**INS**  
**OVR**

Атрибуты **INS** и **OVR** задают режим ввода данных в поля типа ENTRY и TEXT, когда для окна установлен атрибут MASK. INS устанавливает режим вставки, тогда как OVR устанавливает режим забивки. Эти режимы активны только для окон с атрибутом MASK.

### JOIN (установить объединение кнопок прокрутки на листе)

**JOIN**

Атрибут JOIN указывает, что листы выводятся в один ряд, а не в несколько, не взирая на то, сколько листов всего. Для перелистывания на правом (или нижнем) краю листа присоединяются кнопки скроллинга вправо и влево (или вверх и вниз).

### KEY (установить клавишу выполнения элемента)

**KEY(код\_клавиши)**

**KEY**                                      Задаёт “горячую” клавишу для элемента управления.  
*keycode*                                   Код клавиши, принятый в Clagion или мнемоническая метка соответствия кода клавиши.

Атрибут **KEY** задаёт “горячую” клавишу, по которой фокус немедленно переключается на этот элемент и выполняются связанные с ним действия.

Фокус переключается на следующие элементы управления:

**COMBO**

**CUSTOM  
ENTRY  
GROUP  
LIST  
OPTION  
PROMPT  
SPIN  
TEXT**

На следующие элементы и переключается фокус, и выполняются связанные с ним действия:

**BUTTON  
CHECK  
CUSTOM  
RADIO  
MENU  
ITEM**

### Пример:

```
WinOne  WINDOW,AT(0,0,160,400)
        COMBO(@S8),AT(120,120,20,20),USE(?C1),FROM(Q1:F2),KEY(F1Key)
        LIST,AT(120,0,20,20),USE(?L1),FROM(Que1),KEY(F2Key)
        SPIN(@N8.2),AT(280,0,20,20),USE(SpinVar1),FROM(Q),KEY(F3Key)
        TEXT,AT(20,0,40,40),USE(E2),KEY(F4Key)
        PROMPT('Enter &Data in E2:'),AT(10,200,20,20),USE(?P2),KEY(F5Key)
        ENTRY(@S8),AT(100,200,20,20),USE(E2),KEY(F6Key)
        BUTTON('&1'),AT(120,0,20,20),USE(?B7),KEY(F7Key)
        CHECK('&A'),AT(0,120,20,20),USE(?C7),KEY(F8Key)
        OPTION('Option'),USE(OptVar),KEY(F9Key)
        RADIO('Radio 1'),AT(120,0,20,20),USE(?R1),KEY(F10Key)
        RADIO('Radio 2'),AT(140,0,20,20),USE(?R2),KEY(F11Key)
        END
    END
```

## LEFT, RIGHT, ABOVE, BELOW (установить положение листов)

**LEFT( [ширина] )  
RIGHT( [ширина] )  
ABOVE( [ширина] )  
BELOW( [ширина] )**

*ширина* Целочисленная константа определяющая ширину листа. Задается в условных единицах.

Атрибуты LEFT, RIGHT, ABOVE и BELOW элемента управления SHEET задают



расположение составляющих ее листов. Атрибут LEFT определяет, что листы располагаются друг за другом влево, RIGHT - со сдвигом вправо, ABOVE - со сдвигом вверх (расположение по умолчанию), а BELOW - со сдвигом вниз.

Параметр ширина позволяет задать размер листа. Если не заданы атрибуты UP или DOWN, то текст, выводится на листах горизонтально.

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
          SHEET,AT(0,0,320,175),USE(SelectedTab),BELOW
!Разместить листы вниз
    TAB('Tab One'),USE(?TabOne)
    OPTION('Option 1'),USE(OptVar1),KEY(F10Key),HLP('Option1Help')
    RADIO('Radio 1'),AT(20,0,20,20),USE(?R1)
    RADIO('Radio 2'),AT(40,0,20,20),USE(?R2)
    END
    OPTION('Option 2'),USE(OptVar2),MSG('Option 2')
    RADIO('Radio 3'),AT(60,0,20,20),USE(?R3)
    RADIO('Radio 4'),AT(80,0,20,20),USE(?R4)
    END
    PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P1)
    ENTRY(@S8),AT(100,140,32,20),USE(E1)
    PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P2)
    ENTRY(@S8),AT(100,240,32,20),USE(E2)
    END
    TAB('Tab Two'),USE(?TabTwo)
    PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P3)
    ENTRY(@S8),AT(100,140,32,20),USE(E3)
    PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P4)
    ENTRY(@S8),AT(100,240,32,20),USE(E4)
    END
    END
    BUTTON('Ok'),AT(100,180,20,20),USE(?Ok)
    BUTTON('Cancel'),AT(200,180,20,20),USE(?Cancel)
    END
```

## LEFT, RIGHT, CENTER, DECIMAL (установить выравнивание)

```
LEFT([смещение])
RIGHT([смещение])
CENTER([смещение])
DECIMAL([смещение])
```

*смещение*

Целочисленная константа, задающая величину смещения от точки выравнивания. Величина задается в условных единицах.

Атрибуты LEFT, RIGHT, CENTER и DECIMAL задают выравнивание отображаемых данных. LEFT указывает выравнивание на левый край, RIGHT - задает выравнивание по правому краю, CENTER - центрирование, а DECIMAL задает выравнивание числовых данных по десятичной точке.

Параметр смещение атрибута CENTER указывает смещение от центра (отрицательное - влево). Для атрибута DECIMAL параметр смещение задает смещение десятичной точки справа налево.

Для полей BUTTON, CHECK и RADIO допустимы только атрибуты LEFT и RIGHT (без параметра смещение). Для поля TEXT допустимы только атрибуты LEFT(смещение), RIGHT(смещение), и CENTER(смещение).

Поля допускающие атрибуты LEFT(смещение), RIGHT(смещение), CENTER(смещение) и DECIMAL(смещение):

**COMBO**  
**ENTRY**  
**LIST**  
**SPIN**  
**STRING**

### Пример:

```
WinOne WINDOW,AT(0,0,160,400)
        COMBO(@S8),AT(120,120,20,20),USE(?C1),FROM(Q1:F2),RIGHT(4)
        LIST,AT(120,0,20,20),USE(?L1),FROM(Que1),CENTER
        SPIN(@N8.2),AT(280,0,20,20),USE(SpinVar1),FROM(Q),DECIMAL(8)
        TEXT,AT(20,0,40,40),USE(E2),LEFT(8)
        ENTRY(@S8),AT(100,200,20,20),USE(E2),LEFT(4)
        CHECK('&A'),AT(0,120,20,20),USE(?C7),LEFT
        OPTION('Option'),USE(OptVar)
        RADIO('Radio 1'),AT(120,0,20,20),USE(?R1),LEFT
        RADIO('Radio 2'),AT(140,0,20,20),USE(?R2),RIGHT
        END
END
```

**LINEWIDTH (установить толщину линий экранных элементов)****LINEWIDTH**( *толщина* )

**LINEWIDTH**            Задает толщину элемента LINE или толщину контура элементов BOX и ELLIPSE.

*толщина*            Положительная целочисленная константа задает толщину линии в пикселах.

Атрибут **LINEWIDTH** указывает толщину элемента LINE, а для элементов BOX и ELLIPSE - толщину контура.

**Пример:**

```

window    WINDOW('Caption'),AT(.,260,100),GRAY
           LINE,AT(105,78,-49,0),USE(?Line1),LINEWIDTH(3)            !3 pixel line
           BOX,AT(182,27,50,50),USE(?Box1),LINEWIDTH(3)            !Прямоугольник с
толщиной   границы 3 пиксела
           END

```

**LINK (создать элемент управления OLE - связь с объектом в файле)****LINK**( *имя\_файла* )

**LINK**            Задает создание для элемента управления OLE связи с объектом в файле данных, характерном для прикладной программы - сервера.

*имя\_файла*            Строковая константа, содержащая имя файла

Атрибут LINK задает создание для элемента управления OLE связи с объектом в файле данных, характерном для прикладной программы - сервера. Значение параметра имя\_файла должно представлять собой полное имя файла, включая путь, если только он не находится в том же каталоге, что и прикладная программа - сервер.

**Пример:**

```

WinOne    WINDOW,AT(0,0,200,200)
           OLE,AT(10,10,160,100),USE(?OLEObject),LINK('Book1.XLS')
!Таблица Excel
           MENUBAR
           MENU('&Clarion App')
           ITEM('&Deactivate Object'),USE(?DeactOLE)
           END
           END
           END
           END

```

## MARK (установить режим множественного выбора)

### MARK (*признак*)

**MARK** Включает режим выбора нескольких элементов.

*признак* Метка поля из структуры QUEUE.

Атрибут MARK включает режим выбора нескольких элементов из списка поля LIST или COMBO. Когда выбирается какой-либо элемент из списка, соответствующий ему признак устанавливается в значение “истина” (1). Каждый помеченный элемент списка автоматически выделяется. Изменение значения поля - признака также изменяет изображение соответствующего элемента списка.

Если для поля LIST или COMBO указан атрибут MARK, то нельзя задавать атрибут IMM.

Пример:

```

Que1      QUEUE
MarkFlag  BYTE
F1        LONG
F2        STRING(8)
          END
WinOne    WINDOW,AT(0,0,160,400),SYSTEM
          LIST,AT(120,0,20,20),USE(?L1),FROM(Que1.F1),MARK(Que1.MarkFlag)
          COMBO(@S8),AT(120,120,,),USE(?C1),FROM(Que1.F2),MARK(Que1.MarkFlag)
          END

CODE
DO LoadQue      !Заполнить очередь Que1 данными
OPEN(WinOne)
ACCEPT
CASE EVENT()
OF EVENT:CloseWindow
  BREAK
END
END
LOOP X# = 1 to RECORDS(Que1)      !Цикл по элементам очереди
  GET(Que1,X#)
  IF Que1.MarkFlag      !Если пользователь пометил элемент,
    DO ProcessMarked    ! то обработать его
  END
END
END

```

**MSG (установить сообщение для поля)****MSG**(*текст*)**MSG**

Задаёт текст, который должен выводиться в строке сообщений.

*текст*

Строковая константа, содержащая сообщение, которое должно выводиться в строке состояния.

Атрибут **MSG** указывает текст, подлежащий выводу в первой области строки состояния, когда на поле переключен фокус ввода.**Пример:**

```
WinOne WINDOW,AT(0,0,160,400)
COMBO(@S8),AT(120,120,20,20),USE(?C1),FROM(Q1:F2),MSG('Enter or Select')
LIST,AT(120,0,20,20),USE(?L1),FROM(Que1),MSG('Select One')
SPIN(@N8.2),AT(280,0,20,20),USE(SpinVar1),FROM(Q),MSG('Choose One')
TEXT,AT(20,0,40,40),USE(E2),MSG('Enter Text')
ENTRY(@S8),AT(100,200,20,20),USE(E2),MSG('Enter Data')
CHECK('&A'),AT(0,120,20,20),USE(?C7),MSG('On or Off')
OPTION('Option 1'),USE(OptVar),MSG('Pick One or Two')
RADIO('Radio 1'),AT(120,0,20,20),USE(?R1)
RADIO('Radio 2'),AT(140,0,20,20),USE(?R2)
END
OPTION('Option'),USE(OptVar)
RADIO('Radio 1'),AT(120,40,20,20),USE(?R1),MSG('Pick One')
RADIO('Radio 2'),AT(140,40,20,20),USE(?R2),MSG('Pick Two')
END
END
```

**NOBAR (отсутствие полосы - курсора)****NOBAR**Атрибут **NOBAR** указывает, что выбранный в данный момент элемент выделяется только когда на окно списка переключен фокус ввода.**NOSHEET (задать “плавающий” лист)****NOSHEET**Атрибут **NOSHEET** структуры **SHEET** указывает, что листы выводятся неупорядоченными. Тем самым создается эффект “плавающих” листов.**OPEN (открыть из файла объект для поля OLE)****OPEN**( *объект* )**OPEN**

Указывает, что для поля OLE необходимо открыть объект из

специального файла (OLE Compound Storage).

*объект* Строковая константа, содержащая имя файла и объекта в нем, который следует открыть.

Атрибут **OPEN** указывает, что необходимо открыть для поля OLE сохраненный в специальном файле объект. При открытии объекта восстанавливается сохраненный вариант значений свойств контейнера, поэтому нет необходимости их задавать вновь. Синтаксис параметра объект должен иметь вид: ИмяФайла\!ИмяОбъекта.

### Пример:

```
WinOne    WINDOW,AT(0,0,200,200)
          OLE,AT(10,10,160,100),USE(?OLEObject),OPEN('SavFile.OLE\!MyObject')
          MENUBAR
          MENU('&Clarion App')
          ITEM('&Deactivate Object'),USE(?DeactOLE)
          END
          END
          END
          END
```

## PASSWORD (установить неотображаемое поле)

### PASSWORD

Атрибут **PASSWORD** устанавливает, что вводимые в поле ENTRY данные не отображаются. Когда пользователь вводит данные, для каждого из вводимых символов в поле выводится звездочка (\*).

## RANGE (установить границы диапазона)

### RANGE(*начало, конец*)

**RANGE** Задает допустимый диапазон значений данных, которые пользователь может выбрать в поле SPIN или диапазон значений, отображаемый в поле . PROGRESS.

*начало* Числовая константа, которая указывает нижнюю границу (включительно) диапазона допустимых значений.

*конец* Числовая константа, которая указывает верхнюю границу (включительно) диапазона допустимых значений.

Атрибут **RANGE** задает допустимый диапазон значений данных, которые пользователь может выбрать в поле SPIN. А также он определяет диапазон значений, который отображается в поле PROGRESS. Этот атрибут работает совместно с атрибутом STEP, обеспечивая пользователя в поле SPIN значениями для выбора. Когда используются

атрибуты RANGE и STEP, атрибут FROM использоваться не может.

### Пример:

```
WinOne    WINDOW,AT(0,0,160,400)
          SPIN(@N4.2),AT(280,0,20,20),USE(SpinVar1),RANGE(.05,9.95),STEP(.05)
          SPIN(@n3),AT(280,0,20,20),USE(SpinVar2),RANGE(5,995),STEP(5)
          END
```

## READONLY (установить поле только для вывода данных)

### READONLY

Атрибут **READONLY** указывает, что в поле типа COMBO, ENTRY, SPIN или TEXT данные только выводятся. С помощью мыши на поле можно переключить фокус, но вводить данные нельзя. Если пользователь попытается изменить отображаемые данные, то звуковой сигнал предупредит его о невозможности ввода данных в это поле.

## REQ (установить обязательное поле)

### REQ

Атрибут **REQ** указывает, что поле ENTRY или TEXT нельзя оставить пустым или нулевым. Значения полей ENTRY и TEXT с атрибутом REQ, не проверяются до тех пор, пока не будет нажата кнопка BUTTON, имеющая атрибут REQ, или не произойдет обращение к функции INCOMPLETE().

Когда нажата кнопка BUTTON, имеющая атрибут REQ, или произошло обращение к функции INCOMPLETE(), проверяются все поля типа ENTRY или TEXT с атрибутом REQ, чтобы убедиться в том, что все они содержат данные. Фокус ввода переключается на первое выявленное при этой проверке пустое поле.

## ROUND (установить скругление углов у поля BOX)

### ROUND

Атрибут **ROUND** указывает, что поле BOX имеет скругленные углы.

## SCROLL (установить прокручиваемое поле)

### SCROLL

Атрибут **SCROLL** указывает, что, когда происходит скроллинг окна, поле перемещается вместе с окном. Это позволяет создавать “виртуальные” окна, размером больше чем физический экран. Наличие атрибута SCROLL означает, что положение поля зафиксировано относительно левого верхнего угла виртуального окна, независимо от того,

находится это место в поле зрения или нет. Это значит, что по мере скроллинга окна поле появляется на экране. Если атрибут SCROLL опущен, то поле фиксируется в окне относительно верхнего левого угла видимой части окна. Это означает, что поле сохраняет положение на экране, в то время, когда остальные части окна перемещаются. Это полезно для объектов, которые должны всегда оставаться в поле зрения пользователя (как например кнопки OK и Cancel). Сочетание в одном окне объектов, имеющих атрибут SCROLL и не имеющих его, может привести к наложению изображения нескольких объектов. Это произойдет вследствие того, что объекты с атрибутом SCROLL перемещаются, а не имеющие этого атрибута остаются на месте. Ситуация наложения объектов является временной и дальнейший скроллинг устраняет ее. Кроме того, наложения при скроллинге можно избежать продуманно размещая поля в окне. Например можно разместить все поля без атрибута SCROLL в нижней части окна, а затем над ними разместить все поля с атрибутом SCROLL располагая их выше и левее или правее. Таким образом следует создавать окно, которое без проблем прокручивается только горизонтально.

### **SINGLE (поле TEXT для ввода в одну строку)**

#### **SINGLE**

Атрибут **SINGLE** указывает, что данные в этот элемент вводятся в одну строку. Это позволяет использовать элемент управления TEXT вместо ENTRY в языках, в которых текст принято записывать справа налево (как иврит и арабские языки).

### **SKIP (установить пропуск поля при выборе клавишей Tab)**

Атрибут **SKIP** указывает, что пользователь может обратиться к объекту только с помощью мыши или клавиш ускоренного доступа, но не клавишей Tab. Поле, которое подразумевает ввод в него данных, сохраняет фокус ввода только на протяжении ввода данных и не сохраняет фокус после окончания его. На поля, которые не подразумевают ввода данных фокус не переключается. Действие этого атрибута предназначено для того, чтобы организовать такое поведение объектов как на панели инструментов. Когда курсор мыши располагается в границах объекта, имеющего атрибут SKIP, в строке состояния высвечивается текст, заданный атрибутом MSG этого объекта.

### **SPREAD (установить равномерные промежутки между листами)**

#### **SPREAD**

Атрибут **SPREAD** листы TAB в структуре SHEET располагаются через равный промежуток друг от друга.



**STD (установить стандартное действие)****STD** (*действие*)

**STD**                                Задает стандартное в Windows действие.  
*действие*                        Целочисленная константа или мнемоническое имя, указывающее идентификатор стандартного действия в Windows.

Атрибут **STD** указывает, что поле выполняет некоторое стандартное в Windows действие. Это действие автоматически выполняется библиотечной процедурой и для этого поля не генерируется никаких событий.

Операторы EQUATE для мнемонических имен, обозначающих стандартные в Windows действия, содержатся в файле EQUATES.CLW. Пример этих операторов (полный список см. в файле EQUATES.CLW :

STD:WindowList    Список открытых MDI окон  
 STD:TileWindow    Расположить окна рядом  
 STD:CascadeWindow    Расположить окна каскадом  
 STD:ArrangeIcons    Упорядочить пиктограммы  
 STD:HelpIndex    Оглавление справочной системы  
 STD:HelpSearch    Окно поиска в справочной системе

Пример:

```
MDIChild    WINDOW('Child One'),MDI,SYSTEM,MAX
             MENUBAR
             MENU('Edit'),USE(?EditMenu)
             ITEM('Undo'),USE(?UndoText),KEY(CtrlZ),STD(STD:Undo)
             ITEM('Cu&t'),USE(?CutText),KEY(CtrlX),STD(STD:Cut)
             ITEM('Copy'),USE(?CopyText),KEY(CtrlC),STD(STD:Copy)
             ITEM('Paste'),USE(?PasteText),KEY(CtrlV),STD(STD:Paste)
             END
             END
             TOOLBAR

BUTTON('Cut'),USE(?CutButton),ICON(ICON:Cut),STD(STD:Cut)
            BUTTON('Copy'),USE(?CopyButton),ICON(ICON:Copy),STD(STD:Copy)
            BUTTON('Paste'),USE(?PasteButton),ICON(ICON:Paste),STD(STD:Paste)
            END
            END
```

## STEP (установить приращение значения в поле SPIN)

### STEP(*приращение*)

**STEP**                      Задает приращение текущего значения в поле SPIN.  
*приращение*              Числовая константа, указывающая величину приращения.

Атрибут **STEP** задает величину на которую увеличивается или уменьшается значение поля SPIN в пределах диапазона допустимых значений. По умолчанию значение приращения равно 1.0

#### Пример:

```
WinOne   WINDOW,AT(0,0,160,400)
          SPIN(@N4.2),AT(280,0,20,20),USE(SpinVar1),RANGE(.05,9.95),STEP(.05)
          SPIN(@N3),AT(280,0,20,20),USE(SpinVar2),RANGE(5,995),STEP(5)
          SPIN(@T3),AT(280,0,20,20),USE(SpinVar3),RANGE(1,8640000),STEP(6000)
          END
```

## STRETCH (растягивание объекта OLE)

### STRETCH

Атрибут **STRETCH** указывает, что объект OLE растягивается таким образом, чтобы заполнять всю область, определенную атрибутом AT поля-контейнера. При этом пропорции объекта не сохраняются.

## TIP (установить текст “возникающей” подсказки)

### TIP(*строка*)

**TIP**                      Указывает, что строка выводится как “возникающая” подсказка, когда курсор мыши задерживается на экранном объекте.  
*строка*                      Строковая константа, которая задает выводимый текст.

Атрибут **TIP** какого-либо экранного объекта, задает текст, который выводится как “возникающая” подсказка, когда курсор мыши задерживается на этом объекте. Хотя явного ограничения на длину строки нет, она должна все же помещаться на экране.

Хотя этот атрибут допустим для любого экранного объекта, на который может переключаться фокус, чаще всего он используется с кнопками **BUTTON**, имеющими атрибут **ICON** и располагающимися на панели инструментов. Текст “возникающей” подсказки позволяет пользователю быстро определить назначение кнопки, не обращаясь к системе диалоговой помощи.

Автоматический вывод текста, задаваемого атрибутом TIP, для любого отдельного экранного объекта можно отменить, установив PROP:NoTips, необъявленное свойство в единицу (1). Можно отменить автоматический вывод подсказки и для всего приложения, установив в единицу свойство PROP:NoTips для встроенной переменной SYSTEM.

Длительность паузы, по истечении которой выводится подсказка, можно задать для всего приложения, установив свойство PROP:TipDelay для встроенной переменной SYSTEM равным требуемому значению (в сотых долях секунды). Это справедливо только для 16-ти разрядных приложений; в 32-х разрядных операционных системах величина этой задержки является настраиваемым параметром самой операционной системы и устанавливается пользователем.

### Пример:

```
WinOne WINDOW,AT(0,0,160,400)
        TOOLBAR
        BUTTON('E&xit'),USE(?MainExitButton),ICON(ICON:hand),TIP('Exit Window')
        BUTTON('&Open'),USE(?OpenButton),ICON(ICON:Open),TIP('Open a File')
        END
        COMBO(@S8),AT(120,120,20,20),USE(?C1),FROM(Q1:F2)
        ENTRY(@S8),AT(100,200,20,20),USE(E2)
        END
```

## TRN (установить вывод элемента на прозрачном фоне)

Атрибут **TRN** элемента управления указывает, что символы выводятся на “прозрачном” фоне, не изменяя общего фона, на котором располагается поле. На экран выводятся только точки, необходимые для формирования символа. Это позволяет выводить строку поверх поля IMAGE, не нарушая фонового изображения.

### Пример:

```
WinOne WINDOW,AT(0,0,160,400)
        IMAGE('PIC.BMP'),USE(?I1),FULL !Картинка во все окно
        STRING('String Constant'),AT(10,0,20,20),USE(?S1),TRN
                                           !Строка на прозрачном фоне
        END
```

## UP, DOWN (установить ориентацию ярлычков листов)

**UP**  
**DOWN**

Атрибуты **UP** и **DOWN** элемента управления SHEET задают ориентацию заголовка

элемента TAB. UP указывает, что значение параметра текст на листах выводится вертикально для чтения снизу вверх, тогда как DOWN определяет, что текст читается сверху вниз. Если указаны и UP, и DOWN, то текст выводится “вверх ногами”.

**Пример:**

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
SHEET,AT(0,0,320,175),USE(SelectedTab),RIGHT,DOWN
TAB('Tab One'),USE(?TabOne)
PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P1)
ENTRY(@S8),AT(100,140,32,20),USE(E1)
PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P2)
ENTRY(@S8),AT(100,240,32,20),USE(E2)
END
PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P3)
TAB('Tab Two'),USE(?TabTwo)
ENTRY(@S8),AT(100,140,32,20),USE(E3)
PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P4)
ENTRY(@S8),AT(100,240,32,20),USE(E4)
END
END
BUTTON('Ok'),AT(100,180,20,20),USE(?Ok)
BUTTON('Cancel'),AT(200,180,20,20),USE(?Cancel)
END
```

**USE (задать для поля метку соответствия или переменную)**

USE( <i>метка</i> [ <i>,номер</i> ] [ <i>метка соответствия</i> ]   <i>переменная</i> )	
--	--

<b>USE</b>	Задает переменную или метку соответствия поля.
<i>метка</i>	Мнемоническая метка соответствия, предназначенная для того, чтобы ссылаться на это поле в исполняемых операторах.
<i>переменная</i>	Переменная, принимающая введенное в поле значение. Если не задан параметр метка соответствия, то метка этой переменной (со знаком вопроса спереди) становится меткой соответствия для экранного объекта.
<i>номер</i>	Целочисленная константа, которая указывает номер присваиваемый компилятором метке соответствия для этого поля.
<i>метка соответствия</i>	Мнемоническая метка соответствия объекта, которая служит для того, чтобы в исполняемых операторах ссылаться на данный объект, когда имя переменной уже использовалось в этой экранной структуре. Дополнительная метка соответствия обеспечивает уникальность меток соответствия полей, тогда как использование USE-переменных этого не обеспечивает.

Атрибут **USE** задает для экранного поля переменную или метку соответствия. Атрибут с параметром метка просто обеспечивает способ, с помощью которого на это поле можно ссылаться в исполняемых операторах. Для некоторых полей в качестве параметра атрибута **USE** допускается только метка соответствия, но не имя переменной. Это такие поля как: **PROMPT**, **IMAGE**, **LINE**, **BOX**, **ELLIPSE**, **GROUP**, **RADIO**, **REGION**, **MENU**, и **BUTTON**. Атрибут **USE** обеспечивает для поля переменную, в которую заносятся введенные пользователем в поле данные. Это относится к следующим полям: **ITEM** с атрибутом **CHECK**, и полям **ENTRY**, **OPTION**, **SPIN**, **TEXT**, **LIST**, **COMBO**, **CHECK**, и **CUSTOM**.

Всем полям в окне **APPLICATION** или **WINDOW** компилятором автоматически присваиваются номера. Для полей на линейке меню окна **APPLICATION** эти номера начинаются с минус единицы (-1) и уменьшаются на единицу для каждого поля **MENU** или **ITEM** на линейке. В окне **WINDOW** нумерация начинается с 1 и увеличивается с шагом 1 для каждого следующего поля.

Параметр номер атрибута **USE** позволяет задать номер, назначаемый компилятором экранному объекту. Это число используется также в качестве начальной точки отсчета в дальнейшей нумерации полей, не имеющих в атрибуте **USE** параметра номер. Последующие поля не имеющие в атрибуте **USE** параметра номер нумеруются с приращением (или уменьшением) относительно последнего назначенного номера.

Для двух или более полей с одинаковым значением атрибута **USE** в одном окне **APPLICATION** или **WINDOW** создадутся одинаковые метки соответствия полей; когда компилятор обнаруживает такую ситуацию, он удаляет для всех этих полей метки соответствия. Предупреждая возникновение неоднозначности относительно того, на какое поле в действительности делается ссылка, компилятор делает невозможными ссылки на эти поля в исполняемых операторах. Однако такой подход позволяет сознательно создавать такую ситуацию для того, чтобы вывести содержимое одной переменной в нескольких полях с разными шаблонами.

### Пример:

```
WinOne   WINDOW,AT(0,0,160,400)
          COMBO(@S8),AT(120,120,20,20),USE(?C1),FROM(Q1:F2)
          ENTRY(@S8),AT(100,200,20,20),USE(E2)
          END
```

## VALUE (значение, присваиваемое USE-переменной для RADIO или CHECK)

VALUE(	строка	)
	истинн._значение , ложное_значение	

**VALUE**                      Задает значение, присваиваемое USE-переменной структуры **OP-**

TION, когда пользователь включает данную кнопку RADIO или значения, назначаемые USE-переменной элемента CHECK, во включенном и выключенном состоянии.

*строка* Строковая константа, содержащая значение, присваиваемое USE-переменной структуры OPTION.

*истинн.\_значение* Строковая константа, содержащая значение, назначаемое USE-переменной элемента CHECK, во включенном состоянии.

*ложное\_значение* Строковая константа, содержащая значение, назначаемое USE-переменной элемента CHECK, в выключенном состоянии.

Атрибут **VALUE** в структуре OPTION задает значение, которое автоматически присваивается USE-переменной этой структуры, когда пользователь включает данную кнопку RADIO. В этом случае значение этого атрибута используется вместо параметра текст кнопки RADIO.

В элементе управления CHECK этот атрибут задает значения, назначаемые USE-переменной элемента CHECK, во включенном и выключенном состоянии. Они переопределяют используемые по умолчанию ноль и единицу

К значениям, присваиваемым USE-переменной, применимы все правила автоматического преобразования типов. Следовательно, если строка, истинн.\_значение или ложное\_значение содержит допустимые для числовых данных символы и USE-переменная числового типа, то ей присваивается числовое значение.

### Пример:

```
Win WINDOW,AT(0,0,180,400)
  OPTION('Option 1'),USE(OptVar1),MSG('Pick One or Two')
  RADIO('Radio 1'),AT(120,0,20,20),USE(?R1),VALUE('10')!B OptVar1 заносится 10
  RADIO('Radio 2'),AT(140,0,20,20),USE(?R2),VALUE('20')!B OptVar1 заносится 20
  END
  OPTION('Option 2'),USE(OptVar2),MSG('Pick One or Two')
  RADIO('Radio 1'),AT(120,0,20,20),USE(?R1),VALUE('10')!B OptVar2 заносится '10'
  RADIO('Radio 2'),AT(140,0,20,20),USE(?R2),VALUE('20')
  !BOptVar2 заносится'20'
  END
  CHECK('Check 1'),AT(160,0),USE(Check1),VALUE('T','F')
  END
```

## VCR (установить кнопки управления как у видеомаягнитофона)

**VCR**([поле])

**VCR** Устанавливает для поля типа LIST или COMBO кнопки управления списком, подобные клавишам управления видеомаягнитофоном.

*поле* Метка соответствия поля, которая задает поле типа ENTRY, которое

используется для ввода поискового значения - локатора поля типа LIST или COMBO.

Атрибут **VCR** задает для поля типа LIST или COMBO кнопки управления списком, подобные клавишам управления видеомэгнитофоном. Кнопки в стиле управления видеомэгнитофоном влияют на прокрутку данных в окне списка LIST или COMBO.

Шесть кнопок, высвечиваемых атрибутом VCR:

<	На начало списка	(EVENT:ScrollTop)
<<	На страницу вверх	(EVENT:PageUp)
<	На строку вверх	(EVENT:ScrollUp)
>	На строку вниз	(EVENT:ScrollDown)
>>	На страницу вниз	(EVENT:PageDown)
>	На конец списка	(EVENT:ScrollBottom)

Кроме того, для атрибут VCR(поле) выводит посередине между другими кнопками кнопку со знаком вопроса (?). Это кнопка поиска, которая переключает фокус на поле, указанное параметром поле. Когда пользователь вводит данные и затем нажимает клавишу TAB в поисковом поле, список в окне устанавливается на наиболее соответствующее значение.

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
LIST,AT(140,0,20,20),USE(?L8),FROM(Que),HVSCROLL,VCR
ENTRY(@S8),AT(100,200,20,20),USE(E2)
LIST,AT(140,100,20,20),USE(?L8),FROM(Que),HVSCROLL,VCR(?E2)
END
```

## WIZARD (листы не переключаются клавишей Tab)

### WIZARD

Атрибут **WIZARD** указывает, что все листы данного набора сразу не выводятся. Это позволяет программе управлять переходом пользователя от листа к листу в заданном порядке (обычно с помощью кнопок "Next" и "Previous").

## ZOOM (пропорциональное изменение размеров объекта OLE)

### ZOOM

Атрибут **ZOOM** указывает, что объект OLE пропорционально изменяет размеры, заполняя пространство элемента-контейнера.





## **Глава 8 Обработка событий**

### **Создание программ, выполняемых под управлением событий**

Выполнение программы в среде Windows происходит под воздействием возникающих событий. Событие возникает, например, тогда, когда пользователь щелкнул мышью на управляющем поле экрана или нажал клавишу на клавиатуре. Каждое действие пользователя в процессе выполнения программы влечет посылку сообщения от Windows к программе с информацией о том, что пользователь сделал. Посылка сообщения играет роль события, и у программы появляется возможность обработать это событие соответствующим образом. Теперь становится очевидным, что концепция программирования Windows прямо противоположна концепции DOS - операционная система (Windows) предписывает программе что последней нужно делать, а не наоборот (DOS).

Если нет “под рукой” языка Clarion, то написание Windows-программ становится весьма сложным занятием, поскольку необходимо явным образом кодировать обработку всех сообщений, посылаемых Windows. Явным образом придется кодировать и общие процедуры, например, восстановление графического образа при закрытии окна.

Такие ситуации могут быть обработаны автоматически. Для этого нужно написать стандартные процедуры обработки и в случае необходимости вызывать их на исполнение. Не будь языка Clarion, процедуры пришлось бы писать самостоятельно. В Clarion для Windows они уже написаны и находятся в библиотеке времени исполнения. Следовательно, в языке Clarion есть графические команды, благодаря которым отпадает необходимость в явном восстановлении графических образов (чего нет в других языках).

В Clarion-программах большинство сообщений от Windows автоматически обрабатывается обработчиком событий АСCEPT. Это - стандартные сообщения, и обрабатываются они процедурами библиотеки времени исполнения. Clarion-программе передаются только те события, которые требуют явного ее вмешательства. Это позволяет отбросить заботы, связанные с “нудным” кодированием на нижнем уровне, и сосредоточиться на проблемах верхнего уровня.

Существуют два вида событий, которые передаются программе оператором АСCEPT: события связанные с полем, и события не связанные с полем.

Событие связанное с полем возникает при нажатии клавиши, требующей от программы выполнения определенного действия, связанного с полем.

Событие не связанное с полем не имеет отношение к полю, но требует выполнения некоторого действия со стороны программы (например, закрыть окно, завершить программу или переключиться на другой процесс). Многие события этого вида переводят систему в

модальное состояние, т. к. для продолжения программы необходимо принять то или иное решение.

## АССЕРТ (обработчик событий)

**АССЕРТ**  
*операторы*  
**END**

**АССЕРТ**                                      Обработчик события  
*операторы*                                      операторы программы

Цикл **АССЕРТ** - это обработчик событий, предназначенный для обслуживания событий, порожденных Windows для структур APPLICATION и WINDOW. Окно и АССЕРТ-цикл тесно связаны друг с другом - при открытии окна последующий АССЕРТ-цикл будет обрабатывать все события именно для данного окна.

По выполнению оператор **АССЕРТ** сходен с оператором цикла LOOP - внутри него можно использовать операторы BREAK и CYCLE. АССЕРТ-цикл выполняется всякий раз, когда возникающее событие требует определенных действий со стороны программы. АССЕРТ ожидает события от процедур библиотеки времени исполнения с тем, чтобы обработать его посредством исполнения “своих” операторов. Пока АССЕРТ ждет события, управление передается процедурам библиотеки времени исполнения для автоматической обработки стандартных Windows-событий - тех, которые не требуют программных действий (например, восстановление графического образа экрана).

Текущие значения USE-переменных всех STRING-полей (в верхнем окне каждого процесса) автоматически отображаются на экран всякий раз, когда программа возвращается в начало АССЕРТ-цикла. Следовательно, отпадает необходимость в явном исполнении оператора DISPLAY, чтобы обновить экран данными, предназначенными только для отображения. Если не было установлено свойство PROP:Auto, отвечающее за автоматическое отображение всех USE-переменных во время последующей итерации АССЕРТ-цикла, то значения USE-переменной поля любого другого типа автоматически отображаются на экран при возникновении событий, связанных с этим полем.

Внутри АССЕРТ-цикла программа уточняет какое событие произошло, используя следующие функции:

**EVENT()**                                      возвращает значение, показывающее что произошло. В файле EQUATES.CLW находятся символические константы для событий.  
**FIELD()**                                      возвращает номер поля, с которым связано событие, если это - событие связанное с полем  
**ACCEPTED()**                                      возвращает номер поля, с которым связано событие, если

последнее является EVENT:Accepted событием.

**SELECTED()** возвращает номер поля, с которым связано событие, если последнее является EVENT:Selected событием.

**FOCUS()** возвращает номер поля владеющего фокусом; тип события не играет роли.

**MOUSEX()** возвращает x-координату курсора мыши.

**MOUSEY()** возвращает y-координату курсора мыши.

Есть два события, которые влекут за собой неявное выполнение оператора BREAK в ACCEPT-цикле: сигналы на закрытие окна (EVENT:CloseWindow) и на выход из программы (EVENT:CloseDown).

Программа не обязана отслеживать эти события, поскольку они отрабатываются автоматически. Однако, иногда приходится к этим событиям “привязывать” выполнение некоторых специфических действий, например, отображение на экран окна “You sure?” (“Вы уверены?”) или отработку внутренних ситуаций. Если здесь воспользоваться оператором CYCLE, то управление вернется к началу ACCEPT-цикла без закрытия окна или выхода из программы.

Аналогичным образом оператор CYCLE завершает действие событий: EVENT:PreAlertKey, EVENT:Move, EVENT:Size, EVENT:Restore, EVENT:Maximize и EVENT:Iconize. При возникновении одного из них CYCLE завершает действие события и запрещает порождение события, связанного с первым, - соответственно: EVENT:AlertKey, EVENT:Moved, EVENT:Sized, EVENT:Restored, EVENT:Maximized, EVENT:Iconized.

### Пример:

```
CODE
OPEN(Window)
ACCEPT                                !Обработчик событий
CASE FIELD()
OF 0                                  !Обработать не связанные с полем события
CASE EVENT()
OF EVENT:Move
CYCLE                                !Запретить пользователю перемещать окно
OF EVENT:Suspend
CASE FOCUS()
OF ?field1                            !Что-то сохранить
END
OF EVENT:Resume                       !Восстановить сохраненное
END
OF ?Field1                            !Обработать события поля Field1
CASE EVENT()
OF EVENT:Selected                     ! предварительные действия, связанные с
field1
```

```

OF EVENT:Accepted                ! завершающие действия для field1
END
OF ?Field2                       !Обработать события поля Field2
CASE EVENT()
OF EVENT:Selected                ! предварительные действия, связанные с field2
OF EVENT:Accepted                ! завершающие действия для field2
END
END

```

**Смотри также:** EVENT, FIELD, FOCUS, ACCEPTED, SELECTED, CYCLE

## ALERT (установка клавиши, порождающей событие)

**ALERT**([начальный код][,конечный код])

**ALERT** Определяет клавиши, которые порождают события.  
*начальный код* Числовой код или символическое имя клавиши. Может выступать в роли нижней границы при указании группы кодов клавиш.  
*конечный код* Код или символическое имя клавиши верхней границы группы кодов клавиш.

**ALERT** определяет клавишу или группу клавиш в качестве порождающих событие. Когда пользователь нажимает одну из клавиш группы, то порождаются два события не связанные с полем: EVENT:PreAlertKey и EVENT:AlertKey. Если при обработке события EVENT:PreAlertKey программа выполняет оператор CYCLE, то “приостанавливается” событие EVENT:AlertKey, что предотвращает библиотечные операции по обработке нажатия заданных для окна ALERT-клавиш.

Оператор **ALERT** без параметров отменяет ранее установленные ALERT-клавиши. Любой посылаемый клавиатурой код может быть использован в качестве параметра оператора ALERT. Из-за отсутствия связи с каким-либо управляющим полем, ALERT-оператор порождает события не связанные с полем. При возникновении события, связанного с нажатием ALERT-клавиши, не происходит автоматического изменения USE-переменной поля, владеющего фокусом ввода (если требуется, нужно использовать UPDATE).

### Пример:

```

Screen WINDOW,ALRT(F10Key),ALRT(F9Key) !F10 и F9 - ALERT-клавиши
LIST,AT(109,48,50,50),USE(?List),FROM(Que),IMM
BUTTON('Ok'),AT(111,108,,),USE(?Ok)
BUTTON('&Cancel'),AT(111,130,,),USE(?Cancel)
END
CODE
ALERT                                !Отмена ALERT- клавиш
ALERT(F1Key,F12Key)                 !Все функциональные - ALERT-клавиши

```

ALERT(279)	!Ctrl-Esc - ALERT-"клавиша"
OPEN(Screen)	
ACCEPT	
CASE EVENT()	
OF EVENT:PreAlertKey	!Предпроверка ALERT-событий
IF KEYCODE() = F4Key	!"Вырезать" F4 из ALERT-группы
CYCLE	!"Приостановить" ALERT-обработку
OF EVENT:AlertKey	!Обработка ALERT-событий
CASE KEYCODE()	
OF 279	!Проверка на Ctrl+Esc
BREAK	
OF F9Key	!Проверка на F9
F9HotKeyProc	!Вызов процедуры "горячей" клавиши
OF F10Key	!Проверка на F10
F10HotKeyProc	!Вызов процедуры "горячей" клавиши
END	
END	
END	

Смотри также: UPDATE

## EVENT (возвратить номер события)

### EVENT()

Функция **EVENT** возвращает число, которое определяет причину, побудившую оператор **ACCEPT** предупредить программу о том, что что-то произошло и требуется ее вмешательство. В файле **EQUATES.CW** определены символические имена для всех событий, какие могут быть обработаны программой.

Оператор **ACCEPT** порождает события двух типов: события связанные с полем и события не связанные с полем. События связанные с полем воздействуют на отдельное управляющее поле, в то время как события не связанные с полем воздействуют на окно или программу. Тип события можно определить исходя из значений, возвращаемых функциями **ACCEPTED**, **SELECTED** и **FIELD**. Если требуется знать какое поле владеет фокусом ввода в момент возникновения события не связанного с полем, то нужно использовать функцию **FOCUS**.

#### Для событий связанных с полем:

Функция **FIELD** возвращает номер управляющего поля в котором произошло событие. Функция **ACCEPTED** возвращает номер поля, если событие - **EVENT:Accepted**. Функция **SELECTED** возвращает номер поля, если событие - **EVENT:Selected**.

#### Для событий не связанных с полем:

Функции **FIELD**, **ACCEPTED**, **SELECTED** все возвращают нулевое значение (0).

Тип возвращаемых данных: SHORT

### Пример:

```
ACCEPT
CASE EVENT()
  OF EVENT:Selected
    CASE SELECTED()
      OF ?Control1      !Подготовительные действия
      OF ?Control2      !Подготовительные действия
    END
  OF EVENT:Accepted
    CASE ACCEPTED()
      OF ?Control1      !Завершающие действия
      OF ?Control2      !Завершающие действия
    END
  OF EVENT:Suspend      !Что-то сохранить
  OF EVENT:Resume       !Восстановить сохраненное
..
```

## POST (послать событие определенное пользователем)

**POST**(*событие*[,*поле*][,*процесс*])

### POST

Послать событие

*событие*

Целочисленная константа, переменная, выражение или символическое имя, содержащие номер события. Значение в пределах от 400h до 0FFFh - событие определенное пользователем.

*поле*

Целочисленная константа, символическое имя, переменная или выражение, указывающие номер управляющего поля, на которое воздействует событие. Если параметр опущен, то посылаемое событие - не связанное с полем.

*процесс*

Целочисленная константа, символическое имя, переменная или выражение, указывающие номер процесса, АСCEPT-цикл которого должен обработать данное событие. Если параметр опущен, то событие посылается текущему процессу.

Оператор **POST** посылает событие активному в данный момент АСCEPT-циклу указанного процесса. Все события можно разбить на два класса: события, определяемые пользователем, все прочие события. Номера событий определяемых пользователем задаются любым целым числом в пределах от 400h до 0FFFh. Любое событие, посылаемое с указанием поля, - событие связанное с полем, а посылаемое без указания - событие не связанное с полем.

**Пример:**

```
Win1 WINDOW('Tools'),AT(156,46,32,28),TOOLBOX
    BUTTON('Date'),AT(0,0,,),USE(?Button1)
    BUTTON('Time'),AT(0,14,,),USE(?Button2)
END
CODE
OPEN(Win1)
ACCEPT
    IF EVENT() = EVENT:User THEN BREAK.
        !Выделить событие определяемое пользователем
    CASE ACCEPTED()
    OF ?Button1
        POST(EVENT:User,,UseToolsThread)
        !Послать событие не связанное с полем другому процессу
    OF ?Button2
        POST(EVENT:User)
        !Послать событие не связанное с полем данному процессу
    END
END
CLOSE(Win1)
```

**YIELD (разрешить обработку событий)****YIELD**

Оператор **YIELD** на некоторое время передает управление системе Windows с целью предоставить возможность другим параллельно выполняемым Windows-программам обработать предназначенные для них события (за исключением событий, которые посылают сообщения той же, содержащей оператор YIELD, программе, или событий, которые передают фокус ввода другой программе).

YIELD применяют для исключения ситуации, когда продолжительная пакетная обработка данных в Clarion-программе приводит к “блокировке” исполнения остальных программ. Такая “согласованная многозадачность” (“cooperative multi-tasking”) гарантирует мирное сосуществование Windows-программ разработчика с другими приложениями системы Windows.

В Clarion-программе только оператор YIELD - через таймерные события EVENT:Timer - передает управление другим процессам. Это дает возможность реализовать “фоновую” процедуру в отдельном процессе, используя атрибут TIMER для осуществления продолжительной пакетной обработки данных, что позволяет пользователю продолжить выполнение других задач в программе, не ожидая завершения пакетной обработки.

Приводимый пример иллюстрирует два подхода в осуществлении пакетной обработки

данных: ожидание завершения обработки и фоновое выполнение. Оператор YIELD необходим только в процедуре WaitForProcess, поскольку управление целиком сосредотачивается в ней. Фоновое выполнение, используя событие EVENT:Timer, не нуждается в операторе YIELD, т. к. цикл ACCEPT автоматически реализует согласованную многозадачность с другими Windows-программами.

### Пример:

```

StartProcess PROCEDURE
Win  WINDOW('Choose a Batch Process'),MDI
      BUTTON('Full Control'),USE(?FullControl)
      BUTTON('Background'),USE(?Background)
      BUTTON('Close'),USE(?Close)
      END
CODE
OPEN(Win)
ACCEPT
CASE FIELD()
OF ?FullControl
  DISABLE(FIRSTFIELD(),LASTFIELD())  !Блокировка всех кнопок
  WaitForProcess                      ! и вызов проц. пакетной обработки
  ENABLE(FIRSTFIELD(),LASTFIELD())    !Разрешение кнопок по завершению
                                     !пакетной обработки
OF ?Background
  X# = START(BackgroundProcess)       !Запуск процесса с пакетной обработкой
OF ?Close
  BREAK
END
END
WaitForProcess PROCEDURE                !Полностью управляемая пакетная обработка
CODE
SETCURSOR(CURSOR:Wait)                 !Уведомление: идет пакетная обработка
SET(File)                              !Установка пакетной обработки
LOOP
  NEXT(File)
  IF ERRORCODE() THEN BREAK .
      ! Программный код пакетной обработки
  YIELD                                !Уступить управление другим программам и
                                     !событию EVENT:Timer
END
SETCURSOR                             !Восстановить курсор

BackGroundProcess PROCEDURE             !Фоновая пакетная обработка
Win  WINDOW('Batch Processing...'),TIMER(1),MDI
      BUTTON('Cancel'),STD(STD:Close)
      END
CODE

```



```
OPEN(Win)
SET(File)
ACCEPT
CASE EVENT()
  OF EVENT:CloseWindow
    BREAK
  OF EVENT:Timer
    !Обработка записей по разрешению от
таймера
  LOOP 3 TIMES
  NEXT(File)
  IF ERRORCODE() THEN BREAK .
  ! Программный код пакетной обработки
. . .
```

## ***Прикладные программы с несколькими процессами***

### **Процессы и многозадачность**

---

Многопроцессность, термин, используемый в этой книге, не следует путать со способностью компьютера выполнять несколько задач одновременно. Выполнение нескольких процессов в рамках одной выполняющейся программы не подразумевает многозадачности, поскольку в данный момент, обычно, выполняется только один процесс.

Windows 3.1 реализует согласованную неприоритетную многозадачность для нескольких независимых прикладных программ в любом из режимов работы, но только в 386 расширенном режиме возможна приоритетная многозадачность. Основа приоритетной многозадачности - “квантование времени” между задачами. Величину кванта времени, выделяемого каждой программе, назначает конечный пользователь посредством соответствующей конфигурации Windows. В документации по Windows приведено подробное описание установок связанных с многозадачностью.

В Windows 95 реализуется приоритетная многозадачность относительно отдельно выполняющихся приложений. Эта многозадачность основана не на квантовании времени между приложениями, а является истинной приоритетной многозадачностью, при которой для того, чтобы обеспечить выполнение и других программ, величина интервала времени, который приложение получает для непрерывного выполнения, регулируется посредством возврата управления от каждой программы.

В рамках одного приложения, написанного на языке Clarion, используя атрибут TIMER, можно реализовать разновидность согласованной неприоритетной многопроцессовости (подобную многозадачности на уровне приложений). Такая многопроцессовость тоже основывается не на квантовании времени между процессами. Вместо кванта времени каждый процесс, получает управление и не возвращает его до тех пор, пока не закончится выполнение оператора ASK или ACCEPT.

Если верхнее окно какого-либо исполняющегося процесса имеет атрибут TIMER, то для того, чтобы инициировать выполнение его цикла ACCEPT и обработки события, периодически генерируется таймерное событие (EVENT:Timer). Это событие происходит, и даже если на данный процесс в данный момент времени не переключен фокус ввода. Следовательно, если вы хотите реализовать такой тип многопроцессовости, вы должны убедиться, что в каждую выполняющуюся продолжительное время часть программы вставлены операторы YIELD, которые время от времени выполняясь, позволяли генерировать и обрабатывать таймерные события в других процессах.

### **Процессы и MDI**

---

Программа с несколькими процессами предоставляет пользователю возможность переключаться в процессе работы с одного процесса на другой. Это позволяет реализовать Windows концепцию программирования, известную как интерфейс многих документов (MDI). Одной Windows-программе разрешено одновременно использовать не более 64 процессов.

Первым процессом для любой программы считается код главной программы. Процесс открывает структуру APPLICATION в качестве порождающего MDI-окна с элементами глобального меню программы.

При выборе элемента меню в структуре MENUBAR происходит вызов функции START, которая и начинает очередной исполняемый процесс. Процедуры, вызванные функцией START, обычно открывают порождаемую MDI структуру WINDOW в качестве документального или диалогового окна. Эти окна дают возможность пользователю выполнить стоящие перед ним задачи.

Последняя порожденная MDI структура WINDOW, открытая (и не закрытая) некоторым процессом, представляет собой “верхнее” окно процесса, на которое направляется фокус ввода при передаче управления процессу. Пользователь может переключиться с одного процесса на другой щелчком мыши на верхнем окне нужного процесса. Переключение процессов происходит и при выборе открытого окна из списка MDI-окон в глобальном меню, если меню структуры APPLICATION содержит пункт стандартного Windows меню.

## START (возвращает новый процесс)

**START**(процедура[,стек])

**START**  
процедура

Начинает новый процесс  
Метка первой PROCEDURE вызова нового процесса. Описание процедуры не должно иметь входных параметров.

стек

Целочисленная константа или переменная, указывающая размер стека для размещения нового процесса. Если этот параметр опущен, то значение стека по умолчанию - 10,000 байт.

Функция **START** начинает новый процесс посредством вызова процедуры. Возвращаемое значение - номер, присвоенный новому процессу. Номер процесса используется процедурами и функциями (такими как SETTARGET) при их выполнении в других процессах. Для одной программы допустимо одновременно иметь не более 64 процессов.

Для любой программы первым процессом считается код главной программы; номер такого процесса всегда равен 1. Таким образом, первый раз вызываемая функция START

возвращает наименьший номер - 2. Возвращаемый номер 0 всегда означает ошибку открытия процесса. Такое возможно либо при попытке открыть 65 процесс, либо при недостаточном для процесса объеме памяти, либо при попытке начать новый процесс когда система - модальная.

Тип возвращаемого значения: LONG

### Пример:

```

MainWin      APPLICATION('My Application'),SYSTEM,MAX,ICON(' MyIcon.ICO'),STATUS
              ,HVSCROLL.RESIZE
              MENUBAR
              MENU('&File').USE(?FileMenu)
              ITEM('Selection &1...'),USE(?MenuSelection1)
              ITEM('Selection &2...'),USE(?MenuSelection2)
              END
              END
              END
SaveThread1   LONG           !Переменная для номера процесса
SaveThread2   LONG           !Переменная для номера процесса
CODE
OPEN(MainWin)           !Открыть APPLICATION
ACCEPT          !Обработать глобальные события
CASE ACCEPTED( )
OF ?MenuSelection1
    SaveThread1 = START(NewProc1) !Начать новый процесс
OF ?MenuSelection2
    SaveThread2 = START(NewProc2) !Начать новый процесс
OF ?Exit
    RETURN
. . .

```

## THREAD (выдать номер текущего процесса)

### THREAD()

Функция **THREAD** возвращает номер текущего процесса. Номер процесса используется процедурами и функциями (такими как SETTARGET) при их выполнении в других процессах.

Для одной программы допустимо одновременно иметь не более 64 процессов. Для любой программы первым процессом считается код главной программы; номер такого процесса всегда равен 1. Таким образом, функция THREAD всегда возвращает значение из диапазона от 1 до 64.

Тип возвращаемого значения: LONG

### Пример:

```

MainWin  APPLICATION('My Application'),SYSTEM,MAX,ICON('Mylcon.ICO'),STATUS
          ,HVSCROLL.RESIZE
          MENUBAR
          MENU('&File').USE(?FileMenu)
          ITEM('Selection &1...'),USE(?MenuSelection1)
          ITEM('Selection &2...'),USE(?MenuSelection2)
          END
          END
          END
SaveThread  LONG          !Переменная для номера процесса
SaveThread1 LONG          !Переменная для номера процесса
SaveThread2 LONG          !Переменная для номера процесса
CODE
SaveThread = THREAD()          !Запомнить номер текущего процесса
OPEN(MainWin)          !Открыть APPLICATION
ACCEPT      !Обработать глобальные события
CASE ACCEPTED( )
  OF ?MenuSelection1
    SaveThread1 = START(NewProc1) !Начать новый процесс
  OF ?MenuSelection2
    SaveThread2 = START(NewProc2) !Начать новый процесс
  OF ?Exit
    RETURN
  END
END
END

```

## UNLOCKTHREAD (разблокировать обработку сообщений)

### UNLOCKTHREAD

Оператор **UNLOCKTHREAD** позволяет программе на языке Clarion к сторонней программе или процедуре интерфейса прикладных программ (API), которая содержит свою собственную обработку сообщений (подобную циклу ACCEPT).

Обычно, циклы ACCEPT в программе, написанной на языке Clarion, выполняются по очереди, поэтому не возникает проблемы обновременного доступа к данным. Переключение процессов происходит только при выполнении оператора ACCEPT и, таким образом, в любой момент может выполняться только один процесс. Однако, если текущий исполняемый процесс обратится к внешней процедуре (включая и процедуры API), которая, например, открывает окно и пока окно открыто обрабатывает сообщения, то другим процессам должна быть предоставлена возможность параллельно обрабатывать свои собственные сообщения. Это разрешается выполнением оператора UNLOCKTHREAD перед обращением к такой внешней процедуре. А после возврата от нее управления нужно

выполнить оператор LOCKTHREAD.

Поскольку UNLOCKTHREAD позволяет другим процессам прерывать текущий исполняемый процесс, важно чтобы между операторами UNLOCKTHREAD и LOCKTHREAD не было обращений к библиотечным процедурам исполняемой системы Clarion. Это означает, что нельзя обращаться к процедурам и функциям языка Clarion. Также нельзя выполнять операции с данными типа STRING, CSTRING, PSTRING, DECIMAL и PDECIMAL. Единственным исключением является передача переменной типа STRING, CSTRING, PSTRING в качестве параметра типа ROW во внешнюю (не на языке Clarion) процедуру. Несоблюдение этих ограничений может привести к порче данных другого исполняемого процесса или другим непредсказуемым последствиям.

Оператор UNLOCKTHREAD не влияет на выполнение 16-ти разрядных приложений и в них игнорируется. Он необходим только в 32-х разрядных программах.

С помощью функции THREADLOCKED() можно определить является ли данный процесс “разблокированным” или нет.

**Пример:**

UNLOCKTHREAD	!Разблокировать процесс
MyLibraryCodeWithMessageLoop	!Обратиться к процедуре со своим циклом обр. сообщений
LOCKTHREAD	! заблокировать процесс

**Смотри также:** LOCKTHREAD, THREADLOCKED

## **LOCKTHREAD (вновь заблокировать внешнюю обработку сообщений)**

### **LOCKTHREAD**

Оператор **LOCKTHREAD** вновь блокирует внешнюю обработку сообщений, которая была разрешена оператором UNLOCKTHREAD. Оператор LOCKTHREAD не влияет на выполнение 16-ти разрядных приложений и в них игнорируется. Он необходим только в 32-х разрядных программах.

С помощью функции THREADLOCKED() можно определить является ли данный процесс “разблокированным” или нет.

**Пример:**

UNLOCKTHREAD	!Разблокировать процесс
MyLibraryCodeWithMessageLoop	!Обратиться к процедуре со своим циклом обр. сообщений
LOCKTHREAD	! заблокировать процесс

Смотри также: UNLOCKTHREAD, THREADLOCKED

## THREADLOCKED (текущее состояние внешней обработки сообщений)

### THREADLOCKED()

Функция **THREADLOCKED** возвращает текущее состояние внешней обработки сообщений. Если возвращается 0, то она разблокирована, а если 1 то заблокирована. В 16-ти разрядных программах эта функция всегда возвращает 1.

Тип возвращаемого значения: LONG

#### Пример:

X# = THREADLOCKED()	!возвращает 1
UNLOCKTHREAD	!Разблокировать внешнюю обработку
X# = THREADLOCKED()	! возвращает 0
MyLibraryCodeWithMessageLoop	!Вызвать процедуру со своей обработкой
LOCKTHREAD	! Заблокировать

Смотри также: LOCKTHREAD, UNLOCKTHREAD

## Процедуры работы с окнами

### CHANGE (изменить значение поля)

**CHANGE**(*поле, значение*)

<i>поле</i>	Номер или метка соответствия управляющего поля окна
<i>значение</i>	Константа или переменная, содержащая новое значение для поля

Оператор **CHANGE** изменяет значение, отображаемое в поле структур APPLICATION или WINDOW. Содержимое USE-переменной поля заменяется новым значением, и это новое значение отображается в окне.

#### Пример:

```
Screen    WINDOW,PRE(Scr)
          ENTRY(@N3),USE(Ctl:Code)
          ENTRY(@S30),USE(Ctl:Name)
          BUTTON('OK'),USE(?OkButton),KEY(EnterKey)
          BUTTON('Cancel'),USE(?CanxButton),KEY(EscKey)
          END
CODE
OPEN(Screen)
ACCEPT
CASE SELECTED()
OF ?Ctl:Code
    CHANGE(?Ctl:Code,4)           !Заменить Ctl:Code на 4 и отобразить
OF ?Ctl:Name
    CHANGE(?Ctl:Name,'ABC Company') !Заменить Ctl:Code на ABC Company и отобразить
END
CASE ACCEPTED()
OF ?OkButton
    BREAK
OF ?CanxButton
    CLEAR(Ctl:Record)
    BREAK
END
END
```

### CLOSE (закрыть окно)

**CLOSE**(*метка*)

<b>CLOSE</b>	Закрывает текущую структуру APPLICATION или WINDOW
<i>метка</i>	Метка структуры APPLICATION или WINDOW.



Оператор **CLOSE** завершает обработку текущей APPLICATION или WINDOW структуры. При закрытии окна освобождается отведенная под него память и автоматически восстанавливается область экрана, находившаяся “за” окном. Если закрывается окно, которое не является верхним окном процесса, то сначала закрываются все окна открытые вслед за ним, причем порядок их закрытия обратен порядку открытия.

Если структура APPLICATION или WINDOW объявлена внутри процедуры или функции (локальная), то она автоматически закрывается при возврате из процедуры или функции.

### Пример:

CLOSE(MenuScr)	!Закреть экран меню
CLOSE(CustEntry)	!Закреть экран данных заказчика

## CREATE (создать новое поле)

**CREATE**( *поле, тип* [, *предок*] [, *положение*])

<b>CREATE</b>	Создает новое поле
<i>поле</i>	Номер или метка соответствия создаваемого поля
<i>тип</i>	Целочисленная константа, выражение, символическое имя или переменная, которые определяют тип создаваемого поля.
<i>предок</i>	Номер или метка соответствия поля. Указывает OPTION или MENU, в которых будет располагаться новое поле, или поле, непосредственно предшествующее новому в последовательности выбора клавишей TAB.
<i>положение</i>	Целочисленная константа, выражение или переменная, которая задает положение создаваемого пункта в меню. Если этот параметр опущен, то пункт добавляется в конец.

Оператор **CREATE** динамически создает новое поле в текущей структуре APPLICATION или WINDOW. Будучи созданным, поле остается скрытым; его свойства можно установить используя либо синтаксис доступа времени исполнения, либо операторы SETPOSITION и SETFONT. Поле появляется на экране только при выполнении оператора UNHIDE. Чтобы поместить новое поле в линейку инструментов, нужно приплюсовать символическое имя CREATE:TOOLBAR к имени типа этого поля.

Можно также использовать оператор CREATE для создания полей в отчете. В этом случае нужно сначала выполнить оператор SETTARGET и установить отчет в качестве активной в данный момент цели оператора CREATE, а также нужно задать параметр предок.

Операторы EQUATE для параметра тип расположены в файле EQUATES.CLW. Представленный ниже список - фрагмент этого файла (полный перечень смотрите в файле

EQUATES.CLW):

```

CREATE:sstring  STRING(шаблон), USE(переменная)
CREATE:string   STRING(константа)
CREATE:image    IMAGE()
CREATE:region   REGION()
CREATE:line     LINE
CREATE:box      BOX()
CREATE:ellipse  ELLIPSE
CREATE:entry    ENTRY
CREATE:button   BUTTON
CREATE:prompt   PROMPT()
CREATE:option   OPTION()
CREATE:radio    RADIO
CREATE:check    CHECK
CREATE:group    GROUP()
CREATE:list     LIST()
CREATE:combo    COMBO
CREATE:spin     SPIN()
CREATE:text     TEXT()
CREATE:custom   CUSTOM
CREATE:droplist LIST(),DROP()
CREATE:dropcombo COMBO(),DROP()
CREATE:menu     MENU()
CREATE:item     ITEM()

```

### Пример:

```

Screen  WINDOW,PRE(Scr)
        ENTRY(@N3),USE(Ctl:Code)
        ENTRY(@S30),USE(Ctl:Name)
        BUTTON('OK'),USE(?OkButton),KEY(EnterKey)
        BUTTON('Cancel'),USE(?CanxButton),KEY(EscKey)
        END
X       SHORT
Y       SHORT
Width   SHORT
Heigth  SHORT
Code4Entry STRING(10)
?Code4Entry EQUATE(100)
!Создать для использования в операторе CREATE произвольную метку соответствия
CODE
OPEN(Screen)
ACCEPT
CASE ACCEPTED()

```

```

OF ?Ctl:Code
IF Ctl:Code = 4
    CREATE(?Code4Entry,CREATE:entry)      !Создать поле, которое
    ?Code4Entry{PROP:use} = Code4Entry    !Установить USE-переменную
    ?Code4Entry{PROP:text} = '@s10'      !Установить шаблон ввода
    GETPOSITION(?Ctl:Code,X,Y,Width,Heigh)
    ?Code4Entry{PROP:Xpos } = X + Width + 40 !Задать координату x
    ?Code4Entry{PROP:Ypos } = Y          !Задать координату y
    UNHIDE(?Code4Entry)                  !Отобразить новое поле
END
OF ?OkButton
BREAK
OF ?CanxButton
CLEAR(Ctl:Record)
BREAK
END
END
CLOSE(Screen)
RETURN
    
```

Смотри также: DESTROY, SETTARGET

## DESTROY (удалить экранный объект)

**DESTROY**(*первый объект* [,*последний объект*] )

**DESTROY** Удаляет экранные объекты

*первый объект* Номер поля или мнемоническая метка соответствия удаляемого объекта или первого в диапазоне удаляемых объектов. По умолчанию равно 0.

*последний объект* Номер поля или мнемоническая метка соответствия последнего объекта в диапазоне удаляемых объектов.

Оператор **DESTROY** удаляет экранный объект или диапазон объектов в структуре APPLICATION или WINDOW. Ресурсы удаленного объекта возвращаются операционной системе.

Удаление объектов типа GROUP, OPTION, MENU, TAB, и SHEET удаляет также и все находящиеся внутри них объекты.

### Пример:

```

Screen    WINDOW,PRES(Scr)
          ENTRY(@N3),USE(Ctl:Code)
          ENTRY(@S30),USE(Ctl:Name)
          BUTTON('OK'),USE(?OkButton),KEY(EnterKey)
    
```

```

    BUTTON('Cancel'),USE(?CanxButton),KEY(EscKey)
END
CODE
OPEN(Screen)
DESTROY(?Ctl:Code)           !Удалить объект
DESTROY(?Ctl:Code,?Ctl:Name) !Удалить диапазон объектов
DESTROY(2)                   !Удалить объект номер 2

```

Смотри также: CREATE

## DISABLE (блокирует поле)

**DISABLE**(*первое поле* [, *последнее поле*])

**DISABLE** Блокирует доступ к полям в окне  
*первое поле* Номер или метка соответствия поля или первого поля в группе полей.  
 По умолчанию равно 0.  
*последнее поле* Номер или метка соответствия последнего поля в группе полей

Оператор **DISABLE** запрещает доступ к полю или группе полей структур APPLICATION или WINDOW. Поле с заблокированным доступом изображается на экране с уменьшенной интенсивностью.

### Пример:

```

Screen WINDOW,PRE(Scr)
ENTRY(@N3),USE(Ctl:Code)
ENTRY(@S30),USE(Ctl:Name)
BUTTON('OK'),USE(?OkButton),KEY(EnterKey)
BUTTON('Cancel'),USE(?CanxButton),KEY(EscKey)
END
CODE
OPEN(Screen)
DISABLE(?Ctl:Code)           !Сделать поле недоступным
DISABLE(?Ctl:Code,?Ctl:Name) !Сделать недоступными ряд полей
DISABLE(2)                   !Сделать недоступным второе поле

```

Смотри также: ENABLE, HIDE, UNHIDE

## DISPLAY (отобразить USE-переменные на экран)

**DISPLAY**([*первое поле*][, *последнее поле*])

**DISPLAY** Отображает содержимое USE-переменных в соответствующие им поля в окне

*первое поле*                      Номер или метка соответствия поля или первого поля в группе полей.  
По умолчанию равно 0.  
*последнее поле*                      Номер или метка соответствия последнего поля в группе полей

Оператор **DISPLAY** отображает содержимое USE-переменных в соответствующие им поля текущего окна. DISPLAY без параметров выводит на экран значения USE-переменных всех полей. Если параметр первое поле - единственный, то на экран отображается значение USE-переменной указанного поля. Оба параметра первое поле и последнее поле используются для отображения на экран USE-переменных группы полей.

Текущие значения USE-переменных всех полей автоматически отображаются на экран в каждой итерации АСCEPT-цикла. При этом отпадает необходимость использовать для обновления экрана оператор DISPLAY. Однако, может случиться так, что программа в течение длительного времени занята выполнением некоторых действий. Тогда желательно уведомить пользователя о том, что в данный момент происходит, и делать это приходится без возврата к началу АСCEPT-цикла, явно используя оператор DISPLAY, для отображения обновленного значения некоторой переменной.

### Пример:

DISPLAY	!Отобразить все поля
DISPLAY(2)	!Отобразить второе поле
DISPLAY(3,7)	!Отобразить поля с 3-го по 7-е
DISPLAY(?MenuControl)	!Отобразить поле меню
DISPLAY(?TextBlock,?Ok)	!Отобразить ряд полей

**Смотри также:** Метка соответствия поля, UPDATE, ERASE

## ENABLE (разрешает доступ к полю)

**ENABLE**(*первое поле* [,*последнее поле*])

**ENABLE**                      Делает заблокированное поле вновь доступным  
*первое поле*                      Номер или метка соответствия поля или первого поля в группе полей.  
По умолчанию равно 0.  
*последнее поле*                      Номер или метка соответствия последнего поля в группе полей

Оператор **ENABLE** снимает блокировку поля или группы полей уменьшенной интенсивности, либо заблокированных оператором DISABLE, либо объявленных с атрибутом DISABLE. В результате - поле вновь доступно пользователю для выбора.

### Пример:

CODE

```

OPEN(Screen)
DISABLE(?Control2)                !Поле Control2 - недоступно
IF Ctl:Password = 'Supervisor'
    ENABLE(?Control2)             !Поле вновь доступно
END

```

**Смотри также:** DISABLE, HIDE, UNHIDE

## ERASE (очистить поля и USE-переменные)

**ERASE**( [*первое поле*][*последнее поле*] )

**ERASE** Очищает поля и “обнуляет” их USE-переменные  
*первое поле* Номер или метка соответствия поля или первого поля в группе полей.  
 По умолчанию равно 0.  
*последнее поле* Номер или метка соответствия последнего поля в группе полей

Оператор **ERASE** “стирает” данные в полях в окне и очищает соответствующие им USE-переменные. ERASE без параметров очищает все поля в окне. Если указан только один параметр первое поле, то очищается означенное поле и связанная с ним USE-переменная. Оба параметра первое поле и последнее поле используются для очищения группы полей и их USE-переменных.

**Пример:**

```

ERASE(?)                !Очистить текущее поле
ERASE                    !Очистить все поля
ERASE(3,7)              !Очистить поля с 3-го по 7-е
ERASE(?Name,?Zip)       !Очистить поле с Name по Zip
ERASE(?City,?City+2)    !Очистить City и два последующих поля

```

**Смотри также:** Метка соответствия поля

## GETFONT (получить информацию о шрифте)

**GETFONT**( *поле, начертание, размер, цвет, стиль* )

**GETFONT** Выдает информацию об изображаемом шрифте  
*поле* Номер или метка соответствия поля, о шрифте которого выдается информация. Нулевой (0) параметр поле означает структуру WINDOW  
*начертание* Строковая переменная, в которую возвращается имя шрифта  
*размер* Целочисленная переменная, в которой возвращается размер (в пунктах) шрифта  
*цвет* Целочисленная переменная типа LONG, в трех младших байтах которой возвращаются красная, зеленая и синяя компоненты цвета. Отрицательное

значение параметра *цвет* означает системный цвет  
*стиль* Целочисленная переменная, в которой возвращается толщина и стиль шрифта

Оператор **GETFONT** возвращает информацию об изображаемом в поле шрифте. При нулевом (0) значении параметра поле **GETFONT** возвращает информацию об установленном по умолчанию шрифте окна

**Пример:**

```
TypeFace  STRING(20)
Size      BYTE
Color     LONG
Style     LONG
CODE
OPEN(Screen)
GETFONT(0,TypeFace,Size,Color,Style)  !Получить информацию о шрифте окна
```

**Смотри также:** SETFONT

## GETPOSITION (получить информацию о расположении поля)

**GETPOSITION**(поле, *x*, *y*, ширина, высота)

**GETPOSITION** Возвращает информацию о расположении и размерах окон APPLICATION, WINDOW или поля

*поле* Номер или метка соответствия поля, о котором запрашивается информация. Нулевой (0) параметр поле означает окно

*x* Целочисленная переменная, в которой возвращается горизонтальное положение левого верхнего угла

*y* Целочисленная переменная, в которой возвращается вертикальное положение левого верхнего угла

*ширина* Целочисленная переменная в которой возвращается размер по ширине

*высота* Целочисленная переменная, в которой возвращается размер по высоте

Оператор **GETPOSITION** возвращает координаты и размеры поля или окон APPLICATION и WINDOW. Значения координат и размеров зависят от наличия у поля атрибута SCROLL. Если SCROLL присутствует, то значения берутся относительно виртуального окна. В противном случае, значения берутся относительно верхнего левого угла видимой на данный момент части окна. Следовательно, возвращаемые значения не противоречат тому, что указано атрибутом AT или установлено последним по времени оператором SETPOSITION.

Значения параметров x, y, ширина и высота указаны в условных единицах измерения. За условные единицы принимаются одна четверть усредненной ширины символа и одна восьмая его усредненной высоты. Ясно, что величина условной единицы зависит от установленного для окна шрифта по умолчанию. За основу измерений берется либо шрифт, указанный FONT-атрибутом окна, либо системный шрифт по умолчанию.

**Пример:**

```
Screen      WINDOW,PRE(Scr)
            ENTRY(@N3),USE(Ctl:Code)
            ENTRY(@S30),USE(Ctl:Name)
            BUTTON('OK'),USE(?OkButton),KEY(EnterKey)
            BUTTON('Cancel'),USE(?CanxButton),KEY(EscKey)
            END
X           SHORT
Y           SHORT
Width      SHORT
Heigth     SHORT
CODE
OPEN(Screen)
GETPOSITION(?Ctl:Code,X,Y,Width,Heigth)
```

**Смотри также:** SETPOSITION

**HELP (доступ к окну справки)**

**HELP**(*файл\_справки*[,*идентификатор окна*])

<b>HELP</b>	Открывает файл и формирует на экране окно справки
<i>файл_справки</i>	Строковая константа или метка STRING-переменной, содержащие спецификацию файла справки в DOS. Если не указан путь, то подразумевается текущая директория. Если не указано расширение имени, то полагается “.HLP”. Если опущен параметр файл_справки, то его место обязана занять запятая (,).
<i>идентификатор окна</i>	Строковая константа или метка STRING-переменной, в которых находится ключ доступа к справочной системе. Им может быть как ключевое слово, так и “строка контекста”.

Оператор **HELP** открывает файл с именем файл\_справки и формирует окно с именем идентификатор окна. Если в момент выполнения оператора ASK или АССЕПТ пользователь нажмет клавишу F1 (“Help”-клавиша), то текущее окно справки отобразится на экран.

В случае отсутствия параметра идентификатор окна, файл\_справки подготавливается к работе, но на экран не отображается. Если параметр файл\_справки опущен, то берется



текущий файл справки, а на экране формируется окно с именем идентификатор окна. При отсутствии обоих параметров на экран отображается текущий раздел текущего файла справки.

Параметр идентификатор окна может принимать значение ключевого слова справочной (Help) системы. Ключевое слово присутствует в окне Search (Поиск) системы Help. Если ключевое слово относится только к одному разделу файла справки, то при нажатии клавиши F1 этот раздел отображается на экран в окне справки. Если же ключевое слово относится к нескольким разделам, то для пользователя открывается диалоговое окно поиска.

“Строка контекста”, как значение параметра идентификатор окна, обозначается символом тильды (~), за которым следует уникальный идентификатор, относящийся только к одному разделу справки. Отсутствие тильды воспринимается как указание ключевого слова в качестве значения параметра идентификатор окна. При нажатии клавиши F1 на экран отображается тот раздел файла справки, который определяется “строкой контекста”.

Наличие атрибута HLP у поля или у структур APPLICATION и WINDOW также приводит к формированию на экране окон справочной системы.

### Пример:

HELP('C:\HLPDIR\LEDGER.HLP')	!Открыть файл справки для бух. учета
HELP(, '~CustUpd')	!Активизировать окно справки раздела
	! “обновление данных заказчика”
HELP	!Отобразить окно справки

Смотри также: ASK, ACCEPT, HLP

## HIDE (“спрятать” поле)

**HIDE**(*первое поле*[,*последнее поле*])

<b>HIDE</b>	Прячет поля окна
<i>первое поле</i>	Номер или метка соответствия поля или первого поля в группе полей.
	По умолчанию равно 0.
<i>последнее поле</i>	Номер или метка соответствия последнего поля в группе полей

Оператор **HIDE** прячет поле или группу полей структур APPLICATION и WINDOW. Скрытые окна на экране не появляются.

### Пример:

```
Screen    WINDOW,PRE(Scr)
          ENTRY(@N3),USE(Ct1:Code)
```

```
ENTRY(@S30),USE(Ctl:Name)
BUTTON('OK'),USE(?OkButton),KEY(EnterKey)
BUTTON('Cancel'),USE(?CanxButton),KEY(EscKey)
END
```

```
CODE
OPEN(Screen)
HIDE(?Ctl:Code)                                !Скрыть поле
HIDE(?Ctl:Code,?Ctl:Name)                       !Скрыть группу полей
HIDE(2)                                          !Скрыть второе поле
```

Смотри также: UNHIDE, ENABLE, DISABLE

**OPEN (открыть окно для исполнения)**

OPEN()

**OPEN**                   Открывает окно  
*метка*                   Метка структуры APPLICATION или WINDOW

Опреатор **OPEN** подготавливает структуру APPLICATION или WINDOW к отображению на экран. Отражение осуществляется оператором DISPLAY или ACCEPT. Это позволяет выполнить перед высвечиванием некоторые процедуры по настройке экрана.

**Пример:**

```
OPEN(MenuScr)                                !Открыть экран меню
OPEN(CustEntry)                              !Открыть экран данных заказчика
```

**SELECT (выбор поля для последующей обработки)**

SELECT(поле[,позиция           ][,позиция\_конец])

**SELECT**                   Назначает поле, в которое направляется фокус ввода  
*поле*                   Номер или метка соответствия поля, выбираемого для последующей обработки. Если параметр опущен, то оператор SELECT устанавливает режим “принять все” (AcceptAll).  
  
*позиция*                   Указывает местоположение курсора внутри поля. Для полей ENTRY, TEXT, SPIN, COMBO указывается либо позиция символа, либо позиция первого символа блока выделения. Для структуры OPTION - это номер альтернативы выбора. Для управляющего поля LIST - номер QUEUE-элемента. Этот параметр также может задаваться с помощью синтаксиса свойств, как PROP:Selected or PROP:SelStart.  
  
*позиция\_конец*           Указывает позицию последнего символа внутри полей ENTRY, TEXT,

SPIN, COMBO. Позиции символов, указанные параметрами позиция и позиция\_конец, помечаются как начало и конец блока, предназначенного для операций “вырезания” (cut) и “склеивания” (paste). Этот параметр также может задаваться с помощью синтаксиса свойств, как PROP:SelEnd

Оператор **SELECT** нарушает нормальную последовательность выбора полей, осуществляемую TAB-клавишей в структурах APPLICATION или WINDOW. Он оказывает воздействие на последующее выполнение оператора АССЕПТ. Параметр поле определяет то поле, которое будет обработано следующей итерацией АССЕПТ-цикла. Если указанное поле заблокировано операторами DISABLE или HIDE, т. е. не может принять фокус ввода, то фокус устанавливается на следующее незаблокированное поле окна. Если параметр поле определяет объект в структуре TAB, которая в данный момент не имеет фокуса ввода, то перед тем как на объект будет переключен фокус ввода, лист будет перенесен на передний план.

SELECT с параметрами позиция и позиция\_конец определяет в поле блок выделения, предназначенный для операций “вырезания” (cut) и “склеивания” (paste).

SELECT без параметров устанавливает режим “принять все” (AcceptAll), еще называемый нон-стоп режимом. Это режим редактирования поля, в котором каждое управляющее поле окна обрабатывается в последовательности их выбора клавишей TAB, когда для каждого поля порождается событие EVENT:Accepted. Это дает возможность выполнить процедуру проверки правильности ввода данных для всех полей, включая и те, которые не были затронуты пользователем.

Режим “принять все” отменяется, когда возникает одно из следующих условий:

- \* Оператор SELECT(?) выбирает для редактирования то же самое поле. Как правило, такое случается тогда, когда введено недопустимое значение, и пользователь должен повторить ввод данных.

- \* Установлено в 0 свойство Window{PROP:AcceptAll}. Значение свойства равно 1 когда установлен режим “принять все”. Таким образом, присваивая значение атрибуту, можно устанавливать или отменять режим “принять все”.

- \* Нулевое или “пустое” значение поля, у которого присутствует атрибут REQ. Режим “принять все” отменяется, поле подсвечивается для ввода данных, отменяется обработка полей в последовательности выбора клавишей TAB.

Когда обработаны все поля, для данного окна генерируется событие EVENT:Completed.

### Пример:

```
Screen    WINDOW,PRE(Scr)
          ENTRY(@N3),USE(Ctl:Code)
          ENTRY(@S30),USE(Ctl:Name)
```

```

LIST,USE(Ctl:Type),From(TypeQue),Drop(5)
BUTTON('OK'),USE(?OkButton),KEY(EnterKey)
BUTTON('Cancel'),USE(?CanxButton),KEY(EscKey)
END

```

```

CODE
OPEN(Screen)
SELECT(?Ctl:Code)                !Начать с поля Ctl:Code
ACCEPT
CASE SELECTED()
OF ?Ctl:Type
    GET(TypeQue,Ctl:Type)        !Найти тип в списке
    SELECT(?Ctl:Type,POINTER(TypeQue))
END
CASE ACCEPTED()
OF ?Ctl:Code
    IF Ctl:Code > 150             !Если введенные данные неверны
        BEEP                    ! оповестить пользователя и
        SELECT(?)               ! и дать ему возможность повторить ввод данных
    END
OF ?Ctl:Name
    SELECT(?Ctl:Name,1,5)        !Пометить первые пять символов в блоке
OF ?OkButton
    SELECT                      !Инициировать безостановочный режим
END
IF EVENT() = EVENT:Completed THEN BREAK.
    !Закончить безостановочный режим
END

```

**Смотри также:** ACCEPT

## SET3DLOOK (установить объемное изображение окна)

**SET3DLOOK**([переключатель])

**SET3DLOOK**            Устанавливает/отменяет трехмерное восприятие изображения  
*переключатель*        Целочисленная константа, отменяющая (=0) или устанавливающая  
                          (=1) объемное (3D) изображение.

Процедура SET3DLOOK настраивает программу на трехмерное восприятие изображения. По умолчанию программа настраивается на 3D-изображение. Атрибут GRAY структуры WINDOW устанавливает объемное изображение полей. Поля линейки инструментов всегда изображаются объемными, если не было запрета со стороны SET3DLOOK. Атрибут GRAY перестает действовать когда процедурой STE3DLOOK наложен запрет на трехмерное изображение.

SET3DLOOK(0) отменяет изображение в объеме, тогда как SET3DLOOK(1) - устанавливает. Значения отличные от 0 и 1 зарезервированные для будущих применений.

### Пример:

```
MainWin  APPLICATION('My Applicatton'),SYSTEM,MAX,ICON('Mylcon.ICO'),STATUS |
          .HVSCROLL,RESIZE
          MENUBAR
          MENU('&File'),USE(?FileMenu)
          ITEM('&Open...'),USE(?OpenFile)
          ITEM('&Close'),USE(?CloseFile),DISABLE
          ITEM('Turn off 3D Look'),USE(?Toggle3D),CHECK
          ITEM('E&xit'),USE(?MainExit)
          END
          END
          END
CODE
OPEN(MainWin)
ACCEPT
CASE ACCEPTED()
  OF ?Toggle3D      1
    IF MainWin$?Toggle3D{PROP:text} = 'Turn off 3D Look'  !Если 3D разрешено
      SET3DLOOK(0)                                         ! то запретить
      MainWin$?Toggle3D{PROP:text} = 'Turn on 3D Look'    ! и изменить текст
    ELSE !Иначе
      SET3DLOOK(1)                                         ! разрешить 3D
      MainW1n$?Toggle3D(PROP:text) = 'Turn off 3D Look'   ! и изменить текст
    END
  OF ?OpenFi1e
    START(OpenFileProc)
  OF ?MainExit
    BREAK
  END
END
CLOSE(MainWin)
```

## SETCURSOR (временно изменить курсор мыши)

**SETCURSOR**([*курсor*])

**SETCURSOR** Определяет на время форму изображения курсора мыши.  
*курсor* Символическое имя для курсора мыши стандарта Windows

Оператор **SETCURSOR** устанавливает новую форму курсора мыши на время, пока не начнется следующая итерации АСCEPT-цикла. Действие параметров атрибута CURSOR временно приостанавливается. Новая итерация АСCEPT-цикла завершает действие



**SETFONT (установить шрифт)****SETFONT**(поле, начертание, размер, цвет, стиль)

<b>SETFONT</b>	Динамически устанавливает экранный шрифт поля
<i>поле</i>	Номер или метка соответствия поля, для которого устанавливается шрифт. Нулевое значение параметра поле соответствует структуре WINDOW.
<i>начертание</i>	Строковая константа или переменная, в которой указано имя шрифта. Если параметр опущен, то используется системный шрифт.
<i>размер</i>	Целочисленная константа или переменная, указывающая размер шрифта (в пунктах). Если параметр не указан, то принимается размер системного шрифта.
<i>цвет</i>	Целочисленная константа или переменная типа LONG, в трех младших байтах которой указываются значения красной, зеленой и синей компонент цвета шрифта, либо символическое имя значения цвета стандарта Windows. Если параметр не указан, то устанавливается черный цвет.
<i>стиль</i>	Целочисленная константа, константное выражение, символическое имя или переменная, определяющие толщину и стиль шрифта. Если параметр опущен, то принимается нормальная толщина шрифта.

Оператор **SETFONT** динамически устанавливает шрифт поля, замещая действие ранее указанного атрибута FONT. Если параметр поле равно 0, то SETFONT задает шрифт для всех управляющих полей в окне.

В отличие от свойств времени исполнения, которые устанавливаются последовательно один за другим, оператором SETFONT можно изменять все параметры шрифта одновременно. А это дает выигрыш, поскольку все изменения реализуются одновременно, в то время как установка свойства приводит к отдельному изменению и к отдельному отображению этого изменения на экран.

Параметром начертание можно задать имя любого шрифта зарегистрированного в Windows. В файле EQUATES.CLW находятся символические имена значений для стандартных стилей. Значение параметра стиль в диапазоне от 0 до 1000 определяет толщину шрифта. К этому значению можно прибавлять значения для наклона, подчеркивания и перечеркивания текста. Указанные символические имена находятся в файле EQUATES.CLW:

FONT:thin	EQUATE(100)
FONT:regular	EQUATE(400)
FONT:bold	EQUATE(700)
FONT:italic	EQUATE(01000H)
FONT:underline	EQUATE(02000H)
FONT:strikeout	EQUATE(04000H)

**Пример:**

```
SETFONT(1,'Arial',14,,FONT:thin+FONT:Italic)    !Размер - 14 пункт., Arial, черный
                                                ! тонкий, italic
```

**Смотри также:** GETFONT

**SETPOSITION (определить новое положение поля)**

**SETPOSITION**(*поле, x, y, ширина, высота*)

**SETPOSITION** Динамически устанавливает положение и размер поля или окон APPLICATION, WINDOW.

*поле* номер или метка соответствия поля, которое подвергается воздействию. Если параметр поле - нулевой, то воздействию подвергается окно.

*x* Целочисленная константа, выражение или переменная для указания горизонтального положения левого верхнего угла. Если параметр опущен, то x-координата не изменяется.

*y* Целочисленная константа, выражение или переменная для указания вертикального положения левого верхнего угла. Если параметр опущен, то y-координата не изменяется.

*ширина* Целочисленная константа, выражение или переменная для указания ширины. Если параметр опущен, то ширина не изменяется.

*высота* Целочисленная константа, выражение или переменная для указания высоты. Если параметр опущен, то высота не изменяется.

Оператор SETPOSITION динамически устанавливает положение и размер управляющего поля или окон APPLICATION, WINDOW. Если какой-либо из параметров не указан, то соответствующее значение не изменяется.

Значения параметров x, y, ширина и высота указаны в условных единицах измерения. За условные единицы принимаются одна четверть усредненной ширины символа и одна восьмая его усредненной высоты. Ясно, что величина условной единицы зависит от установленного для окна шрифта по умолчанию. За основу измерений берется либо шрифт, указанный FONT-атрибутом окна, либо системный шрифт по умолчанию.

Использование оператора SETPOSITION приводит к более “плавному” изменению внешнего вида поля по сравнению с использованием выражения для свойства при изменении значения параметра атрибута AT. Причина в том, что SETPOSITION изменяет сразу все параметры. Имена свойств должны изменять каждый параметр индивидуально. Поскольку индивидуальное изменение параметра тотчас же ощущается зрительно, то это



приводит к скачкообразным изменениям поля.

### Пример:

CREATE(?Code4Entry,CREATE:entry,?Ctl:Code)	!Создать поле
?Code4Entry{PROP:use} = 'Code4Entry'	!Установить USE-переменную
?Code4Entry{PROP:text} = '@s10'	!Установить шаблон ввода
GETPOSITION(?Ctl:Code,X,Y,Width,Height)	!Считать положение Ctl:Code
SETPOSITION(?Code4Entry,X+Width+40,Y)	!Сдвинуть вправо на 40
UNHIDE(?Code4Entry)	!Вывести новое поле на экран

Смотри также: GETPOSITION

## SETTARGET (сделать окно или отчет текущим)

**SETTARGET([объект][,процесс])**

<b>SETTARGET</b>	Назначает окно (или отчет) текущим для отображения графических образов и для операторов взаимодействия с окнами.
<i>объект</i>	Метка структур APPLICATION, WINDOW или REPORT. Если параметр опущен, то используется последнее открытое и еще не закрытое окно указанного процесса.
<i>процесс</i>	Номер исполняемого процесса, в котором находится процедура или функция с последним для этого процесса открытым и еще не закрытым окном, указанным параметром объект. Если параметр опущен, то используется текущий процесс.

Процедура SETTARGET устанавливает структуру объект в качестве текущей для вывода графики. С ее помощью назначается объект, на который воздействуют установки свойств в процессе исполнения программы, а также объект воздействия операторов CREATE, SETPOSITION, GETPOSITION, SETFONT, GETFONT, DISABLE, HIDE, CONTENTS, DISPLAY, ERASE, UPDATE. Совместное использование этих операторов с процедурой SETTARGET позволяет вносить изменения в изображение верхнего окна любого процесса.

Любые графические образы, которые создаются при помощи графических процедур и функций (см. главу Управление Графикой), будут изображаться на указанном объекте. Это позволяет использовать графику в верхнем окне или отчете любого процесса.

Процедура SETTARGET устанавливает значение встроенной переменной TARGET (автоматически устанавливается при открытии окна), которая может быть использована операторами программы когда нужно указать метку текущего окна или отчета. Поскольку структура данных REPORT не может быть объектом по умолчанию, то SETTARGET должна быть выполнена до применения операторов вывода графики в окно REPORT.

Исполнение SETTARGET не приводит к запуску другого процесса, не происходит также подмены обработчика Windows-событий - АСCEPT-цикла. Процедура SETTARGET без параметров устанавливает в качестве объекта окно процесса, АСCEPT-цикл которого выполняется в данный момент.

### Пример:

```
Report REPORT
    !Управляющие поля документа
END
CODE
OPEN(Report)
SETTARGET(Report)
TARGET{PROP:Landscape} = 1      !Документ - текущий объект
                                ! установить режим печати landscape
```

## UNHIDE (изобразить спрятанное поле)

**UNHIDE**(*первое поле* [, *последнее поле*])

**UNHIDE**                      Отображает ранее спрятанное поле  
*первое поле*                      номер или метка соответствия поля или первого поля в группе полей.  
                                     Если параметр опущен, то по умолчанию равен 0.  
*последнее поле*                      номер или метка соответствия последнего поля в группе полей

Оператор **UNHIDE** вновь активизирует поле или группу полей, спрятанных оператором HIDE. Восстановленные поля вновь появляются на экране.

### Пример:

```
CODE
OPEN(Screen)
HIDE(?Control2)                  !Скрыть поле Control2
IF Ctl:Password = 'Supervisor'
    UNHIDE(?Control2)           !Восстановить Control2
END
```

**Смотри также:** HIDE, ENABLE, DISABLE

## UPDATE (запись с экрана в USE-переменную)

**UPDATE**(*первое поле* [, *последнее поле*])

**UPDATE**                      Заносит содержимое поля в его USE-переменную  
*первое поле*                      номер или метка соответствия поля или первого поля в группе полей.

*последнее поле*            номер или метка соответствия последнего поля в группе полей

Оператор UPDATE записывает изображенное на экране значение поля в его USE-переменную. Значение, изображенное на экране, помещается в переменную, которую определяет USE-атрибут поля.

Оператор ACCEPT автоматически обновляет USE-переменные когда каждое поле становится объектом его воздействия. Однако, не все события (например, нажатие одной из ALERT-клавиш) автоматически обновляют USE-переменные. Приходится прибегать к помощи оператора UPDATE.

### **UPDATE Обновляет все поля экрана**

**UPDATE**(*первое поле*)

Обновляет USE-переменную указанного поля значением с экрана

**UPDATE**(*первое поле, последние поле*)

Обновляет USE-переменные группы полей значениями с экрана

#### **Пример:**

UPDATE(?)	!Обновить текущее поле
UPDATE	!Обновить все поля
UPDATE(?Adress)	!Обновить поле ?Adress
UPDATE(3,7)	!Обновить поля с 3-го по 7-е
UPDATE(?Name,?Zip)	!Обновить поля с Name по Zip
UPDATE(?City,?City+2)	!Обновить City и два последующих поля

**Смотри также:** Метка соответствия поля

## Функции работы с Окнами

### ACCEPTED (указать выполненное поле)

#### ACCEPTED()

Функция **ACCEPTED** возвращает номер поля, для которого имело место событие EVENT:Accepted. Для всех других событий функция возвращает 0.

Компилятор присваивает номерам полей структуры WINDOW положительные значения в порядке возрастания - как поля следуют при объявлении структуры WINDOW. Номерам полей структуры APPLICATION присваиваются отрицательные значения. В операторах программы номера полей представлены, как правило, метками соответствия - именем USE-переменной, перед которой стоит знак вопроса (?FieldName).

Тип возвращаемых данных: SHORT

#### Пример:

```
CASE ACCEPTED()                                !Завершающие действия
OF ?Cus:Company                                  !Редактирование значения поля
!Редактирование значения поля
OF ?Cus:CustType                                !Редактирование значения поля
!Редактирование значения поля
END
```

Смотри также: ACCEPT, EVENT

### CHOICE (указать относительное положение элемента)

#### CHOICE([поле])

**CHOICE** Возвращает номер осуществленного пользователем выбора.  
*поле* метка соответствия полей LIST, COMBO или OPTION.

Функция **CHOICE** возвращает порядковый номер выбранного элемента (выбора) структуры OPTION, окна списка LIST или поля COMBO. Если параметр отсутствует, то функция возвращает порядковый номер выбранного элемента того управляющего поля (LIST, OPTION или COMBO), которое породило последнее по времени связанное с полем событие, обработанное ACCEPT-циклом. CHOICE(поле) возвращает номер текущего выбора в любом из полей LIST, OPTION или COMBO активного на данный момент окна.

Для структуры OPTION функция CHOICE указывает порядковый номер выбранной RADIO-кнопки. Порядковый номер определяется относительным положением в OPTION.

Для окон LIST и COMBO функция CHOICE возвращает “номер очереди”, которую “занимает” выбранный элемент в структуре QUEUE.

### Пример:

CODE	
ACCEPT	
EXECUTE CHOICE()	!Отработка меню:
AddRec	! добавить запись
PutRec	! изменить запись
DelRec	! удалить запись
RETURN	! выход
END	
END	

## CONTENTS (вернуть значение USE-переменной)

## CONTENTS(*поле*)

<b>CONTENTS</b>	Возвращает строку со значением USE-переменной поля.
<i>поле</i>	номер или метка соответствия поля.

Функция **CONTENTS** возвращает строку со значением USE-переменной полей ENTRY, OPTION, RADIO или TEXT.

USE-переменная может быть как длиннее, так и короче, чем указано в шаблоне соответствующего ей поля. В любом случае функция CONTENTS возвращает строку с действительным значением USE-переменной.

Тип возвращаемых данных: STRING

### Пример:

```
IF CONTENTS(?LastName) = ' ' AND CONTENTS(?FirstName) = ' '
    !Если поля пустые, то
    MessageFiel = 'Must Enter a First or Last Name'
    ! вывести сообщение об ошибке
END
```

**FIELD (указать поле, владеющее фокусом)****FIELD()**

Функция **FIELD** возвращает номер поля, владеющего фокусом в момент возникновения событий связанных с полем. В их число попадают события EVENT:Selected и EVENT:Accepted. Функция возвращает 0 для событий не связанных с полем.

Компилятор присваивает номерам полей структуры WINDOW положительные значения в порядке возрастания - как поля следуют при объявлении структуры WINDOW. Номерам полей структуры APPLICATION присваиваются отрицательные значения. В операторах программы номера полей представлены, как правило, метками соответствия - именем USE-переменной, перед которой стоит знак вопроса (?FieldName).

Тип возвращаемых данных: LONG

**Пример:**

```
Screen    WINDOW
          ENTRY(@N4),USE(Control1)
          ENTRY(@N4),USE(Control2)
          ENTRY(@N4),USE(Control3)
          ENTRY(@N4),USE(Control4)
        END
CODE
ACCEPT
  IF NOT ACCEPTED() THEN CYCLE .
  CASE FIELD()
  OF ?Control1
    IF Control1 = 0
      BEEP
      SELECT(?)
    END
  OF ?Control2
    IF Control2 > 4
      Scr:Message = 'Control must be less than 4'
      ERASE(?)
      SELECT(?)
    ELSE
      CLEAR(Scr:Message)
    END
  OF ?Control4
    BREAK
  END
END
END
```

!Редактирование полей  
! первое поле  
! если не было ввода  
! сигнал предупреждения  
! остаться в этом же поле  
  
! второе поле  
! если статус больше 4  
  
! очистить поле  
! и вновь отредактировать  
! значение - допустимо  
! убрать сообщение  
  
! четвертое поле  
! завершить цикл обработки  
! конец CASE  
! конец ACCEPT

**Смотри также:** ACCEPTED, SELECTED, FOCUS, EVENT

## FIRSTFIELD (указать первое поле окна)

### FIRSTFIELD()

Функция FIRSTFIELD возвращает наименьший номер поля активного в данный момент окна или окно или отчет, указанный оператором SETTARGET.

Тип возвращаемых данных: LONG

#### Пример:

DISABLE(FIRSTFIELD(), LASTFIELD()) ! Заблокировать все поля

## FOCUS (указать поле, владеющее фокусом)

### FOCUS()

Функция FOCUS возвращает номер поля владеющего фокусом ввода вне зависимости от времени возникновения некоторого события.

Компилятор присваивает номерам полей структуры WINDOW положительные значения в порядке возрастания - как поля следуют при объявлении структуры WINDOW. Номерам полей структуры APPLICATION присваиваются отрицательные значения. В операторах программы номера полей представлены, как правило, метками соответствия - именем USE-переменной, перед которой стоит знак вопроса (?FieldName).

Тип возвращаемых данных: LONG

#### Пример:

```
Screen      WINDOW
            ENTRY(@N4),USE(Control1)
            ENTRY(@N4),USE(Control2)
            ENTRY(@N4),USE(Control3)
            ENTRY(@N4),USE(Control4)
            END
```

CODE

ACCEPT

CASE EVENT()

OF EVENT:LoseFocus

OROF EVENT:CloseWindow

CASE FOCUS()

!Редактирование полей





## LASTFIELD (указать последнее поле окна)

### LASFIELD()

Функция **LASTFIELD** возвращает наибольший возможный для текущего окна номер поля или окно или отчет, указанный оператором SETTARGET.

Тип возвращаемых данных: LONG

**Пример:**

DISABLE(FIRSTFIELD(), LASTFIELD()) ! Заблокировать все поля

## MESSAGE (изобразить окно сообщений)

### MESSAGE(текст[,заголовок][,пиктограмма][,кнопки][,умолчание][,стиль])

<b>MESSAGE</b>	Выводит на экран окно сообщений и возвращает номер нажатой пользователем кнопки.
<i>текст</i>	Строковая константа или переменная, в коих находится текст, предназначенный для окна сообщений. Вертикальная черта ( ) означает переход на новую строку в сообщениях из нескольких строк.
<i>заголовок</i>	Заголовок окна сообщений. Если параметр опущен, то окно изображается без заголовка.
<i>пиктограмма</i>	Строковая константа, переменная или символическое имя для указания пиктограммы стандарта Windows. Если параметр не указан, то окно изображается без пиктограммы.
<i>кнопки</i>	Целочисленная константа, переменная, символическое имя или выражение для указания тех кнопок стандарта Windows, которые будут размещены в окне сообщений. Одним значением параметра можно указать сразу несколько кнопок. Если параметр не указан, то в окне разместится кнопка "Ok".
<i>умолчание</i>	Целочисленная константа, переменная, символическое имя или выражение для назначения в окне сообщений кнопки по умолчанию. Если параметр не указан, то первая кнопка - кнопка по умолчанию.
<i>стиль</i>	Целочисленная константа или переменная, которые указывают, что окно является модальным для программы (0) или модальным для системы Windows (1). Если параметр опущен, то окно - модальное для программы.

Функция **MESSAGE** выводит на экран окно сообщений стандарта Windows и ждет ответа пользователя. Как правило, это ответ типа "Да" или "Нет", либо типа "Ok".

Следующие символические имена для параметров кнопки, и умолчание находятся в файле EQUATES.CLW :

BUTTON:OK  
BUTTON:YES  
BUTTON:NO  
BUTTON:ABORT  
BUTTON:RETRY  
BUTTON:IGNORE  
BUTTON:CANCEL  
BUTTON:HELP

Функция MESSAGE возвращает номер кнопки нажатой пользователем для выхода из диалогового окна. Номер кнопки это одна из констант, которые представляются этими мнемоническими именами.

Следующие символические имена наиболее часто используются для параметра пиктограмма (в файле EQUATES.CLW их значительно больше):

ICON:None  
ICON:Application  
ICON:Hand  
ICON:Question  
ICON:Exclamation  
ICON:Asterisk  
ICON:Pick  
ICON:Clarion

Параметр стиль устанавливает, что окно сообщений является модальным для программы или модальным для системы Windows. Модальное для программы окно должно быть закрыто прежде, чем разрешить пользователю что-либо делать далее в прикладной программе, но оно не лишает пользователя возможности переключаться на другие Windows-программы. Окно модальное для Windows должно быть закрыто прежде, чем разрешить пользователю что-либо делать далее в Windows.

Тип возвращаемых данных: USHORT

### Пример:

```
CASE MESSAGE('Quit?', 'Editor', ICON:Question,BUTTON:Yes+BUTTON:No,  
BUTTON:No,1)  
    !Пиктограмма для "?" с кнопками "Да" и "Нет"  
    ! с кнопкой по умолчанию "Нет", окно модальное  
    ! для Windows  
    OF BUTTON:No  
    OF BUTTON:Yes  
        MESSAGE('Goodbye|So Long|Sayonara')    !Окно сообщений из 3-х строк только с  
Ок кнопкой
```

```
RETURN  
END
```

## MOUSEX (получить положение мыши по горизонтали)

### MOUSEX()

Функция **MOUSEX** возвращает число, которое определяет горизонтальное положение курсора мыши в момент возникновения события. Положение берется относительно начала окна.

Тип возвращаемых данных: USHORT

**Пример:**

```
SaveMouseX = MOUSEX()           !Запомнить положение мыши
```

## MOUSEY (получить положение мыши по вертикали)

### MOUSEY()

Функция **MOUSEY** возвращает число, которое определяет вертикальное положение курсора мыши в момент возникновения события. Положение берется относительно начала окна.

Тип возвращаемых данных: USHORT

**Пример:**

```
SaveMouseY = MOUSEY()           !Запомнить положение мыши
```

## POPUP(получить выбор пользователя в спускающемся меню)

### POPUP( пункты [ , x ] [ , y ] )

**POPUP** Возвращает целое число, означающее сделанный пользователем выбор из меню.

*пункты* Строковая константа, переменная или выражение, содержащее текст пунктов меню.

*x* Целочисленная константа, переменная или выражение, которое указывает положение левого верхнего угла по горизонтали. Если этот параметр опущен, то меню раскрывается в текущей позиции курсора.

*y* Целочисленная константа, переменная или выражение, которое указывает положение левого верхнего угла по вертикали. Если этот

параметр опущен, то меню раскрывается в текущей позиции курсора.

Функция **POPUP** возвращает целое число, означающее сделанный пользователем выбор из меню, которое раскрывается при обращении к данной функции. Если пользователь щелкает кнопкой мыши вне пределов меню или нажимает клавишу ESC, означающая отсутствие выбора, то функция POPUP возвращает 0.

В строке пункты, каждый пункт меню должен отделяться от другого символом вертикальной черты (|). Дефис между вертикальными чертами (|-) определяет разделитель между группами пунктов. Пункт, следующий непосредственно за символом тильда (~) является деактивизированным (в меню он выглядит затухеванным). Пункт, перед которым стоит символ плюс (+), на экране выводится с пометкой слева от него. Пункт, перед которым стоит символ минус (-), на экране выводится без пометки. Пункт, за которым следует набор пунктов, заключенный в фигурные скобки (|SubMenu{{SubChoice 1|SubChoice 2}}|), определяет вложенное меню (две фигурные скобки нужны для того, чтобы компилятор отличил этот случай от случая повторения символов в строке).

Каждый пункт меню нумеруется в возрастающей последовательности, начиная с единицы (1), в соответствии с его положением в строке пункты. Разделители и вложенные меню в последовательность номеров не включаются (что увеличивает эффективность использования с этой функцией операторной структуры EXECUTE). Когда пользователь щелкает мышью или нажимает клавишу ENTER на определенном пункте меню, выполнение функции завершается и она возвращает номер выбранного пункта.

Тип возвращаемого значения:     SHORT

### Пример:

```

PopupString = 'First|+Second| Sub menu{{One|Two}}|-|Third|~Disabled'
ToggleChecked = 1
ACCEPT
CASE EVENT()
OF EVENT:AlertKey
  IF KEYCODE() = MouseRight
    EXECUTE POPUP(PopupString)
    FirstProc           !Вызвать процедуру для пункта 1
    BEGIN               !Операторы для переключения значения пункта 2
      IF ToggleChecked = 1 !Проверить состояние переключателя
        SecondProc(Off)   !Процедура выключения чего-то
        PopupString[7] = '-' !Изменить строку меню, чтобы не было отметки
        ToggleChecked = 0 !Установить переключатель
      ELSE
        SecondProc(On)    !Процедура включения чего-то
        PopupString = 'First|+Second| Sub menu{{One|Two}}|-|Third|~Disabled'
                          !Изменить строку меню, чтобы отметка была

```

```

    ToggleChecked = 1      !Установить переключатель
  END
END                        !Конец операторов для переключения пункта 2
OneProc                   !Вызвать процедуру для пункта 3
TwoProc                   !Вызвать процедуру для пункта 4
ThirdProc                 !Вызвать процедуру для пункта 5
DisabledProc             !Пункт 6 неактивен, так что нельзя выполнить эту процедуру
END
END
END
END

```

## SELECTED (указать поле, владеющее фокусом ввода)

### SELECTED()

Для события EVENT:Selected функция SELECTED возвращает номер управляющего поля, завладевшего фокусом ввода. Функция возвращает 0 для всех остальных событий.

Компилятор присваивает номерам полей структуры WINDOW положительные значения в порядке возрастания - как поля следуют при объявлении структуры WINDOW. Номерам полей структуры APPLICATION присваиваются отрицательные значения. В операторах программы номера полей представлены, как правило, метками соответствия - именем USE-переменной, перед которой стоит знак вопроса (?FieldName).

Тип возвращаемых данных: SHORT

### Пример:

```

CASE SELECTED()           !Предварительные действия
OF ?Cus:Company
                           !Переустановить значение поля
OF ?Cus:CustType
                           !Переустановить значение поля
END

```

**Смотри также:** ACCEPT, SELECT

## Процедуры работы с клавиатурой

### ALIAS (переопределить код клавиши)

ALIAS([код клавиши],[новый код])

<b>ALIAS</b>	Заменяет код, сгенерированный при нажатии клавиши.
<i>код клавиши</i>	Целочисленная константа или символическое имя кода клавиши. Если опущены оба параметра, то все переопределенные ранее клавиши вернутся к своим первоначальным значениям.
<i>новый код</i>	Целочисленная константа или символическое имя кода клавиши. Если параметр опущен, то код клавиши возвращается к своему истинному значению.

**ALIAS** заменяет сгенерированный при нажатии клавиши код клавиши на новый код. **ALIAS** не воздействует на “нажатие” клавиши, осуществляемое посредством **PRESSKEY**. Процедура **ALIAS** действует глобально - для всех процессов - независимо от того, где она выполняется. Поэтому, если нужно ограничить действие замены кода клавиши, например, текущим окном, то необходимо вернуть все **ALIAS**-клавиши в исходное состояние когда фокус ввода переходит к другому окну.

В качестве значения нового кода может быть использовано значение из диапазона от 0800h до 0FFFFh.

#### Пример:

ALIAS(EnterKey,TabKey)	!Вместо Tab можно нажимать Enter
ALIAS(F3Key,F1Key)	!Теперь Help будет на F3
ALIAS()	!Отменит все переназначения

### ASK (ждать нажатие клавиши)

ASK

**ASK** считывает из буфера клавиатуры код нажатой клавиши. Исполнение программы приостанавливается в ожидании нажатия клавиши. Если код нажатия уже помещен в буфер клавиатуры, то **ASK** считывает его без ожидания.

Оператор **ASK** дает возможность произойти событиям, обусловленным атрибутом **TIMER**, при этом для обработки каждого события активизируется “свой” **АССЕРТ**-цикл. Это обстоятельство позволяет другим процессам “обслуживать” **TIMER**-события во время выполнения данным процессом процедуры пакетной обработки данных.

#### Пример:

ASK	!Ждать нажатия клавиши
LOOP WHILE KEYBOARD()	!Очистить буфер клавиатуры
ASK	! без обработки нажатых клавиш
END	

## **PRESS (поместить в буфер строку символов)**

**PRESS**(*строка*)

**PRESS** Помещает символы во входной буфер клавиатуры.  
*строка* Строковая константа, переменная или выражение

Оператор **PRESS** помещает символы во входной буфер клавиатуры системы Windows. В буфер заносятся все символы строки. Помещенная в буфер строка обрабатывается так же, как если бы символы вводились с клавиатуры.

### **Пример:**

IF Action = 'AddRecord'	!При добавлении записи в мемо
TempString = FORMAT(TODAY(),@D1) & ' ' & FORMAT(CLOCK(),@T4)	
PRESS(TempString)	!Установить дату и время в 1 строку мемо
END	

## **PRESSKEY (поместить в буфер код нажатия клавиши)**

**PRESSKEY**(*код клавиши*)

**PRESSKEY** Помещает код нажатия во входной буфер клавиатуры.  
*код клавиши* Целочисленная константа или символическое имя кода клавиши.

Оператор **PRESSKEY** помещает код нажатия клавиши во входной буфер клавиатуры системы Windows. В буфер заносятся все символы строки. Помещенный в буфер код клавиши обрабатывается так же, как если бы пользователь нажал клавишу клавиатуры. Оператор **ALIAS** не изменяет кода клавиши, “нажатой” в **PRESSKEY**.

### **Пример:**

IF Action = 'Add'	!При добавлении записи в поле мемо
Cus:MemoControl = FORMAT(TODAY(),@D1) & ' ' & FORMAT(CLOCK(),@T4)	
PRESSKEY(EnterKey)	!Установить дату и время в 1 строку мемо
END	! а пользователь предоставить вторую строку

**SETKEYCODE (назначить код клавиши)****SETKEYCODE**(код клавиши)

**SETKEYCODE** Устанавливает код клавиши, который будет возвращен при вызове функции KEYCODE.

*код клавиши* Целочисленная константа или символическое имя кода клавиши.

Оператор **SETKEYCODE** устанавливает внутренний код клавиши, который будет возвращен при вызове функции KEYCODE. Код клавиши в буфер клавиатуры не заносятся.

**Пример:**

SETKEYCODE(0800h) !Функция KEYCODE возвратит 0800h

**Смотри также:** KEYCODE, Символические имена Кодов клавиш



## Функции работы с клавиатурой

### KEYBOARD (возвратить код первый в буфере)

#### KEYBOARD()

Функция **KEYBOARD** возвращает код нажатой клавиши, лежащий первым в буфере клавиатуры. Функцию используют для выяснения: есть ли нажатые клавиши, которые ожидают обработки операторами ASK или ACCEPT.

Тип возвращаемых данных: LONG

#### Пример:

```
LOOP UNTIL KEYBOARD()           !Ждать нажатия клавиши
ASK
IF KEYCODE() = EscKey THEN BREAK .      !Выход из цикла по Esc
END
```

**Смотри также:** ASK, ACCEPT, Символические имена Кодов клавиш

### KEYCHAR (возвратить ASCII код )

#### KEYCHAR()

Функция **KEYCHAR** возвращает ASCII-значение клавиши, нажатой последней на момент возникновения события.

Тип возвращаемых данных: LONG

#### Пример:

```
ACCEPT                          !Ждать события
CASE KEYCHAR()                 !Обработка последней нажатой клавиши
OF 'A' TO 'Z'                  ! верхний регистр?
DO ProcessUpper
OF 'a' TO 'z'                  ! нижний регистр?
DO ProcessLower
END
END
```

**Смотри также:** ASK, ACCEPT, SELECT, Символические имена Кодов клавиш

**KEYCODE (возвратить код последней клавиши)**

**KEYCODE()**

Функция **KEYCODE** возвращает код клавиши, нажатой последней на момент возникновения события, либо значение кода клавиши, которое было установлено последней процедурой SETKEYCODE.

Тип возвращаемых данных: LONG

**Пример:**

```
ACCEPT                                !Обработать экран
CASE KEYCODE()                       !Обработка кода нажатой клавиши
OF UpKey                             ! "стрелка вверх"
  DO GetRecordUp                     ! прочитайте запись
OF DownKey                           ! "стрелка вниз"
  DO GetRecordDn                     ! прочитайте запись
END
END
```

**Смотри также:** ASK, ACCEPT, SELECT, Символические имена Кодов клавиш

**KEYSTATE (возвратить состояние клавиатуры)**

**KEYSTATE()**

Функция **KEYSTATE** возвращает битовую карту, сформированную последним вызовом функции KEYCODE, в которой отражено состояние SHIFT, CTRL, ALT, CAPS LOCK, NUM LOCK, SCROLL LOCK, INSERT и любой из функциональных клавиш. Упомянутая карта располагается в старшем байте возвращаемого SHORT-значения.

```
x . . . . . Insert      (8000h)
. x . . . . . Scroll Lock (4000h)
. . x . . . . Num Lock   (2000h)
. . . x . . . Caps Lock  (1000h)
. . . . x . . Функцион.  (0800h)
. . . . . x . . Alt      (0400h)
. . . . . . x . Ctrl     (0200h)
. . . . . . . x Shift    (0100h)
```

Тип возвращаемых данных: SHORT

**Пример:**

ACCEPT	!Обработать экран
CASE KEYCODE()	!Обработка кода нажатой клавиши
OF EnterKey	!Пользователь нажал клавишу Enter
IF BAND(KEYSTATE(),0800h)	!Отследить Enter в цифровой панели
PRESSKEY(TabKey)	! а пользователю выдать как Tab
END	
END	
END	

**Смотри также:** KEYCODE, BAND

## Функции поддержки окон стандарта Windows

### COLRDIALOG (выбор цвета)

**COLORDIALOG**([заголовок],[цвет])

**COLORDIALOG** Выводит на экран стандартное Windows-окно выбора цвета, в котором пользователь осуществляет свой выбор.

*заголовок* Строковая константа или переменная с текстом заголовка, который будет размещен в окне выбора цвета. Если параметр опущен, то заголовок по умолчанию заимствуется у Windows.

*цвет* Целочисленная переменная типа LONG, в которую заносится значение выбранного цвета.

Функция **COLORDIALOG** выводит на экран стандартное Windows-окно выбора цвета и возвращает значение выбранного пользователем цвета в переменной, указанной параметром цвет. При вызове функции значение параметра цвет устанавливает выбор цвета по умолчанию, предоставляемый пользователю в окне выбора.

**COLORDIALOG** возвращает 0, если в окне выбора пользователем была нажата кнопка Cancel, и 1, если была нажата кнопка Ok.

Тип возвращаемых данных: SHORT

#### Пример:

```
MDIChild1 WINDOW('Child One'), AT(0,0,320,200),MDI,MAX,HVSCROLL
                                     ! Поля окна
END
ColorNow LONG
CODE
IF NOT COLORDIALOG('Choose Box Color',ColorNow)
    ColorNow = 000000FFh           !Для Cancel - по умолчанию - синий
END
OPEN(MDIChild1)
BOX(100,50,100,50,ColorNow)      !Заданный пользователем цвет
                                     !прямоугольника
```

### FILEDIALOG (выбор файла)

**FILEDIALOG**([заголовок],[файл],[расширения],[флаг])

**FILEDIALOG** Выводит на экран стандартное Windows-окно выбора файла, в котором пользователь осуществляет свой выбор.

*заголовок* Строковая константа или переменная с текстом заголовка, который

будет размещен в окне выбора файла. Если параметр опущен, то заголовок по умолчанию заимствуется у Windows.

*файл* Строковая переменная, в которую будет помещено имя выбранного файла.

*расширения* Строковая константа или переменная, в которой для List Files of Type выпадающего окна списка - указываются желаемые расширения файлов. Если параметр опущен, то расширение по умолчанию - все файлы (\*.\*)).

*флаг* Целочисленная константа или переменная, содержащая битовый массив для обозначения действий, которые следует выполнить с файлом.

Функция **FILEDIALOG** выводит на экран стандартное Windows-окно выбора файла и возвращает имя выбранного пользователем файла в переменной, указанной параметром файл. При вызове функции значение параметра файл устанавливает выбор файла по умолчанию, предоставляемый пользователю в окне выбора.

Функция **FILEDIALOG** выводит или стандартное для Windows окно Open... или стандартное окно Save... . По умолчанию открывается окно Open... , если такой файл не существует, пользователю выдается предупреждение и файл не открывается. В случае открытия окна Save... пользователь предупреждается о том, что такой файл уже есть и файл не сохраняется.

В строке параметра расширения должно присутствовать описание файла, за которым следует его маска. Элементы строки отделяются друг от друга символом вертикальной черты (|). Например, строка расширения: 'All Files|\*.\*|Clarion Source|\*.CLW' указывает два варианта выбора для окна списка List Files of Type. Первое указанное в строке расширения расширение файла будет расширением по умолчанию.

Параметр флаг представляет собой битовый массив, в котором обозначаются действия, которые следует выполнить с файлом (см. мнемонические имена соответствия для них в файле EQUATES.CLW).

Bit 0 Если этот бит равен 0 (0000b), то высвечивается окно Open..., а если 1, то Save...

Bit 1 Если этот бит равен 1 (0010b), FILEDIALOG не позволяет сменить каталог.

Bit 2 Если этот бит равен 1 (0100b), то FILEDIALOG не сообщает об ошибках в случае уже наличия такого файла при операции сохранения или отсутствия при операции открытия файла.

Bit 3 Если этот бит равен 1 (1000b), то FILEDIALOG допускает множественный выбор в списке. Результат выбора возвращается разделяемый пробелами.

FILEDIALOG возвращает 0, если в окне выбора пользователем была нажата кнопка

Cancel, и 1, если была нажата кнопка Ok. Если пользователь изменил путь к файлу, использованный в диалоговом окне, то текущий для вашей прикладной программы каталог тоже изменится, если только бит 1 параметра флаг не установлен в единицу. Это свойство операционной системы Windows. Если нужно, чтобы пользователь мог просматривать другие каталоги и при этом текущий каталог прикладной программы не изменялся, сохраните перед выполнением функции FILEDIALOG путь к текущему каталогу с помощью функции PATH(), а после выполнения FILEDIALOG снова восстановите его оператором SETPATH().

Тип возвращаемых данных: SHORT

### Пример:

```

ViewTextFile PROCEDURE
ViewQue  QUEUE                      !Список изображаемых в поле LIST элементов
        STRING(255)
        END
FileName  STRING(64),STATIC         !Переменная имени файла
ViewFile  FILE,DRIVER('ASCII'),NAME(FileName),PRE(View)
Record    RECORD
        STRING(255)
        END
        END
SavePath  STRING(64)
MDIChild1 WINDOW('View Text File'), AT(0,0,320,200),MDI,SYSTEM,HVSCROLL
        LIST,AT(0,0,320,200),USE(?L1),FROM(ViewQue),HVSCROLL
        END

CODE
SavePath = PATH()                  !Запомнить текущий каталог
IF NOT FILEDIALOG('Choose File To View',FileName,'Text|*.TXT|Source|*.CLW',0)
    SETPATH(SavePath)              !Восстановить текущий каталог
    RETURN                         !Возврат, если файл не выбран
END
SETPATH(SavePath)                  !Восстановить текущий каталог
OPEN(ViewFile)                    !Открыть файл
IF ERRORCODE() THEN RETURN .      ! возврат по ошибке
SET(ViewFile)                     !Начать с начала файла
LOOP
    NEXT(ViewFile)                 !Читать каждую строку
    IF ERRORCODE() THEN BREAK .    !Закончить цикл по концу файла
    ViewQue = View:Record          !Подготовить текст для queue
    ADD(ViewQue)                   ! и добавить элемент в queue
END
CLOSE(ViewFile)                   !Закрыть файл
OPEN(MDIChild1)                   ! и открыть окно
ACCEPT                            !Разрешить пользователю читать текст, а

```

END	! выход из ACCEPT-цикла - только при выборе
	! элемента Close системного меню
FREE(ViewQue)	! Освободить queuee-память
RETURN	! и вернуться в программу

## FONTDIALOG (выбор шрифта)

**FONTDIALOG**(*[заголовок]*,*[начертание]*,*[размер]*,*[цвет]*,*[стиль]*,*[дополнение]*)

**FONTDIALOG** Выводит на экран стандартное Windows-окно выбора шрифта, в котором пользователь осуществляет свой выбор.

*заголовок* Строковая константа или переменная с текстом заголовка, который будет размещен в окне выбора шрифта. Если параметр опущен, то заголовок по умолчанию заимствуется у Windows.

*начертание* Строковая переменная, в которую возвращается имя шрифта

*размер* Целочисленная переменная, в которой возвращается размер (в пунктах) шрифта

*цвет* Целочисленная переменная типа LONG, в трех младших байтах которой возвращаются красная, зеленая и синяя компоненты цвета. выбранного шрифта.

*стиль* Целочисленная переменная, в которой возвращается толщина и стиль выбранного шрифта

*дополнение* Целочисленная константа или переменная, которая задает наличие дополнительных экранных или принтерных шрифтов. Значение 0 этого параметра означает добавление экранных шрифтов, 1 - принтерных, а 2 - и тех, и других. Если этот параметр опущен, то используются только зарегистрированные в Windows шрифты.

Функция **FONTDIALOG** выводит на экран стандартное Windows-окно выбора шрифта и предоставляет пользователю возможность выбрать шрифт. При вызове функции значения параметров устанавливают значения параметров шрифта для выбора по умолчанию, предоставляемые пользователю в окне выбора. В этих параметрах возвращаются значения параметров выбранного пользователем шрифта, когда пользователь нажимает кнопку Ok в окне выбора. FONTDIALOG возвращает 0, если в окне выбора пользователем была нажата кнопка Cancel, и 1, если была нажата кнопка Ok.

Тип возвращаемых данных: SHORT

### Пример:

```
MDIChild1 WINDOW('View Text File'), AT(0,0,320,200),MDI,SYSTEM,HVSCROLL
!Управляющие поля окна
END
Typeface STRING(20)
```

```

FontSize LONG
FontColor LONG
FontStyle LONG
CODE
OPEN(MDICHild1)                !Открыть окно
IF NOT FONTDIALOG('Choose Display Font',Typeface,FontSize,FontColor,FontStyle,0)
  SETFONT(0,Typeface,FontSize,FontColor,FontStyle)    !Установить шрифт окна
ELSEND
  SETFONT(0,'Arial',12)          !Установить шрифт по умолчанию
END
ACCEPT
                                !Обработка окна
END

```

## PRINTERDIALOG (выбор шрифта)

**PRINTERDIALOG**([*заголовок*] [,*флаг*])

**PRINTERDIALOG** Выводит на экран стандартное Windows-окно выбора принтера, в котором пользователь осуществляет свой выбор.

*заголовок* Строковая константа или переменная с текстом заголовка, который будет размещен в окне выбора принтера. Если параметр опущен, то заголовок по умолчанию заимствуется у Windows.

*флаг* Числовая константа или переменная, ненулевое значение которой означает вывод диалогового окна “Установка принтера”, вместо диалогового окна “Выбор принтера”. Это то же окно, которое вызывается заданием атрибуту STD значения STD:PrintSetup для пункта меню.

Функция **PRINTERDIALOG** выводит на экран стандартное Windows-окно выбора принтера (или окно установки принтера) и во встроенную переменную PRINTER внутренней библиотеки возвращает осуществленный пользователем выбор принтера. Таким способом выбирается принтер по умолчанию для открываемой в дальнейшем структуры REPORT.

**PRINTERDIALOG** возвращает 0, если в диалоговом окне пользователем была нажата кнопка Cancel, и 1, если была нажата кнопка Ok.

Тип возвращаемых данных: SHORT

### Пример:

```

CustRpt REPORT, AT(1000,1000,6500,9000),THOUS,Font('Arial',12),PRE(Rpt)
                                !Структуры и управляющие поля отчета
END
CODE

```



```
IF NOT PRINTERDIALOG('Choose Printer')
RETURN                                !Если нажата Cancel - выход
END
OPEN(CustRpt)
```

## Процессы *Drag and Drop*

### CLIPBOARD (получить содержимое буфера обмена Windows)

**CLIPBOARD**( [*формат*] )

**CLIPBOARD** Возвращает содержимое буфера обмена Windows.  
*формат* Целочисленная константа или переменная, которая определяет формат содержимого буфера. Если этот параметр опущен, то подразумевается текст.

Функция **CLIPBOARD** возвращает содержимое буфера обмена Windows. Параметр формат по умолчанию имеет значение CF\_TEXT (как определено в интерфейсе прикладных программ - Windows API), однако можно указать любое отличное от этого значение (см. справочное руководство по API). В API определены следующие значения форматов буфера обмена.

CF_TEXT	1
CF_BITMAP	2
CF_METAFILEPICT	3
CF_SYLK	4
CF_DIF	5
CF_TIFF	6
CF_OEMTEXT	7
CF_DIB	8
CF_PALETTE	9
CF_PENDATA	10
CF_RIFF	11
CF_WAVE	12

Тип возвращаемого значения:      STRING

#### Пример:

```
Que1      QUEUE
          STRING(30)
          END
```

```

Que2      QUEUE
          STRING(30)
          END
WinOne    WINDOW,AT(0,0,160,400)
          LIST,AT(120,0,20,20),USE(?List1),FROM(Que1),DRAGID('List1')
          LIST,AT(120,120,20,20),USE(?List2),FROM(Que2),DROPID('List1','~FILE')
          END
          CODE
          OPEN(WinOne)
          ACCEPT
          CASE EVENT()
          OF EVENT:Drag                !При попытке перетаскивания
            IF DRAGID()                ! проверить удачно ли
              SETCLIPBOARD(Que1)      ! и установить данные для передачи
            END
          OF EVENT:Drop                !При успешном перетаскивании
            Que2 = CLIPBOARD()         ! взять передаваемую информацию
            ADD(Que2)                  ! и добавить ее в очередь
          END
          END

```

Смотри также: SETCLIPBOARD

## DRAGID (возвратить совпавший “поташил-отпустил” ярлык)

**DRAGID**([*процесс*],[*поле*])

**DRAGID** При успешном процессе “поташил-отпустил” возвращает совпадающие ярлыки источника и приемника

*процесс* Метка целочисленной переменной, в которой возвращается номер процесса, где расположено поле-источник. Если поле-источник принадлежит внешней программе, то параметр процесс возвращается нулевым (0).

*поле* Метка целочисленной переменной, в которой возвращается метка соответствия поля-источника.

При успешном выполнении процедуры “поташил-отпустил” функция DRAGID возвращает совпадающие ярлыки источника и приемника. Если пользователь прервал исполнение процедуры, то DRAGID возвращает пустую строку (“”), в противном случае, возвращается первый совпавший для двух полей ярлык.

Тип возвращаемых данных: STRING

**Пример:**

```

Que1 QUEUE
  STRING(30)
END
Que2 QUEUE
  STRING(30)
END
WinOne WINDOW, AT(0,0,160,400)
  LIST,AT(120,0,20,20),USE(?List1),FROM(Que1),DRAGID('List1')
  !Можно “взять”, но нельзя “положить”
  LIST,AT(120,120,20,20),USE(?List2),FROM(Que2),DROPID('List1','~FILE')
  !Можно “положить” из List1 или менаджера файлов Windows
  !, но нельзя “взять”
END
CODE
OPEN(WinOne)
ACCEPT
CASE EVENT()
OF EVENT:Drag
  IF DRAGID()
    GET(Que1,CHOICE())
    SETCLIPBOARD(Que1)
  END
  !Если событие - попытка “взять”, то
  ! оценить успех попытки
  ! и подготовить данные для передачи
OF EVENT:Drop
  IF INSTRING(',',DROPID(),1,1)
    !проверить не несколько ли файлов из File Manager
    Que2 = SUB(DROPID(),1,INSTRING(',',DROPID(),1,1)-1) ! и взять только первый
    ADD(Que2)
    ! и добавить ее в список
  ELSE
    Que2 = DROPID()
    ! взять передаваемые данные, из List1 или File Manager
    ADD(Que2)
    ! и добавить их в очередь
  END
END
END
END

```

Смотри также: DROPID,SETDROPID

## DROPID (возвратить “поташил-отпустил”-строку)

**DROPID**([*процесс*],[*поле*])

### DROPID

При успешном процессе “поташил-отпустил” возвращает совпадающие ярлыки источника и приемника

*процесс*

Метка целочисленной переменной, в которой возвращается номер процесса, где расположено поле-источник. Если поле-источник принадлежит внешней программе, то параметр процесс возвращается нулевым (0).

*поле*

Метка целочисленной переменной, в которой возвращается метка

соответствия поля-источника.

При успешном выполнении процедуры “потачил-отпустил” функция DROPID возвращает либо совпадающие ярлыки источника и приемника (см. DRAGID), либо установленную процедурой SETDROPID строку. Если строка ‘~FILE’ включена в список параметров DROPID-атрибута, то функция DROPID возвращает список имен файлов, который был “взят” из окна File Manager системы Windows. Имена списка отделены друг от друга запятыми.

Тип возвращаемых данных: STRING

### Пример:

```
DragDrop  RPROCEDURE
```

```
Que1 QUEUE
```

```
  STRING(90)
```

```
  END
```

```
Que2 QUEUE
```

```
  STRING(90)
```

```
  END
```

```
WinOne WINDOW, AT(0,0,160,400)
```

```
  LIST,AT(120,0,20,20),USE(?List1),FROM(Que1),DRAGID('List1')
```

```
    !Можно “взять”, но нельзя “положить”
```

```
  LIST,AT(120,120,20,20),USE(?List2),FROM(Que2),DROPID('List1','~FILE')
```

```
!Можно “положить” из List1 или  окна File Manager системы Window, но нельзя “взять”
```

```
  END
```

```
CODE
```

```
OPEN(WinOne)
```

```
ACCEPT
```

```
  CASE EVENT()
```

```
  OF EVENT:Drag
```

```
      !Если событие - попытка “взять”, то
```

```
    IF DRAGID()
```

```
      ! оценить успех попытки
```

```
      GET(Que1,CHOICE())
```

```
      SETDROPID(Que1)
```

```
      ! и подготовить данные для передачи
```

```
    END
```

```
  OF EVENT:Drop
```

```
      !Если событие “положить” - успешно, то
```

```
    IF INSTRING(‘,’,DROPID(),1,1)
```

```
      !Если несколько файлов в File Manager, то
```

```
      Que2 = SUB(DROPID(),1,INSTRING(‘,’,DROPID(),1,1)-1)
```

```
      ! вернуть первый
```

```
      ADD(Que2)
```

```
      ! и добавить его в список
```

```
    ELSE
```

```
      Que2 = DROPID()
```

```
      ! вернуть из List1 или File Manager то,
```

```
      ! что “отпустил”
```

```
  ADD(Que2)
```

```
  ! и добавить его в список
```

```
  END
```

END  
END

Смотри также: DRAGID,SETDROPID

## SETCLIPBOARD (положить информацию в Windows-буфер обмена)

### SETCLIPBOARD(*строка*)

**SETCLIPBOARD**      Заносит информацию в буфер обмена системы Windows.  
*строка*                      Строковая константа или переменная, содержимое которой помещается в буфер обмена Windows.

Процедура **SETCLIPBOARD** заносит содержимое строки в Windows-буфер обмена. Предыдущее содержимое буфера теряется.

#### Пример:

```

Que1 QUEUE
  STRING(30)
  END
Que2 QUEUE
  STRING(30)
  END

WinOne WINDOW, AT(0,0,160,400)
  LIST,AT(120,0,20,20),USE(?List1),FROM(Que1),DRAGID('List1')
  !Можно "взять", но нельзя "положить"
  LIST,AT(120,120,20,20),USE(?List2),FROM(Que2),DROPID('List1')
  !Можно "положить" из List1, но нельзя "взять"
  END
CODE
OPEN(WinOne)
ACCEPT
CASE EVENT()
OF EVENT:Drag
  IF DRAGID()
    SETCLIPBOARD(Que1)
  END
  !Если событие - попытка "взять", то
  ! оценить успех попытки
  ! и подготовить данные для передачи
OF EVENT:Drop
  Que2 = CLIPBOARD()
  ADD(Que2)
  !Если событие "положить" - успешно, то
  ! прочитать "положенную" информацию
  ! и добавить ее в список
END
END
```

Смотри также: CLIPBOARD

## SETDROPID (задать строку, возвращаемую DROPID)

### SETDROPID(*строка*)

**SETDROPID** Задает значение, возвращаемое функцией DROPID

*строка* Строковая константа или переменная, значение которой возвратит функция DROPID.

Процедура **SETDROPID** задает значение, возвращаемое функцией DROPID. Это позволяет функции DROPID передавать данные процедуры “поташил-отпустил”. Если процедура “поташил-отпустил” совершается между полями разных Clarion-программ, то передачу данных возможно осуществить используя данный механизм.

### Пример:

```

DragDrop  RPROCEDURE
Que1 QUEUE
  STRING(30)
END
Que2 QUEUE
  STRING(30)
END

WinOne WINDOW, AT(0,0,160,400)
  LIST,AT(120,0,20,20),USE(?List1),FROM(Que1),DRAGID('List1')
  !Можно “взять”, но нельзя “положить”
  LIST,AT(120,120,20,20),USE(?List2),FROM(Que2),DROPID('List1','~FILE')
!Можно “положить” из List1 или окна File Manager системы Window, но нельзя “взять”
END

CODE
OPEN(WinOne)
ACCEPT
CASE EVENT()
OF EVENT:Drag
  IF DRAGID()
    SETDROPID(Que1)
  END
  !Если событие - попытка “взять”, то
  ! оценить успех попытки
  ! и подготовить данные для передачи
OF EVENT:Drop
  Que2 = DROPID()
  ADD(Que2)
  !Если событие “положить” - успешно, то
  ! вернуть из List1 или File Manager то, что “отпустил”
  ! и добавить его в список
END
END

```

**Смотри также:** DRAGID,DROPID

## Поддержка INI-файлов

### GETINI (возвратить элемент INI-файла)

**GETINI**(раздел, элемент[,умолчание][,файл])

<b>GETINI</b>	Возвращает содержимое элемента INI-файла.
<i>раздел</i>	Строковая константа или переменная, содержащая имя раздела INI-файла, в котором (разделе) расположен элемент.
<i>элемент</i>	Строковая константа или переменная, содержащая имя той установки, начение которой требуется получить.
<i>умолчание</i>	Строковая константа или переменная, содержащая значение по умолчанию, которое будет возвращено в случае отсутствия элемента. Если параметр опущен и элемент отсутствует, то GETINI возвратит пустую строку.
<i>файл</i>	Строковая константа или переменная, содержащая имя INI-файла, в котором осуществляется поиск (если полный путь к файлу не задан, то он ищется в каталоге Windows). Если этот параметр не указан, то поиск осуществляется в файле WIN.INI.

Функция **GETINI** возвращает значение элемента INI-файла стандарта Windows. INI-файл стандарта Windows представляет собой текстовый ASCII файл организованный следующим образом:

```
[имя раздела]
элемент=значение
следующий элемент=значение
```

Пусть, например, файл WIN.INI содержит такие элементы:

```
[int1]
sLanguage=enu
sCountry=United States
iCountry=1
```

Функция **GETINI** ищет в указанном файле элемент внутри указанного раздела. Она возвращает текст строки элемента, стоящий справа от символа равенства (=).

Тип возвращаемых данных: STRING

**Пример:**

```
Value STRING(30)
```

CODE

Value = GETINI('int1', 'sLanguage') !Прочитать элемент установки языка

Смотри также: PUTINI

## PUTINI (поместить элемент в INI-файл)

**PUTINI**(*раздел, элемент*[,*значение*][,*файл*])

<b>GETINI</b>	Устанавливает значение элемента INI-файла.
<i>раздел</i>	Строковая константа или переменная, содержащая имя раздела INI-файла, в котором (разделе) расположен элемент.
<i>элемент</i>	Строковая константа или переменная, содержащая имя той установки, значение которой требуется установить.
<i>значение</i>	Строковая константа или переменная, содержащая значение, которое будет помещено в элемент. Пустая строка ("") производит "пустую" установку в элементе. Если параметр опущен, то элемент удаляется.
<i>файл</i>	Строковая константа или переменная, содержащая имя файла, в котором осуществляется поиск (если полный путь к файлу не задан, то он ищется в каталоге Windows). Если параметр не указан, то PUTINI размещает элемент в файле WIN.INI.

Процедура **PUTINI** помещает значение в элемент INI-файла стандарта Windows. INI-файл стандарта Windows представляет собой текстовый ASCII файл организованный следующим образом:

```
[имя раздела]
элемент=значение
следующий элемент=значение
```

Пусть, например, файл WIN.INI содержит такие элементы:

```
[windows]
spooler=yes
load=nwpopup.exe
[int1]
sLanguage=enu
sCountry=United States
iCountry=1
```

Функция **PUTINI** ищет в указанном файле элемент внутри указанного раздела. Она заменяет текст строки элемента, стоящий справа от символа равенства (=), на новое значение. Если раздела и элемента в файле нет, то они будут созданы.



**Пример:**

CODE

PUTINI('MyApp','SomeSetting','Initialized')

!Поместить установку в WIN.INI

PUTINI('MyApp','ASetting','2',MYAPP.INI')

!Поместить установку в MYAPP.INI

**Смотри также:** GETINI



## Глава 9. Создание печатных документов

### Документы в Windows

В основу создания документов (отчетов) в системе Clarion для Windows положена концепция постраничного (а не построчного как в более старых генераторах отчетов) вывода документа на печать. При таком подходе на принтер посылается не отдельная готовая строка, а целиком подготовленная страница. Теперь значительную часть работы по печатанию документов (освобождая от нее программиста) может взять на себя “ядро программы печати” библиотеки времени исполнения, а основой его работы служат атрибуты, указанные при описании структуры REPORT.

Вот лишь некоторые процедуры, которые “ядро программы печати” Clarion-библиотеки времени исполнения выполняет за программиста:

- \* На каждой странице печатает трафарет, который затем заполняется данными
- \* Производит подсчет итоговых (totals) величин ( счетчиков, сумм, усреднений, минимумов и максимумов)
- \* Осуществляет автоматическую обработку перехода на новую страницу, не забывая при этом печатать верхние и нижние колонтитулы страниц
- \* Осуществляет автоматическую обработку завершения раздела, не забывая при этом печатать верхние и нижние колонтитулы разделов
- \* Полностью контролирует ситуацию с “висячими строками”.

Такая автоматизация существенно сокращает программы печати сложных документов, облегчая тем самым труд программиста. Поскольку “ядро программы печати” готовит для вывода на печать сразу всю страницу, то верхний и нижний колонтитулы утрачивают двойственность своего предназначения, связанную как с размещением текста, так и с воздействием на последовательность вывода на печать, сохраняя только воздействие на последовательность вывода. Верхние колонтитулы (заголовки) печатаются в начале последовательности, нижние - в ее конце, при этом их истинное положение на странице не имеет значения. Например, можно так расположить нижний колонтитул страницы, содержащий ее итоговые значения, что он напечатается в начале страницы.

### Переход на новую страницу

---

Ситуация перехода на новую страницу возникает когда оператор PRINT не в состоянии разместить структуру DETAIL на текущей странице. Последнее может быть связано либо с недостатком места, либо с присутствием в структуре DETAIL одного из атрибутов PAGEBEFORE или PAGEAFTER. Ниже представлена последовательность действий при возникновении ситуации перехода на новую страницу:

- 1 Если в структуре REPORT присутствует страничный FOOTER, то печатается нижний колонтитул, начиная с позиции, указанной его атрибутом AT.

- 2     Счетчик страниц увеличивается на единицу.
- 3     Если в структуре REPORT присутствует структура FORM, то печатается FORM, начиная с позиции, указанной ее атрибутом AT.
- 4     Если в структуре REPORT присутствует страничный HEADER, то печатается верхний колонтитул, начиная с позиции, указанной его атрибутом AT.

**Процедуры работы с документом**

**REPORT (объявить структуру документа)**

<i>метка</i>	<b>REPORT</b> ( <i>[имяпри]</i> ), <b>AT</b> ( ), <b>[PRE</b> ( ), <b>[LANDSCAPE]</b> , <b>[PREVIEW]</b> , <b>[ FONT</b> ( ), <b>[,COLOR</b> ( )   <b>THOUS</b>   ] <b>[, PAPER</b>   <b>MM</b>     <b>POINTS</b>    <b>[FORM</b> <i>поля</i> <b>END ]</b> <b>[HEADER</b> <i>поля</i> <b>END ]</b>
<i>метка</i>	<b>DETAIL</b> <i>поля</i> <b>END</b>
<i>метка</i>	<b>[BREAK</b> ( ) <i>структуры раздела</i> <b>END]</b> <b>[FOOTER</b> <i>поля</i> <b>END]</b> <b>END</b>

<b>REPORT</b>	Начинает описание структуры данных документа.
<i>метка</i>	Имя, посредством которого программа осуществляет ссылку на структуру.
<i>имяпри</i>	Именуется процесс печати, запускаемый Администратором Печати системы Windows (Windows Print Manager).
<b>AT</b>	Указывает размер области, предназначенной для вывода документа, и ее положение по отношению к левому верхнему углу страницы.
<b>FONT</b>	Указывает шрифт по умолчанию для всех управляющих полей документа. Если FONT отсутствует, то используется шрифт по умолчанию для принтера.
<b>PRE</b>	Определяет префикс меток структур документа

<b>LANDSCAPE</b>	Устанавливает “альбомный”-режим вывода на печать (когда ширина страницы больше ее длины). Если LANDSCAPE не указан, то по умолчанию - “портретный”-режим (когда длина страницы больше ее ширины).
<b>THOUS</b>	Для атрибутов, использующих координаты, устанавливается единица измерения равная тысячной доле дюйма.
<b>MM</b>	Для атрибутов, использующих координаты, устанавливается единица измерения равная одному миллиметру.
<b>POINTS</b>	Для атрибутов, использующих координаты, устанавливается единица измерения равная одному пункту. 72 пункта составляют один дюйм как по вертикали, так и по горизонтали.
<b>PREVIEW</b>	Устанавливает вывод документа в метафайлы Windows (.WMF); в один файл выводится одна страница документа.
<b>PAPER</b>	Задаст размер бумаги для печати отчета. Если этот параметр опущен, то используется размер бумаги принятый по умолчанию для данного принтера.
<b>COLOR</b>	Задаст цвет фона отчета и цвет лент отчета по умолчанию
<b>FORM</b>	Структура, описывающая трафарет, который печатается на каждой странице.
<i>поля</i>	Выходные поля документа
<b>HEADER</b>	Структура верхнего колонтитула, который печатается в начале каждой страницы.
<b>DETAIL</b>	Структура тела документа.
<b>BREAK</b>	Структура раздела документа, указывающая переменную, изменение значения которой приводит к завершению раздела.
<i>структуры раздела</i>	В их число входят структуры HEADER, FOOTER, DETAIL и/или вложенные BREAK-структуры.
<b>FOOTER</b>	Структура нижнего колонтитула, который печатается в конце каждой страницы.

Оператор **REPORT** открывает описание структуры данных документа. Описание структуры REPORT должно завершаться оператором END или символом точки (.). Внутри REPORT располагаются ее компоненты - структуры FORM, HEADER, DETAIL, FOOTER, BREAK, которые и формируют выходной документ. Структура REPORT должна быть явно открыта оператором OPEN.

Наличие атрибута PREVIEW в объявлении структуры REPORT приводит к записи выходного документа в Windows-метафайлы (.WMF) - по одной странице в файл. PREVIEW указывает имя QUEUE-структуры, в которую заносятся имена метафайлов. Для просмотра документа нужно создать окно с полем IMAGE и установить свойство этого поля - {PROP:Text}, используя значение QUEUE-элемента (имя файла). Это дает возможность пользователю перед выводом документа на печать просмотреть его на экране компьютера.

Атрибут AT определяет на каждой странице область, в которой печатается каждая структура DETAIL. То же относится к структурам HEADER и FOOTER, содержащимся внутри структуры BREAK (групповым заголовкам и итогам).

Единственное, что может (и должно) быть выведено на печать оператором PRINT - это структуры DETAIL. Все другие структуры (HEADER, FOOTER, FORM) автоматически печатаются в соответствующих местах выходного документа.

Структура FORM печатается на каждой странице, исключая те, структуры DETAIL которых объявлены с атрибутом ALONE. “Внешний вид” структуры создается один раз - на начальной стадии подготовки документа к печати. С помощью FORM можно создавать трафареты, которые впоследствии будут заполнены структурами HEADER, DETAIL и FOOTER. Верхний и нижний колонтитулы страницы не входят в структуру BREAK. При переходе на новую страницу они печатаются автоматически.

С помощью структуры BREAK осуществляется разбиение документа на разделы. В состав структуры могут входить HEADER и FOOTER (верхний и нижний колонтитулы раздела), DETAIL-структуры, а также вложенные BREAK-структуры. Кроме того, BREAK может включать в себя несколько структур DETAIL. Структуры HEADER и FOOTER внутри BREAK - колонтитулы раздела. Они автоматически выводятся на печать при изменении значения переменной, указанной в операторе BREAK.

В отличие от оконных структур APPLICATION и WINDOW, структура REPORT не может быть объектом по умолчанию для установок значений свойств во время выполнения. Поэтому, при выполнении установок свойств REPORT-полей необходимо использовать либо метку структуры REPORT, либо предварительно выполнить процедуру SETTARGET, посредством которой REPORT объявляется текущим объектом. SETTARGET используется и тогда, когда в документ требуется вставить графические образы, поскольку в качестве объекта вывода графические функции всегда используют текущий объект.

### Пример:

```
CustRpt  REPORT,AT(1000,1000,6500,9000),THOUS,FONT('Arial',12),PRE(Rpt)
        FORM,AT(1000,1000,6500,9000)
        IMAGE('LOGO.BMP'),AT(0,0,1200,1200),USE(?I1)
        STRING(@n3),AT(6000,500,500,500),PAGENO
        END
        HEADER,AT(1000,1000,6500,1000)
        STRING('ABC Company'),AT(3000,500,1500,500),FONT('Arial',18)
        END
Break1   BREAK(Pre:Key1)
        HEADER,AT(0,0,6500,1000)
        STRING('Group Head'),AT(3000,500,1500,500),FONT('Arial',18)
        END
```

```

Detail    DETAIL,AT(0,0,6500,1000)
          STRING(@N$11.2'),AT(6000,1500,500,500),USE(Pre:F1)
          END
          FOOTER,AT(0,0,6500,1000)
          STRING('Group Total:'),AT(5500,500,1500,500)
          STRING(@N$11.2'),AT(6000,500,500,500),USE(Pre:F1),SUM,RESET(Pre:Key1)
          END
          END
          FOOTER,AT(1000,1000,6500,1000)
          STRING('Page Total:'),AT(5500,1500,1500,500)
          STRING(@N$11.2'),AT(6000,1500,500,500),USE(Pre:F1),SUM,PAGE
          END
          END
          !Конец объявления REPORT

```

```

CODE
OPEN(CustReport)
SET(DataFile)
LOOP
  NEXT(DataFile)
  IF ERRORCODE() THEN BREAK.
  PRINT(Rpt:Detail)
END
CLOSE(CustReport)

```

### АТ (назначение области печати тела документа)

**АТ([x][,y][,ширина][,высота])**

<b>АТ</b>	Определяет положение и размер страничной области, предназначенной для печати тела документа.
<i>x</i>	Целочисленная константа, константное выражение для указания горизонтального положения левого верхнего угла области тела документа.
<i>y</i>	Целочисленная константа, константное выражение для указания вертикального положения левого верхнего угла области тела документа.
<i>ширина</i>	Целочисленная константа, константное выражение для указания ширины области тела документа.
<i>высота</i>	Целочисленная константа, константное выражение для указания высоты области тела документа.

Атрибут АТ структуры REPORT определяет положение и размер страничной области, предназначенной для печати тела документа. В нее входит область для печати HEADER- и FOOTER-структур раздела, которые указаны в структурах BREAK.

Если не присутствует какой-либо из атрибутов THOUS, MM или POINTS, то - по умолчанию - значения параметров x, y, ширина и высота заданы в условных единицах

измерения. За условные единицы принимаются одна четверть усредненной ширины символа и одна восьмая его усредненной высоты. Ясно, что величина условной единицы зависит от установленного для документа шрифта по умолчанию. За основу измерений берется либо шрифт, указанный FONT-атрибутом документа, либо установленный для принтера системный шрифт по умолчанию.

**Пример:**

```
CustRpt1  REPORT,AT(1000,1000,6500,9000),THOUS  !1 дюйм отступа по периметру
! страницы размера 8.5" x 11"
! для области тела документа
!структура документа
END

CustRpt2  REPORT,AT(72,72,468,648),POINTS      !1 дюйм отступа по периметру
! страницы размера 8.5" x 11"
! для области тела документа
!структура документа
END
```

**FONT (назначение для документа шрифта по умолчанию)**

**FONT**(*[начертание]* [*,размер]* [*,цвет]* [*,стиль]*)

<b>FONT</b>	Устанавливает шрифт по умолчанию для вывода на печать.
<i>начертание</i>	Строковая константа, в которой указано имя шрифта. Если параметр опущен, то используется шрифт по умолчанию для принтера.
<i>размер</i>	Целочисленная константа, указывающая размер шрифта (в пунктах). Если параметр не указан, то принимается размер шрифта по умолчанию для принтера.
<i>цвет</i>	Целочисленная константа типа LONG, в трех младших байтах которой указываются значения красной, зеленой и синей компонент цвета шрифта, либо символическое имя значения цвета стандарта Windows. Если параметр не указан, то устанавливается черный цвет.
<i>стиль</i>	Целочисленная константа, константное выражение, символическое имя, определяющие толщину и стиль шрифта. Если параметр опущен, то принимается нормальная толщина шрифта.

Атрибут FONT структуры REPORT устанавливает для всех REPORT-полей шрифт по умолчанию для вывода на печать. Этот шрифт используется когда у поля либо нет собственного атрибута FONT, либо он есть, но уже сама печатаемая структура, в которой расположено это поле, собственного FONT-атрибута не имеет.

Параметром начертание можно задать имя любого зарегистрированного в Windows



шрифта, который поддерживается драйвером принтера. К ним относятся TrueType-шрифты, поддерживаемые большинством принтеров. В файле EQUATES.CLW находятся символические имена значений для стандартных стилей. Значение параметра стиль в диапазоне от 0 до 1000 определяет толщину шрифта. К этому значению можно прибавлять значения для наклона, подчеркивания и перечеркивания текста. Указанные символические имена находятся в файле EQUATES.CLW:

FONT:thin	EQUATE(100)
FONT:regular	EQUATE(400)
FONT:bold	EQUATE(700)
FONT:italic	EQUATE(01000H)
FONT:underline	EQUATE(02000H)
FONT:strikeout	EQUATE(04000H)

### Пример:

```
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS, |
      FONT('Arial', 12,,FONT:Bold+FONT:Italic)
      !структура документа
END
```

## PRE (назначение префикса для меток структур документа)

**PRE**(*префикс*)

### PRE

*префикс*

Предусматривает префикс меток структур документа.

Строковая переменная, содержащая префикс для меток в структуре REPORT. Допустимыми являются символы алфавита, цифровые символы (0 - 9) и символ подчеркивания. Префикс должен начинаться с буквы или со знака подчеркивания. По традиции длина префикса не превышает трех символов, хотя он может быть и длиннее.

Атрибут **PRE** структуры REPORT предусматривает префикс меток структур DETAIL и BREAK.

Используют префикс для того, чтобы можно было отличить друг от друга переменные, которые имеют одинаковые имена, но принадлежат разным структурам. Если в программе нужно сделать ссылку на такую переменную, то перед именем переменной ставится префикс, за которым следует символ двоеточия (Pre:LABEL). Кроме того, для этой же цели можно использовать синтаксис уточнения имен.

### Пример:

```
Report REPORT,PRE('Rpt')
DetailOne DETAIL           !Rpt:DetailOne - ссылка на структуру
```

!Report controls  
 END ! в программе  
 END

**Смотри также:** Зарезервированные слова

## PREVIEW (направить вывод документа в метафайлы)

### PREVIEW(*список*)

**PREVIEW** Указывает, что созданный документ будет записан в метафайлы Windows, при этом каждая страница будет записана в отдельный файл.  
*список* Метка структуры QUEUE или QUEUE-элемента для хранения имен метафайлов.

Атрибут **PREVIEW** структуры REPORT направляет созданный документ в метафайлы Windows (.WMF) - каждую страницу документа в отдельный файл. PREVIEW указывает имя списка, в котором будут храниться имена метафайлов. Имена метафайлов - внутренние имена, временно создаваемые Clarion-библиотекой - представляют собой полную спецификацию файла (длиной до 64 символов, включая устройство и путь). Метафайлы удаляются с диска при закрытии структуры REPORT оператором CLOSE.

Для просмотра документа нужно создать окно с полем IMAGE и установить свойство этого поля - {PROP:text}, используя список, хранящий имена файлов. Это дает возможность пользователю перед выводом документа на печать просмотреть его на экране компьютера. Доступное только во время исполнения свойство {PROP:flushpreview}, будучи установленным в значение ON, "сбрасывает" метафайлы на принтер.

### Пример:

```
SomeReport PROCEDURE
  WMFQue  QUEUE !Список хранения .WMF метафайлов
  PageImage STRING(64)
  END
  NextEntry BYTE(1) !Счетчик элементов списка
  Report REPORT,PREVIEW(WMFQue.PageImage) !REPORT с PREVIEW-атрибутом
  DetailOne DETAIL !Поля документа
  END
  END
ViewReport WINDOW('View Report'),AT(0,0,320,200),MDI,MAX,HVSCROLL
  IMAGE('',AT(0,0,320,180),USE(?ImageField)
  BUTTON('View Next Page'),AT(0,180,60,20),USE(?NextPage),DEFAULT
  BUTTON('Print Report'),AT(80,180,60,20),USE(?PrintReport)
  BUTTON('Exit Without Printing'),AT(160,180,60,20),USE(?ExitReport)
```

```

END
CODE
OPEN(Report)
SET(SomeFile)                                !Процедура создания документа
LOOP
  NEXT(SomeFile)
  IF ERRORCODE() THEN BREAK.
  PRINT(DetailOne)
END
ENDPAGE(Report)
OPEN(ViewReport)                             !Открыть окно просмотра документа
GET(WMFQue,NextEntry)                         !Прочсть первый элемент списка
?ImageField{PROP:text} = WMFQue.PagelImage    !Прочсть первую страницу
документа
ACCEPT
CASE ACCEPTED()
OF ?NextPage
  NextEntry += 1                               !Увеличить на 1 счетчик элементов списка
  IF NextEntry > RECORDS(WMFQue) THEN CYCLE. !Проверка на конец документа
  GET(WMFQue,NextEntry)                       !Прочсть следующий элемент списка
  ?ImageField{PROP:text} = WMFQue.PagelImage   !Прочсть следующую страницу
документа
  DISPLAY                                     ! и вывести ее на экран
OF ?PrintReport
  Report{PROP:flushpreview} = TRUE            !"Сбросить" файлы на принтер
  BREAK ! и выйти из процедуры
OF ?ExitReport
  BREAK !Выйти из процедуры
END
END
CLOSE(ViewReport)                             !Закрыть окно
FREE(WMFQue)                                  !Освободить память "из-под" списка
CLOSE(Report)                                 !Закрыть REPORT (удаляя .WMF файлы)
RETURN ! и вернуться в программу

```

## PAPER (установить размер бумаги для отчета)

**PAPER**( [*min*] [*ширина*] [*высота*])

### **PAPER**

Определяет размер листа бумаги для отчета.

- min* Целочисленная константа или мнемоническая метка соответствия, задающая стандартный в Windows размер бумаги. Мнемонические метки соответствия содержатся в файле PRNPROP.CLW .
- ширина* Целочисленная константа или константное выражение, которое задает ширину листа бумаги.
- высота* Целочисленная константа или константное выражение, которое задает высоту

листа бумаги.

Атрибут **PAPER** в структуре REPORT определяет размеры листа бумаги для отчета. Если в качестве типа бумаги выбрано свойство PAPER:User, то требуются параметры высота и ширина.

По умолчанию величины этих параметров задаются в условных единицах, если только для структуры REPORT не заданы дополнительно атрибуты THOUS, MM, или POINTS. За условные единицы принимаются одна четверть усредненной ширины символа и одна восьмая его усредненной высоты. Ясно, что величина условной единицы зависит от установленного для документа шрифта по умолчанию. За основу измерений берется либо шрифт, указанный FONT-атрибутом документа, либо установленный для принтера системный шрифт по умолчанию.

### Пример:

```
CustRpt1  REPORT,AT(1000,1000,6500,9000),THOUS,PAPER(PAPER:Custom,8500,7000)
                                                ! Печать на бумаге 8.5 x 7 дюймов
          !объявления элементов отчета
          END
```

```
CustRpt2  REPORT,AT(72,72,468,648),POINTS,PAPER(PAPER:A4)
                                                ! Печать на бумаге формата A4
          ! объявления элементов отчета
          END
```

## LANDSCAPE (выбор ориентации страницы)

### LANDSCAPE

Присутствие атрибута **LANDSCAPE** в структуре REPORT означает, что по умолчанию вывод документа на печать будет производиться в режиме “альбом”. Если атрибут LANDSCAPE отсутствует, то по умолчанию вывод документа на печать будет производиться в режиме “портрет”.

### Пример:

```
Report  REPORT,PRE('Rpt'),LANDSCAPE  !Режим по умолчанию - “альбом”
      !Структура документа
      END
```

**COLOR (установить цвет фона отчета)****COLOR**( *цвет* )**COLOR***цвет*

Задает цвет фона отчета.

Константа тип LONG или ULONG, или мнемоническая метка константы, содержащая в младших трех байтах (байты 0, 1, и 2) красный, зеленый и синий компоненты, составляющие цвет фона. Или это мнемоническая метка для стандартного в Windows цвета.

Атрибут **COLOR** указывает цвет фона для отчета и, используемый по умолчанию цвет фона для структур DETAIL, HEADER, FOOTER и FORM, для которых не указан атрибут COLOR.

Мнемонические имена стандартных для Windows цветов, содержатся в файле EQUATES.CLW. При выполнении программы в каждом видеорежиме Windows автоматически подбирает наиболее соответствующий заданному цвет. Стандартные цвета Windows пользователь может изменить в меню Управляющая Панель. После такого изменения все элементы управления, для которых использовались стандартные цвета, будут раскрашены по-новому.

**Пример:**

```
RptOne REPORT,AT(0,0,160,400),COLOR(00FF0000h)    !Синий фон
END
```

**THOUS, MM, POINTS (выбор единицы измерений координат документа)****THOUS****MM****POINTS**

Атрибуты THOUS, MM, POINTS указывают в каких единицах измерения будет производиться отсчет координат при размещении полей структуры REPORT.

THOUS устанавливает единицу измерения в одну тысячную долю дюйма, MM устанавливает единицу измерения в один миллиметр, POINTS устанавливает единицу измерения в один пункт (как по вертикали, так и по горизонтали 72 пункта составляют один дюйм).

Если ни один из атрибутов не присутствует, то по умолчанию измерения производятся в условных единицах. За условные единицы принимаются одна четверть усредненной ширины символа и одна восьмая его усредненной высоты. Ясно, что величина условной единицы зависит от установленного для документа шрифта по умолчанию. За основу измерений берется либо шрифт, указанный FONT-атрибутом документа, либо установленный для принтера системный шрифт по умолчанию.

## Структуры, образующие документ

### BREAK (объявление структуры раздела документа)

<i>метка</i>	<b>BREAK</b> ( <i>переменная</i> )[,USE( )]
	<i>структуры раздела</i>
	<b>END</b>

**BREAK**                      Объявляет структуру раздела.

*метка*                      Имя, посредством которого в программе производится ссылка на структуру.

*переменная*              Переменная, изменение значения которой приводит к завершению раздела.

**USE**                      Мнемоническая метка для ссылок на структуру BREAK в исполняемых операторах.

*структуры раздела*      В их число входят структуры HEADER, FOOTER, DETAIL и/или вложенные BREAK-структуры.

Структура **BREAK** указывает переменную, изменение значения которой приводит к завершению раздела. BREAK-структура должна заканчиваться символом точки или оператором END. В состав структуры могут входить HEADER и FOOTER, DETAIL-структуры, а также вложенные BREAK-структуры. Внутри BREAK структуры HEADER и FOOTER обязаны быть в единственном числе, однако, допустимы несколько структур DETAIL и/или вложенные BREAK-структуры.

В BREAK структуре HEADER и FOOTER представляют собой верхний и нижний колонтитулы раздела. Они автоматически выводятся на печать при изменении значения переменной, указанной в операторе BREAK.

#### Пример:

CustRpt REPORT	!Документ с информацией о клиенте
Break1 BREAK(SomeVariable)	
HEADER	! начало верхнего колонтитула раздела
	!поля документа
END	! конец верхнего колонтитула раздела
GroupDet DETAIL	
	!поля документа
END	! конец описания тела документа
FOOTER	! начало нижнего колонтитула раздела
	!поля документа
END	! конец нижнего колонтитула раздела
END	! конец описания раздела
END	!Конец описания структуры документа

## DETAIL (объявить структуру документа)

```

метка    DETAIL,AT() [FONT()] [ALONE] [ABSOLUTE] [PAGEBEFORE()]
           [PAGEAFTER()] [WITHPRIOR()] [WITHNEXT()][USE()][COLOR( )]
           поля
           END ]

```

**DETAIL**                    Объявляет компоненты, которые должны быть напечатаны в теле документа.

*метка*                    Имя, посредством которого программа осуществляет ссылку на структуру.

**AT**                        Указывает смещение и минимальные значения ширины и высоты DETAIL внутри области, размеры которой установлены атрибутом AT структуры REPORT.

**FONT**                    Указывает шрифт по умолчанию для всех управляющих полей DETAIL-структуры. Если FONT отсутствует, то используется шрифт атрибута FONT структуры REPORT, а если и он отсутствует, то - шрифт по умолчанию для принтера.

**ALONE**                    Объявляет, что структура DETAIL должна быть напечатана без вывода на страницу структур FORM, (страничной) HEADER и (страничной) FOOTER.

**ABSOLUTE**                Объявляет, что структура DETAIL будет напечатана в фиксированном месте страницы.

**PAGEBEFORE**            Объявляет, что структура DETAIL будет напечатана в начале новой страницы по завершению обычных действий, связанных с переходом на новую страницу.

**PAGEAFTER**            Объявляет, что структура DETAIL сначала будет напечатана, а затем посредством выполнения обычных действий, связанных с переходом на новую страницу, - начнется новая страница.

**WITHPRIOR**            Объявляет, что структура DETAIL будет напечатана на той же странице, что и предшествующая ей структура, а это - либо DETAL, либо HEADER раздела, либо FOOTER раздела.

**WITHNEXT**            Объявляет, что вместе со структурой DETAIL на той же странице будет напечатана и непосредственно следующая за ней структура, а это - либо DETAL, либо HEADER раздела, либо FOOTER раздела.

**USE**                     Метка соответствия поля, с помощью которой программа осуществляет ссылку на структуру.

**COLOR**                    Задает цвет фона для данной структуры DETAIL и цвет фона по умолчанию для всех элементов внутри нее.

*поля*                     Выходные поля документа

Структура DETAIL объявляет компоненты, которые должны быть напечатаны в теле документа. DETAIL-структура должна заканчиваться символом точки или оператором END. Более одной DETAIL могут входить в состав структуры REPORT.

Поскольку автоматически структура **DETAIL** на печать не выводится, то это нужно делать явно, используя оператор **PRINT**. Следовательно, каждая **DETAIL**-структура, которую нужно вывести на печать, должна быть снабжена меткой.

Вывод структуры **DETAIL** на печать можно осуществлять вполне произвольно. Наличие нескольких **DETAIL**-структур позволяет формировать выходной документ желаемым образом. Все определяется логикой программной процедуры, которая организует вывод документа на печать.

### Пример:

CustRpt	REPORT	!Документ с информацией о клиенте
	HEADER	! начало верхнего колонтитула страницы
		!элементы структуры
	END	! конец верхнего колонтитула страницы
CustDetail1	DETAIL	! начало <b>DETAIL</b> -объявления
		!элементы структуры
	END	! конец <b>DETAIL</b> -объявления
CustDetail2	DETAIL	! начало <b>DETAIL</b> -объявления
		!элементы структуры
	END	! конец <b>DETAIL</b> -объявления
	END	!Конец описания структуры документа
CODE		
OPEN(CustRpt)		
SET(SomeFile)		
LOOP		
NEXT(SomeFile)		
IF ERRORCODE() THEN BREAK.		
IF SomeCondition		
PRINT(CustDetail1)		
ELSE		
PRINT(CustDetail2)		
END		
END		
CLOSE(CustRpt)		

**Смотри также:** **PRINT**



**FOOTER (структура нижнего колонтитула страницы или раздела)**

```
FOOTER,AT()[,FONT()[,ABSOLUTE][,PAGEBEFORE()][,PAGEAFTER() ]
[,WITHPRIOR() ] [,WITHNEXT()[,ALONE)][,USE()][,COLOR( ]
```

*поля*

```
END ]
```

<b>FOOTER</b>	Объявляет нижний колонтитул страницы или раздела.
<b>AT</b>	Указывает размер и расположение нижнего колонтитула.
<b>FONT</b>	Указывает шрифт по умолчанию для всех управляющих полей FOOTER-структуры. Если FONT отсутствует, то используется шрифт атрибута FONT структуры REPORT, а если и он отсутствует, то - шрифт по умолчанию для принтера.
<b>ABSOLUTE</b>	Объявляет, что структура FOOTER будет напечатана в фиксированном месте страницы. Атрибут допустим только для FOOTER в структуре BREAK (положение FOOTER для страницы всегда фиксировано)
<b>PAGEBEFORE</b>	Объявляет, что структура FOOTER будет напечатана в начале новой страницы по завершению обычных действий, связанных с переходом на новую страницу. Атрибут допустим только для FOOTER в структуре BREAK.
<b>PAGEAFTER</b>	Объявляет, что структура FOOTER сначала будет напечатана, а затем посредством выполнения обычных действий, связанных с переходом на новую страницу, - начнется новая страница. Атрибут допустим только для FOOTER в структуре BREAK.
<b>WITHPRIOR</b>	Объявляет, что структура FOOTER будет напечатана на той же странице, что и предшествующая ей структура, а это - либо DETAL, либо HEADER раздела, либо FOOTER раздела. Атрибут допустим только для FOOTER в структуре BREAK.
<b>WITHNEXT</b>	Объявляет, что вместе со структурой FOOTER на той же странице будет напечатана и непосредственно следующая за ней структура, а это - либо DETAL, либо HEADER раздела, либо FOOTER раздела. Атрибут допустим только для FOOTER в структуре BREAK.
<b>ALONE</b>	Объявляет, что структура FOOTER (для раздела) должна быть напечатана без вывода на страницу структур FORM, (страничной) HEADER и (страничной) FOOTER.
<b>USE</b>	Метка соответствия поля, с помощью которой программа осуществляет ссылку на структуру.
<b>COLOR</b>	Задает цвет фона для данной структуры FOOTER и цвет фона по умолчанию для всех элементов внутри нее.
<i>поля</i>	Выходные поля документа

Структура **FOOTER** объявляет выходные данные, которые будут напечатаны в конце каждой страницы или раздела. FOOTER-структура должна заканчиваться символом точки

или оператором END.

Структура FOOTER, если она расположена вне BREAK-структуры, представляет собой нижний колонтитул страницы. В REPORT должно быть не более одной страничной FOOTER-структуры. При возникновении ситуации перехода на новую страницу нижний колонтитул страницы автоматически выводится на печать. Положение колонтитула в напечатанной странице определяется его AT-атрибутом.

С помощью структуры BREAK осуществляется разбиение документа на разделы. В состав структуры могут входить HEADER и FOOTER (верхний и нижний колонтитулы раздела), DETAIL-структуры, а также вложенные BREAK-структуры. Кроме того, BREAK может включать в себя несколько структур DETAIL. Структуры HEADER и FOOTER внутри BREAK - колонтитулы раздела. Они автоматически выводятся на печать - со следующей допустимой позиции области тела документа (область определяется атрибутом AT структуры REPORT) - при изменении значения переменной, указанной в операторе BREAK. В структуре BREAK должно быть не более одной FOOTER-структуры.

### Пример:

CustRpt REPORT	!Документ с информацией о клиенте
FOOTER	! начало объявления FOOTER страницы
	!поля документа
END	! конец объявления FOOTER
Break1 BREAK(SomeVariable)	
GroupDet DETAIL	
	!поля документа
END	! конец описания тела документа
FOOTER	! начало нижнего колонтитула раздела
	!поля документа
END	! конец нижнего колонтитула раздела
END	! конец описания раздела
END	!Конец описания структуры документа

### FORM (структура трафарета страницы)

```
FOOTER,AT()[,FONT()][,USE()][,COLOR( )]
    поля
END ]
```

**FORM** Объявляет структуру документа, которая печатается на каждой странице.

**AT** Указывает размер структуры и ее расположение по отношению к левому верхнему углу страницы.

**FONT** Указывает шрифт по умолчанию для всех управляющих полей

FORM-структуры. Если FONT отсутствует, то используется шрифт атрибута FONT структуры REPORT, а если и он отсутствует, то - шрифт по умолчанию для принтера.

**USE** Метка соответствия поля, с помощью которой программа осуществляет ссылку на структуру.

**COLOR** Задает цвет фона для данной структуры FOOTER и цвет фона по умолчанию для всех элементов внутри нее.

*поля* Выходные поля документа

**FORM** объявляет структуру документа, которая печатается на каждой странице выходного документа (исключая те, в которых печатаются структуры DETAIL с атрибутом ALONE). FORM-структура должна заканчиваться символом точки или оператором END. В REPORT должно быть не более одной FORM-структуры. При возникновении ситуации перехода на новую страницу FORM автоматически выводится на печать.

“Внешний вид” структуры FORM создается один раз - на начальной стадии подготовки документа к печати. Ее расположение на странице не влияет на расположение в этой странице других структур документа. Например, структуры могут быть напечатаны “поверх” напечатанной FORM-структуры. Поэтому, FORM чаще всего используют для создания трафарета страницы, который затем заполняется последующими структурами HEDER, DETAIL и FOOTER. Ее можно использовать и для нанесения “водяных знаков”, и для графического оформления рамки страницы.

### Пример:

```
CustRpt REPORT                                !Документ с информацией о клиенте
  FORM
    IMAGE('LOGO.BMP'),AT(0,0,1200,1200),USE(?I1)
    STRING(@N3),AT(6000,500,500,500),PAGE NO
  END
GroupDet DETAIL
  !report controls
  . . !Конец описания структуры документа
```

### HEADER (верхний колонтитул страницы или раздела)

```
HEADER,AT()[,FONT()][,ABSOLUTE][,PAGEBEFORE()][,PAGEAFTER() ]
    [,WITHPRIOR() ] [,WITHNEXT()][,ALONE)][,USE()][,COLOR( )]
    поля
  END ]
```

**HEADER** Объявляет верхний колонтитул страницы или раздела.

**AT** Указывает размер и расположение верхнего колонтитула.

**FONT** Указывает шрифт по умолчанию для всех управляющих полей

HEADER-структуры. Если FONT отсутствует, то используется шрифт атрибута FONT структуры REPORT, а если и он отсутствует, то - шрифт по умолчанию для принтера.

**ABSOLUTE**           Объявляет, что структура HEADER будет напечатана в фиксированном месте страницы. Атрибут допустим только для HEADER в структуре BREAK (положение HEADER для страницы всегда фиксировано)

**PAGEBEFORE**       Объявляет, что структура HEADER будет напечатана в начале новой страницы по завершению обычных действий, связанных с переходом на новую страницу. Атрибут допустим только для HEADER в структуре BREAK.

**PAGEAFTER**       Объявляет, что структура HEADER сначала будет напечатана, а затем - посредством выполнения обычных действий, связанных с переходом на новую страницу, - начнется новая страница. Атрибут допустим только для HEADER в структуре BREAK.

**WITHPRIOR**       Объявляет, что структура HEADER будет напечатана на той же странице, что и предшествующая ей структура, а это - либо DETAL, либо HEADER раздела, либо FOOTER раздела. Атрибут допустим только для HEADER в структуре BREAK.

**WITHNEXT**       Объявляет, что вместе со структурой HEADER на той же странице будет напечатана и непосредственно следующая за ней структура, а это - либо DETAL, либо HEADER раздела, либо FOOTER раздела. Атрибут допустим только для HEADER в структуре BREAK.

**ALONE**           Объявляет, что структура HEADER (для раздела) должна быть напечатана без вывода на страницу структур FORM, (страничной) HEADER и (страничной) FOOTER.

**USE**           Метка соответствия поля, с помощью которой программа осуществляет ссылку на структуру.

**COLOR**           Задает цвет фона для данной структуры HEADER и цвет фона по умолчанию для всех элементов внутри нее.

*поля*           Выходные поля документа

Структура **HEADER** объявляет выходные данные, которые будут напечатаны в начале каждой страницы или раздела. HEADER-структура должна заканчиваться символом точки или оператором END.

Структура HEADER, если она расположена вне BREAK-структуры, представляет собой верхний колонтитул страницы. В REPORT должно быть не более одной страничной HEADER-структуры. При возникновении ситуации перехода на новую страницу верхний колонтитул страницы автоматически выводится на печать. Положение колонтитула в напечатанной странице определяется его AT-атрибутом.

С помощью структуры BREAK осуществляется разбиение документа на разделы. В

состав структуры могут входить HEADER и FOOTER (верхний и нижний колонтитулы раздела), DETAIL-структуры, а также вложенные BREAK-структуры. Кроме того, BREAK может включать в себя несколько структур DETAIL. Структуры HEADER и FOOTER внутри BREAK - колонтитулы раздела. Они автоматически выводятся на печать - со следующей допустимой позиции области тела документа (область определяется атрибутом AT структуры REPORT) - при изменении значения переменной, указанной в операторе BREAK. В структуре BREAK должно быть не более одной HEADER-структуры.

**Пример:**

CustRpt REPORT	!Документ с информацией о клиенте
HEADER	! начало верхнего колонтитула страницы
	!поля документа
END	! конец верхнего колонтитула
Break1 BREAK(SomeVariable)	
HEADER	! начало верхнего колонтитула раздела
	!поля документа
END	! конец верхнего колонтитула раздела
GroupDet DETAIL	
	!поля документа
END	! конец описания тела документа
END	! конец описания раздела
END	!Конец описания структуры документа

## Атрибуты Структур Документа

### ABSOLUTE (печатать с фиксированной позиции)

#### ABSOLUTE

Атрибут **ABSOLUTE** гарантирует, что при выводе на печать структура DETAIL, или HEADER (раздела), или FOOTER (раздела) (HEADER и FOOTER внутри BREAK) попадет в фиксированное место страницы. Если структура объявлена с атрибутом ABSOLUTE, то параметры x и y ее AT-атрибута указывают положение относительно левого верхнего угла страницы.

#### Пример:

```
CustRpt  REPORT
        HEADER
        !элементы структуры
        END
CustDetail1  DETAIL
        !элементы структуры
        END
CustDetail2  DETAIL,ABSOLUTE           ! DETAIL с фиксированной позицией
        !элементы структуры
        END
END
```

### ALONE (печатать с фиксированной позиции)

#### ALONE

Атрибут **ALONE** устанавливает, что вывод на печать структуры DETAIL, или HEADER (раздела), или FOOTER (раздела) (HEADER и FOOTER внутри BREAK) будет производиться без печатания FORM, HEADER (страницы), FOOTER (страницы) (HEADER и FOOTER вне BREAK) структур. Используется, в основном, для вывода на печать страницы с наименованием документа и для печати страниц с общими итогами.

#### Пример:

```
CustRpt  REPORT
TitlePage  DETAIL,ALONE               !Структура страницы наименования документа
        !элементы структуры
        END
CustDetail  DETAIL
        !элементы структуры
        END
```

FOOTER  
!элементы структуры  
END  
END

**АТ (указание области страницы для печати структуры)**

АТ([ <i>x</i> ][, <i>y</i> ][, <i>ширина</i> ][, <i>высота</i> ])	
АТ	Определяет положение и размер страничной области, предназначенной для Создание печатных документов печати структуры.
<i>x</i>	Целочисленная константа, константное выражение для указания горизонтального положения левого верхнего угла области для печати структуры.
<i>y</i>	Целочисленная константа, константное выражение для указания вертикального положения левого верхнего угла области для печати структуры.
<i>ширина</i>	Целочисленная константа, константное выражение для указания минимальной ширины области для печати структуры.
<i>высота</i>	Целочисленная константа, константное выражение для указания минимальной высоты области для печати структуры.

Атрибут **АТ** выводимой на печать структуры выполняет одну из двух возможных функций (в зависимости от того, в какой структуре он расположен).

В структурах **FORM**, страничных **HEADER** и **FOOTER** (вне структуры **BREAK**) атрибут **АТ** определяет положение и размер области внутри страницы, куда будет выводиться структура. Параметры *x* и *y* указывают положение относительно левого верхнего угла страницы.

Если атрибут **АТ** относится к структурам **DETAIL**, **HEADER** и **FOOTER** для раздела (внутри структуры **BREAK**), то вывод указанных структур на печать подчиняется следующим правилам (при условии, когда отсутствует атрибут **ABSOLUTE**):

- \* Параметры *ширина* и *высота* определяют минимальные размеры области на странице для вывода структуры.
- \* Собственно вывод структуры на печать производится с текущей свободной позиции в области печати тела документа (область печати тела документа определена атрибутом **АТ** структуры **REPORT**).
- \* Параметры *x* и *y* определяют смещение относительно текущей свободной позиции в области печати тела документа.
- \* Вывод первой структуры на печать начинается с левого верхнего угла (со смещением, установленным **АТ**-атрибутом структуры) области печати тела документа
- \* Вывод всех последующих структур на печать будет производиться относительно конечной позиции предыдущей структуры:
- \* Если есть место для печати следующей структуры рядом с уже напечатанной, то

она будет напечатана рядом.

\* Если рядом места нет, то она будет напечатана ниже предыдущей.

Если не присутствует какой-либо из атрибутов THOUS, MM или POINTS, то - по умолчанию - значения параметров x, y, ширина и высота заданы в условных единицах измерения. За условные единицы принимаются одна четверть усредненной ширины символа и одна восьмая его усредненной высоты. Ясно, что величина условной единицы зависит от установленного для документа шрифта по умолчанию. За основу измерений берется либо шрифт, указанный FONT-атрибутом документа, либо установленный для принтера системный шрифт по умолчанию.

### Пример:

```
CustRpt1  REPORT,AT(1000,1000,6500,9000),THOUS
          !1 дюйм отступа по периметру  страницы
          HEADER,AT(1000,1000,6500,1000)  !Позиция относительно страницы
          !элементы структуры              !полоса шириной 1 дюйм в начале страницы
          END
CustDetail1  DETAIL,AT(0,0,6500,1000)      !Позиция относительно предыдущей структуры
          !элементы структуры              ! полоса шириной 1 дюйм внутри страницы
          END
CustDetail2  DETAIL,ABSOLUTE,AT(1000,8000,6500,1000) !Позиция относительно страницы
          !элементы структуры              ! полоса шириной 1 дюйм ближе к концу  страницы
          END
          FOOTER,AT(1000,9000,6500,1000)  !Позиция относительно страницы
          !элементы структуры              !полоса шириной 1 дюйм в конце страницы
          END
          END
```

## COLOR (установить цвет фона)

**COLOR**( *цвет* )

### COLOR

*цвет*

Задает цвет фона печати.

Константа тип LONG или ULONG, или мнемоническая метка константы, содержащая в младших трех байтах (байты 0, 1, и 2) красный, зеленый и синий компоненты, составляющие цвет фона. Или это мнемоническая метка для стандартного в Windows цвета.

Атрибут **COLOR** указывает цвет фона для структур DETAIL, HEADER, FOOTER и FORM, или используемый по умолчанию цвет фона элементов, для которых не указан атрибут COLOR.

Мнемонические имена стандартных для Windows цветов, содержатся в файле EQUATES.CLW. При выполнении программы в каждом видеорежиме Windows



автоматически подбирает наиболее соответствующий заданному цвет. Стандартные цвета Windows пользователь может изменить в меню Управляющая Панель. После такого изменения все элементы управления, для которых использовались стандартные цвета, будут раскрашены по-новому.

### Пример:

```
RptOne    REPORT,AT(0,0,160,400),COLOR(00FF0000h)    !По умолчанию синий фон
          HEADER,COLOR(0000FF00h)    !Зеленый фон заголовка
          !элементы структуры
          END
CustDetail1  DETAIL                                !использовать цвет фона по умолчанию
          !элементы структуры
          END
          FOOTER,COLOR(000000FFh)    !Красный цвет фона нижнего колонтитула
          !элементы структуры
          END
          END
```

## FONT (назначение для документа шрифта по умолчанию)

**FONT**([начертание][,размер][,цвет][,стиль])

<b>FONT</b>	Устанавливает шрифт по умолчанию для вывода на печать.
<i>начертание</i>	Строковая константа, в которой указано имя шрифта. Если параметр опущен, то используется шрифт по умолчанию для принтера.
<i>размер</i>	Целочисленная константа, указывающая размер шрифта (в пунктах). Если параметр не указан, то принимается размер шрифта по умолчанию для принтера.
<i>цвет</i>	Целочисленная константа типа LONG, в трех младших байтах которой указываются значения красной, зеленой и синей компонент цвета шрифта, либо символическое имя значения цвета стандарта Windows. Если параметр не указан, то устанавливается черный цвет.
<i>стиль</i>	Целочисленная константа, константное выражение, символическое имя, определяющие толщину и стиль шрифта. Если параметр опущен, то принимается нормальная толщина шрифта.

Атрибут **FONT** структур FORM, DETAIL, HEADER и FOOTER устанавливает для всех полей в структуре, у которых нет собственного атрибута FONT, шрифт по умолчанию для вывода на печать.

Параметром начертание можно задать имя любого зарегистрированного в Windows шрифта, который поддерживается драйвером принтера. К ним относятся TrueType-шрифты, поддерживаемые большинством принтеров. В файле EQUATES.CLW находятся

символические имена значений для стандартных стилей. Значение параметра стиль в диапазоне от 0 до 1000 определяет толщину шрифта. К этому значению можно прибавлять значения для наклона, подчеркивания и перечеркивания текста. Указанные символические имена находятся в файле EQUATES.CLW:

FONT:thin	EQUATE(100)
FONT:regular	EQUATE(400)
FONT:bold	EQUATE(700)
FONT:italic	EQUATE(01000H)
FONT:underline	EQUATE(02000H)
FONT:strikeout	EQUATE(04000H)

### Пример:

CustRpt	REPORT, FONT('Arial', 12)	!Шрифт по умолчанию: 12 пунктов Arial
	HEADER('Arial', 18,, FONT:bold)	!18 пунктов жирный Arial для HEADER
		!элементы структуры
	END	
CustDetail1	DETAIL	!DETAIL использует шрифт по умолчанию
		!элементы структуры
	END	
	FOOTER, FONT('Arial', 12, 00FF0000h)	!12 пунктов Красный Arial для FOOTER
		!элементы структуры
	END	
	END	

## PAGEAFTER (затем - переход на новую страницу)

**PAGEAFTER**(*новая страница*)

**PAGEAFTER** Указывает на то, что сначала структура будет напечатана, а затем - переход на новую страницу.

*новая страница* Целочисленная константа, константное выражение для указания номера страницы, который будет напечатан на следующей странице. Если параметр не указан, то берется увеличенный на единицу номер текущей страницы.

Атрибут **PAGEAFTER** указывает, что одна из структур DETAIL, HEADER раздела или FOOTER раздела (HEADER и FOOTER - внутри BREAK) - после того, как будет напечатана - инициирует переход на новую страницу. То есть, сначала будет напечатана структура, у которой есть атрибут PAGEAFTER, сразу за ней - страничный FOOTER, а уж затем - FORM и страничный HEADER.

Параметр новая страница, если он присутствует, перенастраивает автоматическую

нумерацию страниц на указанный номер.

### Пример:

```
CustRpt REPORT
  HEADER
    !элементы структуры
  END
Break1 BREAK(SomeVariable)
  HEADER
    !элементы структуры
  END
CustDetail DETAIL
  !элементы структуры
  END
  FOOTER,PAGEAFTER           !FOOTER раздела инициирует переход на
                              ! новую страницу
  !элементы структуры
  END
  END
  FOOTER
    !элементы структуры
  END
  END
```

## PAGEBEFORE (сначала - переход на новую страницу)

**PAGEBEFORE**(*новая страница*)

**PAGEBEFORE** Указывает на то, что сначала произойдет переход на новую страницу, а затем структура будет напечатана.

*новая страница* Целочисленная константа, константное выражение для указания номера станицы, который будет напечатан на следующей странице. Если параметр не указан, то берется увеличенный на единицу номер текущей страницы.

Атрибут **PAGEBEFORE** указывает, что после того, как произойдет переход на новую страницу, будет напечатана одна из структур DETAIL, HEADER раздела или FOOTER раздела (HEADER и FOOTER - внутри BREAK). То есть, сначала будет напечатан страничный FOOTER, затем - FORM и страничный HEADER. Структура, у которой есть атрибут PAGEBEFORE, будет напечатана только после завершения этих, связанных с переходом на новую страницу, действий.

Параметр *новая страница*, если он присутствует, перенастраивает автоматическую нумерацию страниц на указанный номер.

**Пример:**

```

CustRpt  REPORT
  HEADER
    !элементы структуры
  END
Break1   BREAK(SomeVariable)
  HEADER,PAGEBEFORE           !HEADER раздела инициирует переход на
                               ! новую страницу
    !элементы структуры
  END
CustDetail  DETAIL
  !элементы структуры
  END
  FOOTER
    !элементы структуры
  END
  END
  FOOTER
    !элементы структуры
  END
  END

```

**USE (определить метку соответствия структуры)**

**USE**(метка[,номер])

<b>USE</b>	Устанавливает для структуры метку соответствия поля
<i>метка</i>	Метка соответствия поля, которая используется программой для ссылки на структуру.
<i>номер</i>	Целочисленная константа, определяющая номер, который присваивается компилятором метке соответствия поля для структуры.

Атрибут **USE** одной из структур FORM, BREAK, DETAIL, HEADER или FOOTER устанавливает для структуры метку соответствия поля. Это дает возможность операторам исполняемой программы осуществлять ссылки на структуру.

Существует аналогия между выводимыми на печать структурами в составе структуры REPORT и управляющими полями структуры WINDOW: и тем и другим компилятор присваивает положительные номера.

Параметр USE-атрибута номер позволяет указать желаемый номер, который компилятор присвоит структуре. Этот номер становится отправной точкой для последовательного присвоения номеров меткам соответствия тех полей и структур, в USE-атрибутах которых

отсутствует параметр номер. Номера таким структурам или полям присваиваются в порядке возрастания (или убывания), начиная с номера установленного последним.

### Пример:

```
BuildRpt PROCEDURE
CustRpt REPORT
    HEADER,USE(?PageHeader)      !Верхний колонтитул страницы
    !элементы структуры
    END
CustDetail DETAIL,USE(?Detail)    !Строка тела документа
    !элементы структуры
    END
    FOOTER,USE(?PageFooter)      !Нижний колонтитул страницы
    !элементы структуры
    END
END
CODE
PrintRpt(CustRpt,?Detail)        !Передать процедуре печати в качестве
                                !параметров структуру REPORT и номер ее
                                !DETAIL структуры

PrintRpt PROCEDURE(RptToPrint,DetailNumber)
CODE
OPEN(RptToPrint)                !Открыть эту REPORT-структуру
PRINT(RptToPrint,DetailNumber)  !Напечатать ее DETAIL-структуру
CLOSE(RptToPrint)               !Закрыть REPORT
```

## WITHNEXT (предотвратить отрыв от последующих)

### WITHNEXT(сородичи)

**WITHNEXT**                      Указывает, что структура будет всегда напечатана на той же странице, что и последующие, выведенные на печать оператором PRINT, структуры.

*сородичи*                      Целочисленная константа или константное выражение, определяющие сколько последующих структур должно быть напечатано на странице. Если параметр отсутствует, то на печать выводится одна структура.

Атрибут **WITHNEXT** указывает, что одна из структур DETAIL, HEADER раздела или FOOTER раздела (HEADER и FOOTER - внутри BREAK) будет всегда напечатана на одной странице вместе с указанным числом последующих, выведенных на печать оператором PRINT, структур. Тем самым гарантируется, что структура никогда не будет напечатана на странице отдельно, в отрыве от последующего текста. Отрыв от последующего текста означает, что, например, верхний колонтитул раздела или первый экземпляр DE-

TAIL из группы текстуально объединенных экземпляров напечатан на одной странице, а связанный с ним последующий текст выведен на другую страницу.

Если указан, параметр сородичи определяет сколько последующих структур должно быть напечатано на странице. При подсчете учитываются те последующие структуры, которые поступают из одной и той же, или из вложенной в нее, BREAK-структуры. Эти структуры должны быть связаны текстуально. Если структуры - не из той же, или вложенной в нее, BREAK-структуры, то они выводятся на печать, но не учитываются при подсчете числа сородичей.

### Пример:

```
CustRpt  REPORT
Break1   BREAK(SomeVariable)
          HEADER,WITHNEXT(2)      !Всегда печатается с 2 сородичами
          !элементы структуры
          END
CustDetail  DETAIL,WITHNEXT()      !Всегда печатается с 1 сородичем
          !элементы структуры
          END
          FOOTER
          !элементы структуры
          END
          END
          END
```

## WITHPRIOR (предотвратить отрыв от предыдущих)

### WITHPRIOR(*сородичи*)

**WITHPRIOR**            Указывает, что структура будет всегда напечатана на той же странице, что и предыдущие, выведенные на печать оператором PRINT, структуры.

*сородичи*              Целочисленная константа или константное выражение, определяющие сколько предыдущих структур должно быть напечатано на странице. Если параметр отсутствует, то на печать выводится одна структура.

Атрибут **WITHPRIOR** указывает, что одна из структур DETAIL, HEADER раздела или FOOTER раздела (HEADER и FOOTER - внутри BREAK) будет всегда напечатана на одной странице вместе с указанным числом предшествующих, выведенных на печать оператором PRINT, структур. Тем самым гарантируется, что структура никогда не будет напечатана на странице отдельно, в отрыве от предшествующего текста. Отрыв от предшествующего текста означает, что, например, нижний колонтитул раздела или последний экземпляр DETAIL из группы текстуально объединенных экземпляров

напечатан на новой странице, а связанный с ним предшествующий текст выведен на предыдущей странице.

Если указан, параметр `сородичи` определяет сколько предшествующих структур должно быть напечатано на странице. При подсчете учитываются те предшествующие структуры, которые поступают из одной и той же, или из вложенной в нее, `BREAK`-структуры. Эти структуры должны быть связаны текстуально. Если структуры - не из той же, или вложенной в нее, `BREAK`-структуры, то они выводятся на печать, но не учитываются при подсчете числа сородичей.

### Пример:

```
CustRpt  REPORT
Break1   BREAK(SomeVariable)
        HEADER
        !элементы структуры
        END
CustDetail  DETAIL,WITHPRIOR()      !Всегда печатается с 1 сородичем
        !элементы структуры
        END
        FOOTER,WITHPRIOR(2)        !Всегда печатается с 2 сородичами
        !элементы структуры
        END
        END
        END
        END
```

## Управляющие поля в структуре *REPORT*

### **BOX** (объявить поле окна в структуре документа)

**BOX,AT()[,USE()[,COLOR()[,FILL()[,ROUND][,HIDE][,LINEWIDTH( )]**

<b>BOX</b>	Размещает в REPORT окно прямоугольной формы.
<b>AT</b>	Определяет размер и местоположение поля. Если не указан, то значения по умолчанию устанавливаются библиотекой времени исполнения.
<b>USE</b>	Определяет для поля метку соответствия.
<b>COLOR</b>	Определяет цвет ограничивающей рамки поля. Если не указан, то для рамки устанавливается черный цвет.
<b>FILL</b>	Определяет цвет “раскраски” поля. Если не указан, то окно не закрашивается.
<b>ROUND</b>	Устанавливает, что углы у окна будут закругленной формы. Если не указан, то углы - обычной формы.
<b>HIDE</b>	Указывает, что поле не будет выведено на печать, пока не выполнится оператор UNHIDE.
<b>LINEWIDTH</b>	Задаёт толщину контура прямоугольника.

Поле **BOX** размещает окно прямоугольной формы в структуре REPORT. Атрибут AT указывает размеры окна и его местоположение относительно левого верхнего угла выводимой на печать структуры, в которую входит BOX.

#### Пример:

```
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail DETAIL,AT(0,0,6500,1000)
    BOX,AT(0,0,20,20),USE(?B1)      !Не раскрашено, черная рамка
    BOX,AT(20,20,20,20),ROUND      !Не раскрашено, скруглено, черная рамка
    BOX,AT(40,40,20,20),FILL(COLOR:ACTIVEBORDER)
    !Раскрашено, черная рамка
    BOX,AT(60,60,20,20),COLOR(COLOR:ACTIVEBORDER)
    !Не раскрашено, рамка цвета текущей активной рамки
END
END
```



**CHECK (объявить поле флагов в структуре документа)**

<b>CHECK</b> ( <i>текст</i> ), <b>AT</b> ( <i> </i> ), <b>USE</b> ( <i> </i> ), <b>FONT</b> ( <i> </i> ), <b>HIDE</b> ], <b>DISABLE</b> ],	<b>LEFT</b> ]
[ <b>TRN</b> ] [ <b>COLOR</b> ( <i> </i> )	<b>RIGHT</b>

**CHECK**

Размещает поле флагов в структуре REPORT.

*текст*

Строковая константа, текст которой изображается на экране в поле флагов.

**AT**

Определяет размер и местоположение поля. Если не указан, то значения по умолчанию устанавливаются библиотекой времени исполнения.

**USE**

Метка численной переменной, содержащей значение поля флагов : ноль (0=OFF) или единица (1=ON).

**FONT**

Определяет для поля шрифт вывода на экран.

**HIDE**

Указывает, что поле не будет выведено на печать, пока не выполнится оператор UNHIDE.

**DISABLE**

Устанавливает, что в REPORT будет уменьшена яркость изображения поля.

**TRN**

Задаст, что элемент имеет как бы прозрачный фон и печатается поверх предыдущего изображения.

**COLOR**

Задаст цвет фона для элемента.

**LEFT**

Указывает, что текст в поле будет сдвинут влево.

**RIGHT**

Указывает, что текст в поле будет сдвинут вправо (положение по умолчанию).

Поле **CHECK** размещает поле флагов в структуре REPORT. Атрибут AT указывает размеры поля флагов и его местоположение относительно левого верхнего угла выводимой на печать структуры, в которую входит CHECK.

**Пример:**

```
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail DETAIL,AT(0,0,6500,1000)
CHECK('1'),AT(0,0,20,20),USE(C1)
CHECK('2'),AT(20,80,20,20),USE(C2),LEFT
CHECK('3'),AT(0,100,20,20),USE(C3),FONT('Arial',12)
END
END
```

**VBX (объявить в документе импортируемое.VBX-поле)**

**VBX(текст),AT()[,CLASS()[,USE()[,DISABLE][,FONT()[,META]**  
**[,свойство( значение )]**

<b>VBX</b>	Помещает в структуру REPORT .VBX-поле, импортируемое из программной оболочки Visual Basic.
<i>текст</i>	Строковая константа, содержащая надпись для поля.
<b>AT</b>	Определяет размер и местоположение поля. Если не указан, то значения по умолчанию устанавливаются библиотекой времени исполнения.
<b>CLASS</b>	Указывает имя .VBX файла и тип поля.
<b>USE</b>	Определяет для поля метку соответствия.
<b>DISABLE</b>	Устанавливает, что в REPORT будет уменьшена яркость изображения поля.
<b>FONT</b>	Определяет для поля шрифт вывода на экран.
<b>META</b>	Устанавливает, что вывод на печать будет производиться как для метафайла Windows.
<b>HIDE</b>	Указывает, что поле не будет выведено на печать, пока не выполнится оператор UNHIDE.
<i>свойство</i>	Строковая константа, содержащая имя устанавливаемого свойства импортируемого поля.
<i>значение</i>	Строковая константа, содержащая значение свойства или символическое имя для значения свойства.

Управляющее поле **VBX** размещает в документе .VBX-поле, импортируемое из программной оболочки Visual Basic. Атрибут AT указывает размеры поля и его местоположение.

Атрибут свойство позволяет определить дополнительные установки свойств, которые могут понадобиться для .VBX-поля. Это - те свойства, которые необходимо установить для соответствующего функционирования поля; они не являются стандартными свойствами Clarion (как AT или USE). В поле следует задавать значения для тех свойств, которые определены для этого поля. Допустимые свойства и их значения могут быть определены в документации по применению импортируемого поля. Для одного VBX-поля возможно указание нескольких атрибутов свойство.

**Пример:**

```
Report REPORT
DetailOne DETAIL
    VBX,AT(0,0,120,320),CLASS('graph.vbx','graph'),'graphstyle'('2')
END
END
```

**ELLIPSE (объявить эллипс-поле в документа)****ELLIPSE,AT()[,USE()][,COLOR()][,FILL()][,HIDE] [,LINEWIDTH]**

<b>ELLIPSE</b>	Размещает “круглую” фигуру в структуре REPORT.
<b>AT</b>	Определяет размер и местоположение поля. Если не указан, то значения по умолчанию устанавливаются библиотекой времени исполнения.
<b>USE</b>	Определяет для поля метку соответствия.
<b>COLOR</b>	Определяет цвет границы эллипса. Если не указан, то для границы эллипса устанавливается черный цвет.
<b>FILL</b>	Определяет цвет “раскраски” поля. Если не указан, то эллипс не закрашивается.
<b>HIDE</b>	Указывает, что поле не будет выведено на печать, пока не выполнится оператор UNHIDE.
<b>LINEWIDTH</b>	Задаёт толщину контура эллипса.

Поле **ELLIPSE** размещает “круглую” фигуру в структуре REPORT. Атрибут AT указывает размеры поля и его местоположение. Эллипс прорисовывается внутри “ограничивающего прямоугольника”, размеры которого определяются параметрами x, y, ширина и высота атрибута AT. Параметры x и y определяют начальную точку относительно верхнего левого угла печатаемой структуры, содержащей эллипс, а параметры ширина и высота определяют горизонтальный и вертикальный размеры “ограничивающего прямоугольника”.

**Пример:**

```

CustRpt  REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail  DETAIL,AT(0,0,6500,1000)
    ELLIPSE,AT(0,0,20,20)           !Не раскрашен, черная граница
    ELLIPSE,AT(0,20,20,20),USE(?Ellipse1),DISABLE
                                     !Не раскрашен, черная граница, недоступен
    ELLIPSE,AT(20,20,20,20),ROUND !Не раскрашен, rounded, черная граница
    ELLIPSE,AT(40,40,20,20),FILL(COLOR:ACTIVEBORDER)
                                     !Раскрашен, черная граница
    ELLIPSE,AT(60,60,20,20),COLOR(COLOR:ACTIVEBORDER)
                                     !Не раскрашен, граница цвета текущей активной рамки
END
END

```

**GROUP (объявить группу полей документа)**

```

GROUP(текст),AT()поля [,USE()поля] [,FONT()поля] [,BOXED] [,HIDE] [,TRN] [,COLOR()поля]
END

```

<b>GROUP</b>	Объявляет группу полей, на которую можно ссылаться как на единое целое.
<i>текст</i>	Строковая константа с поясняющим текстом для группы полей. Данный текст будет выведен на печать только тогда, когда для GROUP указан атрибут BOXED.
<b>AT</b>	Определяет размер и местоположение поля. Если не указан, то значения по умолчанию устанавливаются библиотекой времени исполнения.
<b>USE</b>	Определяет для поля метку соответствия.
<b>FONT</b>	Определяет для поля шрифт вывода на экран и шрифт по умолчанию для всех полей в группе.
<b>BOXED</b>	Устанавливает, что группа полей будет очерчена одинарной рамкой, в верхнем элементе которой расположен текст.
<b>HIDE</b>	Указывает, что поле не будет выведено на печать, пока не выполнится оператор UNHIDE.
<b>TRN</b>	Задаст, что элемент имеет как бы прозрачный фон и печатается поверх предыдущего изображения.
<b>COLOR</b>	Задаст цвет фона для элемента.
<i>поля</i>	Объявления полей, на которые можно ссылаться как на GROUP.

Поле **GROUP** объявляет группу полей, на которую можно ссылаться как на единое целое. С его помощью создаются документы, которые на бумаге выглядят так же, как и на экране.

**Пример:**

```

CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail DETAIL,AT(0,0,6500,1000)
  GROUP('Group 1'),USE(!G1),AT(80,0,20,20),BOXED
  STRING(@S8),AT(80,0,20,20),USE(E5)
  STRING(@S8),AT(100,0,20,20),USE(E6)
END
  GROUP('Group 2'),USE(?G2),FONT('Arial',12)
  STRING(@S8),AT(120,0,20,20),USE(E7)
  STRING(@S8),AT(140,0,20,20),USE(E8)
END
END
END

```

**IMAGE (объявить в документе графическое поле)**

**IMAGE(файл),AT()[,USE()][,HIDE]**

<b>IMAGE</b>	Помещает в REPORT графический образ.
<i>файл</i>	Строковая константа с именем файла, который требуется вывести на печать. Указанный файл будет помещен в .EXE-файл в качестве ресурса.
<b>AT</b>	Определяет размер и местоположение поля. Если не указан, то значения по умолчанию устанавливаются библиотекой времени исполнения.
<b>USE</b>	Определяет для поля метку соответствия.
<b>HIDE</b>	Указывает, что поле не будет выведено на печать, пока не выполнится оператор UNHIDE.

Поле **IMAGE** помещает в REPORT графический образ, размер и местоположение для которого указывается AT-атрибутом поля. В качестве графического образа может выступать битовое отображение (.BMP), файл формата PaintBrush (.PCX), файл формата GIF (.GIF), файл формата JPEG (.JPG) или метафайл Windows (.WMF). В поле типа IMAGE не может выводиться пиктограмма (файл .ICO), потому что Windows не поддерживает печать графики в таком формате.

**Пример:**

```
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail DETAIL,AT(0,0,6500,1000)
    IMAGE('PIC.BMP'),AT(0,0,20,20),USE(?I1)
    IMAGE('PIC.WMF'),AT(40,0,20,20),USE(?I2)
    IMAGE('PIC. JPG'),AT(60,0,20,20),USE(?I3)
END
END
```

**LINE (объявить в документа поле линии)**

**LINE,AT()[,USE()][,COLOR()][,HIDE] [,LINEWIDTH( )]**

<b>LINE</b>	Размещает в REPORT прямую линию.
<b>AT</b>	Определяет размер и местоположение поля. Если не указан, то значения по умолчанию устанавливаются библиотекой времени исполнения.
<b>USE</b>	Определяет для поля метку соответствия.
<b>COLOR</b>	Определяет цвет линии. Если не указан, то для линии устанавливается черный цвет.
<b>HIDE</b>	Указывает, что поле не будет выведено на печать, пока не выполнится оператор UNHIDE.

**LINEWIDTH**                      Задает толщину линии.

Поле **LINE** размещает в REPORT прямую линию, область для которой указана AT-атрибутом поля.

Начальная точка линии определяется параметрами x и y атрибута AT. Положение конечной точки относительно начальной задается параметрами ширина и высота. Если их значения - положительные числа, то линия распространяется вправо и вниз от начальной точки. Если значение параметра ширина отрицательно, то линия распространяется влево от начальной точки; если значение параметра высота отрицательно, то линия распространяется влево от начальной точки. Если либо ширина, либо высота - нулевые, то линия либо вертикальна, либо горизонтальна.

<u>Ширина</u>	<u>Высота</u>	<u>Результат</u>
положительная	положительная	вправо и вниз от начальной точки
отрицательная	отрицательная	влево и вниз от начальной точки
положительная	отрицательная	вправо и вверх от начальной точки
отрицательная	положительная	влево и вверх от начальной точки
нулевая	положительная	вертикально вниз от начальной точки
нулевая	отрицательная	вертикально вверх от начальной точки
положительная	нулевая	горизонтально вправо от начальной точки
отрицательная	нулевая	горизонтально влево от начальной точки

**Пример:**

```
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail DETAIL,AT(0,0,6500,1000)
    LINE,AT(60,60,20,20),COLOR(COLOR:ACTIVEBORDER) !С цветом рамки
    LINE,AT(480,180,20,20),USE(?L2)
END
END
```

**LIST (объявить в документе поле списка)**

LIST,FROM(),AT()[,FONT()][,USE()][,HIDE][,   FORMAT()   ]	
	LEFT  [,TRN] [,COLOR( )]
	RIGHT
	CENTER
	DECIMAL

**LIST**                      Размещает в REPORT текущий элемент списка элементов данных.  
**FROM**                    Указывает источник данных для отображения в списке  
**AT**                        Определяет размер и местоположение поля. Если не указан, то

значения по умолчанию устанавливаются библиотекой времени исполнения.

<b>FONT</b>	Определяет для поля шрифт вывода на экран.
<b>USE</b>	Определяет для поля метку соответствия.
<b>HIDE</b>	Указывает, что поле не будет выведено на печать, пока не выполнится оператор UNHIDE.
<b>TRN</b>	Задаёт, что элемент имеет как бы прозрачный фон и печатается поверх предыдущего изображения.
<b>FORMAT</b>	Определяет формат вывода данных на печать
<b>LEFT</b>	Устанавливает, что данные внутри LIST будут выровнены слева.
<b>RIGHT</b>	Устанавливает, что данные внутри LIST будут выровнены справа.
<b>CENTER</b>	Устанавливает, что данные внутри LIST будут отцентрированы.
<b>DECIMAL</b>	Устанавливает, что данные внутри LIST будут выровнены по положению десятичной точки.
<b>COLOR</b>	Задаёт цвет фона для элемента.

Поле **LIST** размещает в REPORT текущий элемент списка элементов данных, область для которого указана AT-атрибутом поля. Поле LIST допустимо использовать только внутри DETAIL-структуры. Цель применения LIST-поля - вывести на печать данные в том же формате, в котором они появляются на экране (устанавливается атрибутом FORMAT поля LIST). Когда на печать выводится первый экземпляр структуры DETAIL, содержащей поле LIST, то вместе с текущим элементом FROM-атрибута на печать выводится и любая заголовочная информация из FORMAT. При выводе на печать последней DETAIL-структуры, содержащей поле LIST, то вместе с текущим элементом FROM-атрибута на печать выводится и любая оконечная информация из FORMAT.

### Пример:

```
Q    QUEUE
F1   STRING(1)
F2   STRING(4)
      END
```

```
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail DETAIL,AT(0,0,6500,1000)
      LIST,AT(80,0,20,20),USE(?L1),FROM(Q),FORMAT('5C~List~15L~Box~'),COLUMN
      END
      END
```

**OPTION (объявить группу RADIO-полей документа)**

```
OPTION(текст),AT()[,USE()][,BOXED][,HIDE] [,TRN] [,COLOR( )]
      радио-кнопки
END
```

<b>OPTION</b> <i>текст</i>	Выводит на печать группу RADIO-полей. Строчковая константа с поясняющим текстом для группы полей. Данный текст будет выведен на печать только тогда, когда для GROUP указан атрибут BOXED.
<b>AT</b>	Определяет размер и местоположение поля. Если не указан, то значения по умолчанию устанавливаются библиотекой времени исполнения.
<b>USE</b>	Метка строковой переменной, в которой содержится значение выбранной пользователем RADIO-кнопки.
<b>BOXED</b>	Устанавливает, что группа RADIO-полей будет очерчена одинарной рамкой, в верхнем элементе которой расположен текст.
<b>HIDE</b>	Указывает, что поле не будет выведено на печать, пока не выполнится оператор UNHIDE.
<b>TRN</b>	Задаёт, что элемент имеет как бы прозрачный фон и печатается поверх предыдущего изображения.
<b>COLOR</b> <i>радио-кнопки</i>	Задаёт цвет фона для элемента. Объявления RADIO-полей.

Поле **OPTION** выводит на печать группу RADIO полей, которые изображают на экране перечень выбираемых компонент. Если в структуре OPTION присутствует несколько RADIO-полей, то тем самым определяются несколько выбираемых компонент.

Допустимой считается ситуация, когда ни одна из RADIO-кнопок не выбрана. Это возможно только при условии, когда значение USE-переменной структуры OPTION не совпадает со значением параметра текст поля RADIO. Сделанный выбор обозначается закрашиванием соответствующей кнопки RADIO.

**Пример:**

```
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail DETAIL,AT(0,0,6500,1000)
  OPTION('Option'),USE(OptVar),AT(80,0,20,20),BOXED
  RADIO('Radio 1'),AT(80,0,20,20),USE(?R1)
  RADIO('Radio 2'),AT(100,0,20,20),USE(?R2)
END
END
END
```



**RADIO (объявить поле radio-кнопки в документе)**

**RADIO**(*текст*),AT()[,FONT()[,| LEFT | ][,USE()[,HIDE]  
| RIGHT | ][,TRN] [,COLOR( )]

**RADIO**

Размещает в REPORT radio-кнопку.

*текст*

Строковая константа, текст которой изображается на экране в поле radio-кнопки.

**AT**

Определяет размер и местоположение поля. Если не указан, то значения по умолчанию устанавливаются библиотекой времени исполнения.

**FONT**

Определяет для поля шрифт вывода на экран.

**LEFT**

Указывает, что текст в поле будет сдвинут влево.

**RIGHT**

Указывает, что текст в поле будет сдвинут вправо (положение по умолчанию).

**USE**

Определяет для поля метку соответствия.

**HIDE**

Указывает, что поле не будет выведено на печать, пока не выполнится оператор UNHIDE.

**TRN**

Задаст, что элемент имеет как бы прозрачный фон и печатается поверх предыдущего изображения.

**COLOR**

Задаст цвет фона для элемента.

Поле **RADIO** размещает в REPORT radio-кнопку, размер и местоположение для которой указывается AT-атрибутом поля. Поле RADIO размещается только внутри OPTION-поля. Когда пользователь осуществил RADIO-выбор (значение USE-переменной поля OPTION), то на экране закрашивается соответствующая кнопка RADIO.

**Пример:**

```
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail DETAIL,AT(0,0,6500,1000)
  OPTION('Option'),USE(OptVar),AT(80,0,20,20),BOXED
  RADIO('Radio 1'),AT(80,0,20,20),USE(?R1)
  RADIO('Radio 2'),AT(100,0,20,20),USE(?R2)
  RADIO('Radio 3'),AT(100,0,20,20),USE(?R2),LEFT
  ...
```

**STRING (объявить в документе поле строки)**

```

STRING(текст),AT()[,FONT()][,HIDE][,TRN][,USE()][,COLOR()][,ANGLE()][,SKIP]
[, LEFT ][, PAGENO ]
[, RIGHT ][, CNT [,RESET() / PAGE [, TALLY( ) ] ]
[, CENTER ][, SUM [,RESET() / PAGE [, TALLY( ) ] ]
[, DECIMAL ][, AVE [,RESET() / PAGE [, TALLY( ) ] ]
[, MIN [,RESET() / PAGE [, TALLY( ) ] ]
[, MAX [,RESET() / PAGE [, TALLY( ) ] ]

```

**STRING**

Помещает текст в REPORT

*текст*

Строковая константа, содержащая либо текст для вывода на экран, либо шаблон вывода на экран для форматирования переменной, указанной в USE-атрибуте.

**AT**

Определяет размер и местоположение поля. Если не указан, то значения по умолчанию устанавливаются библиотекой времени исполнения.

**FONT**

Определяет для поля шрифт вывода на экран.

**HIDE**

Указывает, что поле не будет выведено на печать, пока не выполнится оператор UNHIDE.

**TRN**

Указывает, что при выводе текст или символы USE-переменной будут “прозрачно” нанесены поверх фона.

**USE**

Указывает переменную, значения которой при выводе на печать будут отформатированы согласно шаблону, объявленному параметром текст (если вместо текстовой строки указан шаблон).

**COLOR**

Задаёт цвет фона для элемента.

**ANGLE**

Задаёт печать элемента под заданным углом, отсчитываемым против движения часовой стрелки от горизонтального направления отчета

**SKIP**

Указывает, что данный элемент не печатается, если он содержит пробелы, а все остальные элементы смещаются вверх, чтобы не оставлять “пустого” места в бланке.

**LEFT**

Устанавливает, что текст внутри области, определенной атрибутом AT, будет выровнен слева.

**RIGHT**

Устанавливает, что текст внутри области, определенной атрибутом AT, будет выровнен справа.

**CENTER**

Устанавливает, что текст внутри области, определенной атрибутом AT, будет отцентрирован.

**DECIMAL**

Устанавливает, что текст внутри области, определенной атрибутом AT, будет выровнен по положению десятичной точки.

**PAGENO**

Устанавливает, что номер текущей страницы будет выведен в формате шаблона, объявленного параметром текст (если вместо текстовой строки указан шаблон).

**CNT**

Устанавливает, что счетчик напечатанных DETAIL-структур будет выведен в формате шаблона, объявленного параметром текст (если вместо

	текстовой строки указан шаблон).
<b>SUM</b>	Устанавливает, что “сумматор” USE-переменной будет выведен в формате шаблона, объявленного параметром текст (если вместо текстовой строки указан шаблон).
<b>AVE</b>	Устанавливает, что “среднее” USE-переменной будет выведено в формате шаблона, объявленного параметром текст (если вместо текстовой строки указан шаблон).
<b>MIN</b>	Устанавливает, что минимальное значение USE-переменной будет выведено в формате шаблона, объявленного параметром текст (если вместо текстовой строки указан шаблон).
<b>MAX</b>	Устанавливает, что максимальное значение USE-переменной будет выведено в формате шаблона, объявленного параметром текст (если вместо текстовой строки указан шаблон).
<b>RESET</b>	Устанавливает, что значение CNT, SUM, AVE, MIN или MAX будет обнулено по завершению указанного раздела документа.
<b>PAGE</b>	Устанавливает, что значение CNT, SUM, AVE, MIN или MAX будет обнулено при переходе на новую страницу.
<b>TALLY</b>	Задается при вычислении полей CNT, SUM, AVE, MIN или MAX.

Поле **STRING** размещает в REPORT текст, размер и местоположение для которого указывается AT-атрибутом поля. Если параметр текст представляет собой шаблон, а не текстовую константу или переменную, то значения переменной, указанной атрибутом USE, выводятся в формате шаблона в область, установленную AT-атрибутом.

STRING с атрибутом TRN “прозрачно” выводит символы, не уничтожая фона, т. е. выводятся только те точки, которые изображают символ. Это позволяет “наносить” STRING-строку непосредственно на рисунок поля IMAGE, без разрушения самого рисунка.

### Пример:

```

CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
    BREAK(Pre:Key1)
    HEADER,AT(0,0,6500,1000)
        STRING('Group Head'),AT(3000,500,1500,500),FONT('Arial',18)
    END
Detail    DETAIL,AT(0,0,6500,1000)
    STRING(@N$11.2'),AT(6000,1500,500,500),USE(Pre:F1)
    END
    FOOTER,AT(0,0,6500,1000)
        STRING('Group Total:'),AT(5500,500,1500,500)
        STRING(@N$11.2'),AT(6000,500,500,500),USE(Pre:F1),SUM,RESET(Pre:Key1)
    END
END
END
END

```

**TEXT (объявить многостроковое текстовое поле)**

TEXT,AT()[,USE()],[,FONT()][	CAP   ],	LEFT   ][,HIDE]
UPR		RIGHT
[,TRN][,COLOR()],[,SKIP]		CENTER  [,RESIZE]

<b>TEXT</b>	Размещает в REPORT много-строковое поле для вывода на печать.
<b>AT</b>	Определяет размер и местоположение поля. Если не указан, то значения по умолчанию устанавливаются библиотекой времени исполнения.
<b>USE</b>	Метка переменной, значение которой выводится на печать.
<b>FONT</b>	Определяет для поля шрифт вывода на экран.
<b>UPR / CAP</b>	Устанавливает заглавными либо все буквы текста, либо начальные буквы каждого слова текста.
<b>LEFT</b>	Устанавливает, что текст внутри области, определенной атрибутом AT, будет выровнен слева.
<b>RIGHT</b>	Устанавливает, что текст внутри области, определенной атрибутом AT, будет выровнен справа.
<b>CENTER</b>	Устанавливает, что текст внутри области, определенной атрибутом AT, будет отцентрирован.
<b>HIDE</b>	Указывает, что поле не будет выведено на печать, пока не выполнится оператор UNHIDE.
<b>TRN</b>	Задаст, что элемент имеет как бы прозрачный фон и печатается поверх предыдущего изображения.
<b>COLOR</b>	Задаст цвет фона для элемента.
<b>SKIP</b>	Указывает что данный элемент не печатается, если он содержит пробелы, а все остальные элементы смещаются вверх, чтобы не оставлять “пустого” места в бланке.
<b>RESIZE</b>	Задаст изменение высоты поля в соответствии с его реальным содержанием.

Поле **TEXT** размещает в REPORT много-строковое поле для вывода на печать, размер и местоположение для которого указывается AT-атрибутом поля. Выводимые на печать данные содержатся в переменной, которую определяет USE-атрибут.

**Пример:**

```

CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
Detail  DETAIL,AT(0,0,6500,1000)
        TEXT,AT(0,0,40,40),USE(E1)
        TEXT,AT(100,0,40,40),USE(E6),FONT('Arial',12)
        TEXT,AT(120,0,40,40),USE(E7),CAP
        TEXT,AT(140,0,40,40),USE(E8),UPR
        TEXT,AT(160,0,40,40),USE(E9),LEFT

```

```
TEXT,AT(180,0,40,40),USE(E10),RIGHT  
TEXT,AT(200,0,40,40),USE(E11),CENTER  
END  
END
```

## Атрибуты элементов в отчетах

### ANGLE (установить угол печати элемента)

**ANGLE**( *величина* )

**ANGLE**                                      Определить ориентацию элемента типа STRING.  
*величина*                                      Целочисленная константа или выражение, которая задает величину поворота в десятых долях градуса. Если величина положительная, то угол отсчитывается против часовой стрелки от горизонтали отчета. Допустимые значения величины от 3600 до -3600.

Атрибут **ANGLE** задает печать элемента типа STRING под заданным углом, измеряемым в направлении против движения часовой стрелки от горизонтального направления отчета (и книжной ориентации, и альбомной). Он дает возможность печатать элемент в произвольном направлении, помимо горизонтального. Шрифт для такого элемента должен использоваться TrueType.

#### Пример:

```
CustRpt  REPORT,AT(1000,1000,6500,9000),THOUS,FONT('Arial',10)
Detail    DETAIL,AT(0,0,6500,1000)
          STRING('String Constant'),AT(500,500,1500,500)
          !Печатать горизонтальный текст
          STRING('String Constant'),AT(500,500,1500,500),ANGLE(900)
          ! Печатать вертикальный текст
          STRING('String Constant'),AT(500,500,1500,500),ANGLE(1800)
          ! Печатать текст "вверх ногами"
          END
          END
```

### AT (установить местоположение и размер поля в документе)

**AT**(*[x][,y][,ширина][,высота]*)

**AT**                                      Определяет местоположение и размер поля.  
*x*    Целочисленная константа, константное выражение для указания исходного горизонтального положения левого верхнего угла поля по отношению к положению левого верхнего угла структуры, содержащей данное поле. Если параметр не указан, то значение по умолчанию устанавливаются библиотекой времени исполнения.  
*y*    Целочисленная константа, константное выражение для указания исходного вертикального положения левого верхнего угла поля по отношению к положению левого верхнего угла структуры, содержащей данное поле. Если

	параметр не указан, то значение по умолчанию устанавливаются библиотекой времени исполнения.
<i>ширина</i>	Целочисленная константа, константное выражение для указания ширины поля. Если параметр не указан, то значение по умолчанию устанавливаются библиотекой времени исполнения.
<i>высота</i>	Целочисленная константа, константное выражение для указания высоты поля. Если параметр не указан, то значение по умолчанию устанавливаются библиотекой времени исполнения.

Атрибут **AT** указывает размер поля и его положение относительно левого верхнего угла структуры, содержащей данное поле. Если какой-либо параметр не указан, то значение по умолчанию устанавливаются библиотекой времени исполнения.

Если не присутствует какой-либо из атрибутов THOUS, MM или POINTS, то - по умолчанию - значения параметров x, y, ширина и высота заданы в условных единицах измерения. За условные единицы принимаются одна четверть усредненной ширины символа и одна восьмая его усредненной высоты. Ясно, что величина условной единицы зависит от установленного для документа шрифта по умолчанию. За основу измерений берется либо шрифт, указанный FONT-атрибутом документа, либо установленный для принтера системный шрифт по умолчанию.

### Пример:

```
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
                                     !AT определяет область для печати тела
                                     ! документа
Detail  DETAIL,AT(0,0,6500,1000)    !At определяет размер DETAIL-области и
                                     ! и ее смещение относительно
                                     ! последней напечатанной DETAIL
      STRING('String Constant'),AT(500,500,1500,500)
                                     !AT определяет размер поля и
                                     ! его смещение внутри DETAIL-области
END
END
```

## AVE (установить итоговое среднее)

AVE( [ переменная ] )

<b>AVE</b>	Вычисляет среднее значение (арифметическое среднее) USE-переменных печатаемых элементов типа STRING.
<i>переменная</i>	Имя числовой переменной, куда заносится промежуточное значение вычисляемого среднего. Позволяет создавать обобщенные итоги по таким полям.

Атрибут **AVE** устанавливает вывод на печать среднего (арифметического) значения USE-переменной поля **STRING**. Если атрибут **TALLY** не указан, то подсчет ведется следующим образом:

- \* Если поле с атрибутом **AVE** входит в структуру **DETAIL**, то среднее вычисляется всякий раз, когда **DETAIL** выводится на печать оператором **PRINT**.

- \* Если поле с атрибутом **AVE** входит в **FOOTER** раздела документа (**FOOTER** внутри **BREAK**), то среднее вычисляется всякий раз, когда любая **DETAIL**, входящая в **BREAK**, выводится на печать оператором **PRINT**.

- \* Если поле с атрибутом **AVE** входит в **FOOTER** страницы документа (**FOOTER** вне **BREAK**), то среднее вычисляется всякий раз, когда любая **DETAIL**, входящая в любую **BREAK**, выводится на печать оператором **PRINT**.

- \* Бессмысленно помещать поле с атрибутом **AVE** в структуру **HEADER**, поскольку в тот момент, когда **HEADER** выводится на печать, ни одной **DETAIL** не будет напечатано.

Значение среднего обнуляется только тогда, когда вместе с **AVE** для поля **STRING** указан один из атрибутов **RESET** или **PAGE**. Как правило, такое **STRING**-поле располагают в **FOOTER** для раздела или для страницы.

### Пример:

```
CustRpt  REPORT,AT(1000,1000,6500,9000),THOUS
Break1   BREAK(LocalVar),USE(?BreakOne)
Break2   BREAK(Pre:Key1),USE(?BreakTwo)
Detail   DETAIL,AT(0,0,6500,1000),USE(?DetailOne)
          STRING(@N$11.2'),AT(6000,1500,500,500),USE(Pre:F1)
          END
          FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
          STRING('Group Average:'),AT(5500,500)
          STRING(@N$11.2'),AT(6000,500),USE(Pre:F1),AVE(LocalVar),RESET(Break2)
          END
          END
          FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
          STRING('Grand Average:'),AT(5500,500)
          STRING(@N$11.2'),AT(6000,500),USE(LocalVar),AVE,TALLY(?BreakTwo)
          END
          END
          END
```



**BOXED (установить рамку для группы полей документа)****BOXED**

Атрибут **BOXED** устанавливает, что поля структуры GROUP или OPTION будут заключены в рамку, представленную одинарной линией. Промежуток, выделенный в верхнем элементе рамки, будет заполнен значением параметра текст поля GROUP или OPTION. Если BOXED не указан, то параметр текст поля GROUP или OPTION на печать не выводится.

**CAP,UPR (установить регистр вывода символа на печать)****CAP  
UPR**

Атрибуты **CAP** и **UPR** устанавливают автоматический выбор регистра при выводе на печать текста поля TEXT. UPR устанавливает верхний регистр для всех символов; CAP устанавливает верхний регистр только для первого символа каждого слова, остальные символы выводятся в нижнем регистре.

**CNT (установить подсчет итога)****CNT( [ переменная ] )****CNT***переменная*

Подсчитывает, сколько раз напечатана структура DETAIL.  
Имя числовой переменной, куда заносится промежуточное значение подсчитываемой величины. Позволяет создавать обобщенные итоги по таким полям.

Атрибут **CNT** устанавливает автоматический подсчет напечатанных структур DETAIL. Если атрибут TALLY не указан, то подсчет ведется следующим образом:

\* Если поле с атрибутом CNT входит в структуру DETAIL, то значение счетчика напечатанных структур увеличивается на единицу всякий раз, когда DETAIL выводится на печать оператором PRINT. Этим обеспечивается “текущий” счет./

\* Если поле с атрибутом CNT входит в FOOTER раздела документа (FOOTER внутри BREAK), то значение счетчика напечатанных структур увеличивается на единицу всякий раз, когда любая DETAIL, входящая в BREAK, выводится на печать оператором PRINT. Этим достигается подсчет общего числа напечатанных в разделе документа структур DETAIL.

\* Если поле с атрибутом CNT входит в FOOTER страницы документа (FOOTER вне BREAK), то значение счетчика напечатанных структур увеличивается на единицу всякий раз, когда любая DETAIL, входящая в любую BREAK, выводится на печать оператором PRINT. Этим достигается подсчет общего числа напечатанных на странице (или

в документе) структур DETAIL.

\* Бессмысленно помещать поле с атрибутом CNT в структуру HEADER, поскольку в тот момент, когда HEADER выводится на печать, ни одной DETAIL не будет напечатано. CNT обнуляется только тогда, когда вместе с CNT указан один из атрибутов RESET или PAGE.

### Пример:

```
CustRpt  REPORT,AT(1000,1000,6500,9000),THOUS
Break1   BREAK(LocalVar),USE(?BreakOne)
Break2   BREAK(Pre:Key1),USE(?BreakTwo)
Detail   DETAIL,AT(0,0,6500,1000),USE(?DetailOne)
          STRING(@N$11.2'),AT(6000,1500,500,500),USE(Pre:F1)
          END
          FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
          STRING('Group Count:'),AT(5500,500)
          STRING(@N$11.2'),AT(6000,500),USE(Pre:F1),CNT(LocalVar),RESET(Break2)
          END
          END
          FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
          STRING('Grand Count:'),AT(5500,500)
          STRING(@N$11.2'),AT(6000,500),USE(LocalVar),CNT,TALLY(?BreakTwo)
          END
          END
          END
```

## COLOR (установить цвет печати)

### COLOR(*кзс*)

#### COLOR

*кзс*

Устанавливает при выводе на печать цвет элемента.

Целочисленная константа типа LONG или ULONG, содержащая либо красную, зеленую и синюю компоненты цвета, которые указываются в трех младших байтах (байты 0,1 и 2), либо символическое имя для значения цвета стандарта Windows.

Атрибут **COLOR** устанавливает цвет элемента LINE при выводе на печать. Для BOX или ELLIPSE указанный цвет определяет цвет рамки. Для других элементов он устанавливает цвет фона. Символические имена для значений цвета стандарта Windows находятся в файле EQUATES.CLW.

### Пример:

```
CustRpt  REPORT,AT(1000,1000,6500,9000),THOUS
```

```

CustDetail DETAIL,AT(0,0,6500,1000)
    ELLIPSE,AT(60,60,200,200),COLOR(COLOR:ACTIVEBORDER)
        !Символическое имя цвета
    BOX,AT(360,60,200,200),COLOR(00FF0000h)        !Чисто красный
    ..

```

## FILL (установить при выводе на печать цвет заполнения)

### FILL(*кзс*)

**FILL** Устанавливает при выводе на печать цвет “раскраски” для полей BOX или ELLIPSE.

*кзс* Целочисленная константа типа LONG или ULONG, содержащая либо красную, зеленую и синюю компоненты цвета, которые указываются в трех младших байтах (байты 0,1 и 2), либо символическое имя для значения цвета стандарта Windows.

Атрибут **FILL** устанавливает при выводе на печать цвет “раскраски” для полей BOX или ELLIPSE. Если атрибут не указан, то “раскраски” поля не происходит.

### Пример:

```

CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail DETAIL,AT(0,0,6500,1000)
    ELLIPSE,AT(60,60,200,200),FILL(COLOR:ACTIVEBORDER)
        !Символическое имя цвета
    BOX,AT(360,60,200,200),FILL(00FF0000h)    !Чисто красный
END
END

```

## FONT (установить шрифт по умолчанию)

### FONT(*[начертание][,размер][,цвет][,стиль]*)

**FONT** Устанавливает при выводе на печать шрифт по умолчанию для поля.

*начертание* Строковая константа, в которой указано имя шрифта. Если параметр опущен, то используется шрифт FONT-атрибута (если он присутствует) выводимой на печать структуры; если и он опущен, то используется шрифт FONT-атрибута (если он присутствует) структуры REPORT; иначе - используется шрифт по умолчанию для принтера.

*размер* Целочисленная константа, указывающая размер шрифта (в пунктах). Если параметр не указан, то принимается размер шрифта по умолчанию для принтера.

*цвет* Целочисленная константа типа LONG, в трех младших байтах

которой указываются значения красной, зеленой и синей компонент цвета шрифта, либо символическое имя значения цвета стандарта Windows. Если параметр не указан, то устанавливается черный цвет.

*стиль*

Целочисленная константа, константное выражение, символическое имя, определяющие толщину и стиль шрифта. Если параметр опущен, то принимается нормальная толщина шрифта.

Атрибут **FONT** устанавливает при выводе на печать шрифт для поля, отменяя FONT-установки на уровне выводимой на печать структуры или REPORT.

Параметром начертание можно задать имя любого зарегистрированного в Windows шрифта, который поддерживается драйвером принтера. К ним относятся TrueType-шрифты, поддерживаемые большинством принтеров. В файле EQUATES.CLW находятся символические имена значений для стандартных стилей. Значение параметра стиль в диапазоне от 0 до 1000 определяет толщину шрифта. К этому значению можно прибавлять значения для наклона, подчеркивания и перечеркивания текста. Указанные символические имена находятся в файле EQUATES.CLW:

```
FONT:thin    EQUATE (700)
FONT:italic   EQUATE (01000H)
FONT:underline EQUATE (02000H)
FONT:strikeout EQUATE (04000H)
```

### Пример:

```
CustRpt  REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail DETAIL,AT(0,0,6500,1000)
    STRING('Text'),AT(0,0),FONT('Arial',14,00FF0000h)
    STRING('Text'),AT(160,160),FONT('Arial',12,,FONT:italic)
END
END
```

## FORMAT (установить формат печати поля LIST)

**FORMAT**(строка формата)

**FORMAT** Устанавливает формат вывода данных на печать.  
строка формата Строковая константа, описывающая формат вывода в одну или в несколько колонок.

Атрибут **FORMAT** устанавливает формат вывода на печать данных поля LIST. Строка формата содержит информацию о форматировании вывода данных в одну или в несколько колонок.

В строку формата входят “поля-описатели”, которые соотносятся с полями QUEUE-структуры языка Clarion. Если “полей-описателей” - несколько, то их можно объединить в группу - “поле-группы”, которая заключается в квадратные скобки ([ ]). Квадратные скобки здесь принадлежат единой синтаксической конструкции и не являются метасимволами описания синтаксиса.

На печать выводятся только те поля из QUEUE, для которых существуют “поля-описатели”. Например, если в строке формата указано два поля, а в QUEUE содержится три, то только два указанных в строке формата поля будут напечатаны в поле LIST.

Формат “поля-описателя”: ширина выравнивание [ (отступ) ] [ модификаторы ]

**ширина** Обязательное целое, которое определяет ширину поля. Если не переустановлено одним из атрибутов THOUS, MM или POINTS, то используются условные единицы измерения.

**выравнивание** Одиночная заглавная буква (L, R, C или D), означающая выравнивание слева (Left), справа (Right), по центру (Center) или по десятичной точке (Decimal). Обязательно наличие одной из указанных букв.

**отступ** Необязательное целое, будучи заключено в круглые скобки, указывает отступ от границы выравнивания. Может принимать отрицательные значения. При выравнивании слева (L) отступ определяет отступ слева, при выравнивании справа (R) или по десятичной точке (D) он определяет отступ справа; при выравнивании по центру (C) им определяется отступ от центра поля.

**модификаторы:** Необязательные специальные символы (представлены ниже) модифицируют для поля или группы формат вывода на печать. Для одного поля или одной группы можно указать несколько модификаторов.

**~заголовок~[выравнивание] [ (отступ) ] ]** Заключенная в разделители тильда (~) строка заголовка, за которой следуют необязательные выравнивание и/или отступ, печатает заголовок в начале списка. Если явно не переустановлены, то выравнивание и отступ для поля используются и для заголовка.

**@шаблон@** Форматирует вывод поля на печать. Концевой символ @ необходим для обозначения конца шаблона, поскольку, например, использование шаблона @N12~Kг~ в строке формата не приводит к неоднозначности.

**#номер#** Заключенный в разделители “решетка” (#) номер определяет какое QUEUE-поле должно быть выведено на печать. Если в последующих полях строки формата #номер# явно не указан, то последующие QUEUE-поля выбираются в порядке их следования за #номер# полем. Например, #2# в первом поле строки формата означает, что вывод начнется со второго QUEUE-поля, минуя первое. Если число полей строки формата >= числа полей в QUEUE, то форматирование “зациклится” на начало QUEUE.

**—** Символ подчеркивания означает, что выведенное поле будет

подчеркнуто.

/                      Слэш (косая черта) указывает, что следующее поле появится на новой строке (используется только для полей в группе).

|                      Вертикальная черта помещает символ вертикальной черты справа от поля.

Формат “поля-группы”: [ поля-описатели ] [ (размер) ] [ модификаторы ]

поля-описатели      Заключенный в квадратные скобки список полей-описателей. Квадратные скобки здесь принадлежат единой синтаксической конструкции и не являются метасимволами описания синтаксиса.

размер                Необязательное целое, будучи заключено в круглые скобки, определяет стандартную ширину группы. Если отсутствует, то вычисление ширины производится с учетом входящих в группу полей.

модификаторы        Модификаторы “поля-группы” воздействуют сразу на все поля группы. Описание модификаторов приведено выше.

### Пример:

```
TD    QUEUE,AUTO
FName  STRING(20)
LName  STRING(20)
Init   STRING(4)
Wage   REAL
      END
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail DETAIL,AT(0,0,6500,1000)
      LIST,AT(0,34,366,146),FORMAT(""),FROM(TD),USE(?Show)
      END
      END
CODE
OPEN(CustRpt)
SETTARGET(CustRpt)
IF SomeCondition
    ?Show{PROP:format} = '80C~First Name~80C~Last Name~16L~Intls~60R~Wage~|'
ELSE
    ?Show{PROP:format} = '80C~First Name~80C~Last Name~16L~Intls~60D(10)~Wage~|'
END
```

**FROM (установить источник данных поля списка)****FROM**(*источник*)**FROM***источник*

Указывает источник данных, выводимых на печать в поле LIST.

Метка QUEUE или переменная (обычно GROUP) где находятся элементы данных, предназначенные для вывода на печать в поле LIST.

Атрибут **FROM** указывает источник данных, выводимых на печать в поле LIST. Форматирование отображения элементов данных осуществляется согласно информации, содержащейся в атрибуте FORMAT.

Если в качестве источника указана метка структуры QUEUE, то на печать выводятся все поля QUEUE. Если в качестве источника указана метка поля структуры QUEUE, то на печать выводится только это поле. В поле LIST печатается буфер данных структуры QUEUE, в котором может находиться только текущий элемент структуры.

Если в качестве источника указана строковая константа или переменная, то в LIST печатается вся строка.

**Пример:**

```
TD    QUEUE,AUTO
FName  STRING(20)
LName  STRING(20)
Init   STRING(4)
Wage   REAL
      END
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
CustDetail DETAIL,AT(0,0,6500,1000)
      LIST,AT(0,34,366,146),FORMAT('80L80L16L60L'),FROM(TD),USE(?Show1)
      LIST,AT(0,200,100,146),FORMAT('80L'),FROM(Fname),USE(?Show2)
      ..
```

**HIDE (запретить вывод поля на печать)****HIDE**

Атрибут **HIDE** устанавливает, что поле не будет выведено на печать пока не выполнится оператор UNHIDE, снимающий с поля запрет вывода.

**LEFT, RIGHT, CENTER, DECIMAL (установить выравнивание )**

```
LEFT([отступ])  
RIGHT([отступ])  
CENTER([отступ])  
DECIMAL([отступ])
```

*отступ* Целочисленная константа, определяющая отступ от границы выравнивания. Если не переустановлена одним из атрибутов THOUS, MM или POINTS, то за единицу измерения принимается условная единица.

Атрибуты **LEFT**, **RIGHT**, **CENTER** и **DECIMAL** устанавливают выравнивание выводимых на печать данных. LEFT определяет выравнивание слева, RIGHT - справа, CENTER - центрирование текста и DECIMAL устанавливает, что числа будут выравниваться относительно десятичной точки.

Параметр отступ атрибута CENTER определяет смещение текста относительно центра. Для атрибута DECIMAL отступ определяет местоположение десятичной точки.

Перечисленные ниже поля допускают только атрибуты LEFT или RIGHT (параметр отступ - отсутствует):

```
CHECK  
GROUP  
OPTION  
RADIO
```

Поля

```
LIST  
STRING
```

могут использовать LEFT(отступ), RIGHT(отступ), CENTER(отступ) или DECIMAL(отступ).

LEFT, RIGHT и CENTER атрибуты (без параметра отступ) могут использоваться полем TEXT.

**Пример:**

```
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS  
CustDetail DETAIL,AT(0,0,6500,1000)  
    LIST,AT(0,20,100,146),FORMAT('800L'),FROM(Fname),USE(?Show2),LEFT(100)  
END  
END
```



## LINEWIDTH (установить толщину линии элемента)

**LINEWIDTH**( *толщина* )

**LINEWIDTH**            Задает толщину линии для элемента LINE или толщину линии контура для BOX и ELLIPSE.

*толщина*                Положительная константа указывает толщину в пикселах.

Атрибут **LINEWIDTH** задает толщину линии для элемента LINE или толщину линии контура для BOX и ELLIPSE.

### Пример:

```
CustRpt   REPORT,AT(1000,1000,6500,9000),THOUS
Detail     DETAIL,AT(0,0,6500,1000),USE(?DetailOne)
           LINE,AT(105,78,-49,0),USE(?Line1),LINEWIDTH(3)      !Линия в 3 пиксела
толщиной
           BOX,AT(182,27,50,50),USE(?Box1),LINEWIDTH(3)        !Прямоугольник с
контуром в 3 пиксела
           STRING(@N$11.2'),AT(6000,1500,500,500),USE(Pre:F1)
           END
           END
```

## MAX (установить итоговый максимум)

**MAX**( [ *переменная* ] )

**MAX**                    Находит максимальное среди значений USE-переменных элементов типа STRING.

*переменная*            Имя числовой переменной, куда заносится промежуточное значение искомой величины. Позволяет создавать обобщенные итоги по таким полям.

Атрибут **MAX** инициирует вывод на печать максимального значения, которого к данному моменту достигла USE-переменная поля STRING. Если атрибут TALLY не указан, то подсчет ведется следующим образом:

- \* MAX-поле структуры DETAIL оценивается всякий раз, когда DETAIL выводится на печать оператором PRINT. Этим обеспечивается “текущее” максимальное значение.

- \* MAX-поле структуры FOOTER раздела документа (FOOTER внутри BREAK) оценивается всякий раз, когда оператором PRINT выводится на печать любая DETAIL из структуры BREAK, в которую (BREAK) входит STRING-поле. Этим обеспечивается получение максимального значения переменной в разделе документа.

- \* MAX-поле структуры FOOTER страницы документа оценивается всякий раз, когда оператором

PRINT выводится на печать любая DETAIL из любой BREAK. Этим

обеспечивается получение максимального значения переменной в странице документа.

\* Бессмысленно помещать MAX-поле в структуру HEADER, поскольку в тот момент, когда HEADER выводится на печать, ни одной DETAIL еще не напечатано.

Значение **MAX** обнуляется только тогда, когда вместе с MAX указан один из атрибутов RESET или PAGE.

**Пример:**

```
CustRpt    REPORT,AT(1000,1000,6500,9000),THOUS
Break1     BREAK(LocalVar),USE(?BreakOne)
Break2     BREAK(Pre:Key1),USE(?BreakTwo)
Detail     DETAIL,AT(0,0,6500,1000),USE(?DetailOne)
           STRING(@N$11.2'),AT(6000,1500,500,500),USE(Pre:F1)
           END
           FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
           STRING('Group Maximum:'),AT(5500,500)
           STRING(@N$11.2'),AT(6000,500),USE(Pre:F1),MAX(LocalVar),RESET(Break2)
           END
           END
           FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
           STRING('Grand Maximum:'),AT(5500,500)
           STRING(@N$11.2'),AT(6000,500),USE(LocalVar),MAX,TALLY(?BreakTwo)
           END
           END
           END
```

**META (печатать .VBX как .WMF)**

**META**

Атрибут **META** определяет, что поле .VBX будет выводиться на печать как .WMF - Windows метафайл, а не как битовый массив. Обычно это повышает качество печати, снижает затраты памяти и ускоряет печать, однако печать в метафайл поддерживается не всеми объектами .VBX . Если нет уверенности, что данный .VBX поддерживает печать в метафайл, попробуйте, не получится - уберите этот атрибут.

**MIN (установить итоговый минимум)**

**MIN( [ переменная ] )**

**MIN** Находит максимальное к данному моменту среди значений USE-переменных элементов типа STRING.

*переменная* Имя числовой переменной, куда заносится промежуточное значение искомой величины. Позволяет создавать обобщенные итоги по таким полям.

Атрибут MIN инициирует вывод на печать минимального значения, которого к данному моменту достигла USE-переменная поля STRING. Если атрибут TALLY не указан, то подсчет ведется следующим образом:

- \* MIN-поле структуры DETAIL оценивается всякий раз, когда DETAIL выводится на печать оператором PRINT. Этим обеспечивается “текущее” минимальное значение.
- \* MIN-поле структуры FOOTER раздела документа (FOOTER внутри BREAK) оценивается всякий раз, когда оператором PRINT выводится на печать любая DETAIL из структуры BREAK, в которую (BREAK) входит STRING-поле. Этим обеспечивается получение минимального значения переменной в разделе документа.
- \* MIN-поле структуры FOOTER страницы документа оценивается всякий раз, когда оператором PRINT выводится на печать любая DETAIL из любой BREAK. Этим обеспечивается получение минимального значения переменной в странице документа.
- \* Бессмысленно помещать MIN-поле в структуру HEADER, поскольку в тот момент, когда HEADER выводится на печать, ни одной DETAIL еще не напечатано.

Значение MIN обнуляется только тогда, когда вместе с MIN указан один из атрибутов RESET или PAGE.

### Пример:

```
CustRpt  REPORT,AT(1000,1000,6500,9000),THOUS
Break1   BREAK(LocalVar),USE(?BreakOne)
Break2   BREAK(Pre:Key1),USE(?BreakTwo)
Detail   DETAIL,AT(0,0,6500,1000),USE(?DetailOne)
          STRING(@N$11.2'),AT(6000,1500,500,500),USE(Pre:F1)
          END
          FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
          STRING('Group Minimum:'),AT(5500,500)
          STRING(@N$11.2'),AT(6000,500),USE(Pre:F1),MIN(LocalVar),RESET(Break2)
          END
          END
          FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
          STRING('Grand Minimum:'),AT(5500,500)
          STRING(@N$11.2'),AT(6000,500),USE(LocalVar),MIN,TALLY(?BreakTwo)
          END
          END
          END
```

### PAGE (установить сброс страничных итогов)

#### PAGE

Атрибут PAGE определяет, что при переходе на новую страницу производится сброс (обнуление) итогового значения CNT, SUM, AVE, MIN или MAX.

**PAGENO (установить печать номера страницы)****PAGENO**

Атрибут **PAGENO** определяет, что STRING-поле будет выводить на печать номер страницы.

**RESET (установить сброс итогов)****RESET(подраздел)**

**RESET** Сбрасывает в 0 значение CNT, SUM, AVE, MIN или MAX.  
*подраздел* Метка BREAK-структуры.

Атрибут **RESET** определяет раздел документа, в котором производится сброс (обнуление) итогового значения CNT, SUM, AVE, MIN или MAX.

**Пример:**

```
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
    BREAK(Pre:Key1)
    HEADER,AT(0,0,6500,1000)
        STRING('Group Head'),AT(3000,500,1500,500),FONT('Arial',18)
    END
Detail    DETAIL,AT(0,0,6500,1000)
    STRING(@N$11.2'),AT(6000,1500,500,500),USE(Pre:F1)
    END
    FOOTER,AT(0,0,6500,1000)
        STRING('Group Total:'),AT(5500,500,1500,500)
        STRING(@N$11.2'),AT(6000,500,500,500),USE(Pre:F1),SUM,RESET(Pre:Key1)
    END
END
END
```

**RESIZE (переменная высота элемента TEXT)****RESIZE**

Атрибут **RESIZE** указывает, что высота поля TEXT изменяется в соответствии количеством данных, которое в нем следует напечатать. Максимальная высота определяется атрибутом AT.

**Пример:**

```
CustRpt    REPORT,AT(1000,1000,6500,9000),THOUS
Detail     DETAIL,AT(0,0,6500,)
           STRING(@N$11.2'),AT(500,500,500,),USE(Pre:F1)
           TEXT,AT(500,1000,500,5000),USE(Pre:Memo1),RESIZE !Print height up to 5"
           END
           END
ROUND      (сгладить углы BOX-поля документа)
           ROUND
Атрибут ROUND специфицирует сглаживание углов управляющего BOX-поля
```

## SKIP (условная печать поля STRING или TEXT)

### SKIP

Атрибут **SKIP** указывает, что поле типа STRING или TEXT печатается только если соответствующая ему USE-переменная содержит данные. Если она данных не содержит, то это поле не печатается, а все остальные поля “подтягиваются” вверх, чтобы предотвратить появление пустого места. Наиболее полезно при печати этикеток и адресов.

#### Пример:

```
CustRpt    REPORT,AT(1000,1000,6000,9000),THOUS
Detail     DETAIL,AT(0,0,2000,1000)          !фиксированной высоты
           STRING(@s35),AT(250,250,500,),USE(Pre:Name)
           STRING(@s35),AT(250,250,500,),USE(Pre:Address1)
           STRING(@s35),AT(250,250,500,),USE(Pre:Address2),SKIP      !Не печатать если
пусто
           STRING(@s35),AT(250,250,500,),USE(CityStateZip)          ! и подтянуть это
поле вверх
           END
           END
```

## SUM (установить итоговое значение)

### SUM( [ переменная ] )

**SUM** Находит сумму значений USE-переменных элементов типа STRING.  
*переменная* Имя числовой переменной, куда заносится промежуточное значение суммы. Позволяет создавать обобщенные итоги по таким полям.

Атрибут SUM инициирует вывод на печать суммы значений, принимаемых USE-переменной поля STRING. Если атрибут TALLY не указан, то подсчет ведется следующим образом:

\* SUM-поле структуры DETAIL увеличивается всякий раз, когда DETAIL выводится на печать оператором PRINT. Этим обеспечивается “текущее” итоговое значение.

\* SUM-поле структуры FOOTER раздела документа (FOOTER внутри BREAK) увеличивается всякий раз, когда оператором PRINT выводится на печать любая DETAIL из структуры BREAK, в которую (BREAK) входит STRING-поле. Этим обеспечивается получение суммы значений переменной в разделе документа.

\* SUM-поле структуры FOOTER страницы документа увеличивается всякий раз, когда оператором PRINT выводится на печать любая DETAIL из любой BREAK. Этим обеспечивается получение суммы значений переменной в странице документа.

\* Бессмысленно помещать SUM-поле в структуру HEADER, поскольку в тот момент, когда HEADER выводится на печать, ни одной DETAIL еще не напечатано.

Значение **SUM** обнуляется только тогда, когда вместе с SUM указан один из атрибутов RESET или PAGE.

### Пример:

```
CustRpt    REPORT,AT(1000,1000,6500,9000),THOUS
Break1     BREAK(LocalVar),USE(?BreakOne)
Break2     BREAK(Pre:Key1),USE(?BreakTwo)
Detail     DETAIL,AT(0,0,6500,1000),USE(?DetailOne)
           STRING(@N$11.2'),AT(6000,1500,500,500),USE(Pre:F1)
           END
           FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
           STRING('Group Total:'),AT(5500,500)
           STRING(@N$11.2'),AT(6000,500),USE(Pre:F1),SUM(LocalVar),RESET(Break2)
           END
           END
           FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
           STRING('Grand Total:'),AT(5500,500)
           STRING(@N$11.2'),AT(6000,500),USE(LocalVar),SUM,TALLY(?BreakTwo)
           END
           END
           END
```

## TALLY (установить общее число вычислений)

**TALLY**( *точки* )

**TALLY**                      Указывает, когда производить вычисление AVE, CNT, MAX, MIN и SUM.

*точки*                      Разделенные запятыми имена структур DETAIL и/или BREAK для которых вычисляются итоговые результаты.

Атрибут **TALLY** задает, когда производить вычисление AVE, CNT, MAX, MIN и SUM. Соответствующие вычисления производятся всякий раз, когда печатается какая либо из перечисленных в списке точки структура DETAIL или для структуры BREAK происходит

изменение контролируемого значения.

### Пример:

```
CustRpt    REPORT,AT(1000,1000,6500,9000),THOUS
Break1     BREAK(LocalVar),USE(?BreakOne)
Break2     BREAK(Pre:Key1),USE(?BreakTwo)
           HEADER,AT(0,0,6500,1000),USE(?GroupHead)
           STRING('Group Head'),AT(3000,500,1500,500),FONT('Arial',18)
           END
Detail     DETAIL,AT(0,0,6500,1000),USE(?DetailOne)
           STRING(@N$11.2'),AT(6000,1500,500,500),USE(Pre:F1)
           END
           END
           FOOTER,AT(0,0,6500,1000),USE(?BreakOneGroupFoot)
           STRING('Group Total:'),AT(5500,500,1500,500)
           STRING(@N$11.2'),AT(6000,500,500,500),USE(Pre:F1),CNT,TALLY(Break2)
           END
           END
           END
           CODE
           OPENCustRpt)
           CustRpt$?Pre:F1{PROP:Tally} = ?BreakOne      !Изменить подсчет на Break1
```

## TRN (установить “прозрачность” элемента документа)

### TRN

**TRN**-атрибут элемента отчета устанавливает “прозрачность” вывода символов на печать - без нарушения фона, на который накладывается этот элемент. На печать выводятся только те точки, которые образуют сам символ. Это позволяет наносить элемент “поверх” IMAGE без нарушения фоновой картинки.

### Пример:

```
WinOne    WINDOW,AT(0,0,160,400)
           IMAGE('PIC.BMP'),USE(?I1),FULL !Графическая картинка всего окна
           STRING('String Constant'),AT(10,0,20,20),USE(?S1),TRN
           !"Прозрачная" строка на графической картинке
           END
```

**USE (задать имя для ссылки из программы)**

```
USE( |метка| [,номер] )  
    |переменная|
```

<b>USE</b>	Задаёт для объекта переменную или метку соответствия.
<i>метка</i>	Метка соответствия, которая служит для того, чтобы ссылаться на объект в исполняемых операторах.
<i>переменная</i>	Переменная, содержащая значение, которое должно печататься в области объекта
<i>номер</i>	Целочисленная константа, которая указывает номер, который компилятор ставит в соответствие мнемонической метке для данного объекта.

Атрибут **USE** задаёт переменную или метку соответствия для объекта. Атрибут **USE**, имеющий параметр *метка* просто обеспечивает способ, для указания ссылок на этот объект в исполняемых операторах. Для некоторых объектов в качестве параметров атрибута допустимы только мнемонические метки соответствия, но не переменные. Это такие объекты как **IMAGE**, **LINE**, **BOX**, **ELLIPSE**, **GROUP**, и **RADIO**. Атрибут **USE**, имеющий в качестве параметра имя переменной, указывает для объекта переменную, которая должна принимать новое значение при вводе данных оператором. Такой подход применим для **OPTION**, **TEXT**, **LIST**, **CHECK**, и **CUSTOM**. Для объекта типа **STRING** можно использовать или мнемоническую метку соответствия.

Всем объектам в структуре **REPORT** компилятором автоматически присваиваются номера. Нумерация начинается с единицы (1) и увеличивается для каждого объекта в отчете. Параметр *номер* в атрибуте **USE** позволяет задать номер, который компилятор присваивает этому объекту. Кроме того, этот номер используется как начальная точка для дальнейшей нумерации объектов, у которых в атрибуте **USE** нет параметра *номер*. Номера последующих объектов, не имеющих параметра *номер* в атрибуте **USE**, наращиваются относительно последнего присвоенного параметра *номер*.

Для двух или более объектов, имеющих одинаковую переменную в атрибуте **USE**, в одной структуре **REPORT** были бы присвоены одинаковые мнемонические метки соответствия. Поэтому, когда компилятор встречает такую ситуацию, все метки соответствия для этой **USE**-переменной отвергаются. Это делает невозможным ссылки на эти объекты в исполняемых операторах (если только вы не знаете номера объектов присвоенные компилятором или если вы не присвоили его параметром *номер* ). Можно сознательно создать такую ситуацию, для того чтобы вывести содержимое одной переменной в нескольких объектах с различным шаблоном форматирования или при подведении итогов.

**Пример:**



```
CustRpt REPORT,AT(1000,1000,6500,9000),THOUS
Detail  DETAIL,AT(0,0,6500,1000)
        STRING('Group Total:'),AT(5500,500,1500,500),USE(?Constant)
                                !Метка соответствия
        STRING(@N$11.2'),AT(6000,1500,500,500),USE(Pre:F1)      !USE- переменная
END
END
```

## Процедуры отчета

### CLOSE (заккрыть структуру REPORT)

**CLOSE**(*отчет*)

**CLOSE**                      Закрывает структуру REPORT  
*отчет*                      Метка структуры REPORT.

Оператор **CLOSE** печатает нижний колонтитул последней страницы (если только печатавшаяся последней структура не имела атрибут ALONE) и закрывает структуру REPORT. Если REPORT была описана с атрибутом PREVIEW, то удаляются все временные метафайлы.

Если в процедуре, открывшей структуру REPORT, выполняется оператор RETURN, то REPORT автоматически закрывается.

#### Пример:

CLOSE(CustRpt)                      !Заккрыть отчет

### ENDPAGE (форсировать переход на новую страницу)

**ENDPAGE**(*отчет*[, *итоги*])

**ENDPAGE**                      Форсирует переход на новую страницу.  
*отчет*                      Метка структуры REPORT.  
*итоги*                      Целочисленная константа или переменная. Если параметр опущен или равен 0, то это вызывает печать групповых итогов по групповому прерыванию (обычно используется для остановить выдачу отчета для просмотра), Если этот параметр равен 1, то по концу страницы групповые итоги не печатаются (обычно используется при непрерывной выдаче отчета).

Оператор **ENDPAGE** инициирует переход на новую страницу и сбрасывает (flush) буфер печатаемой структуры ядра программы печати. Если структура REPORT описана с атрибутом PREVIEW, то последнее исключает потерю информации при подготовке всего документа для просмотра.

#### Пример:

SomeReport PROCEDURE  
 WMFQue    QUEUE                      !Список имен .WMF файлов  
           STRING(64)

```

END
NextEntry BYTE(1)                !Счетчик элементов списка
Report    REPORT,PREVIEW(WMFQue) !Документ с атрибутом PREVIEW
DetailOne  DETAIL
          !Поля документа
...
ViewReport WINDOW('View Report'),AT(0,0,320,200),MDI,MAX,HVSCROLL
          IMAGE('',AT(0,0,320,180),USE(?ImageField)
          BUTTON('View Next Page'),AT(0,180,60,20),USE(?NextPage),DEFAULT
          BUTTON('Print Report'),AT(80,180,60,20),USE(?PrintReport)
          BUTTON('Exit Without Printing'),AT(160,180,60,20),USE(?ExitReport)
END
CODE
OPEN(Report)
SET(SomeFile)                !Данные для создания документа
LOOP
  NEXT(SomeFile)
  PRINT(DetailOne)
END
ENDPAGE(Report)              !Сброс буфера
OPEN(ViewReport)              !Открыть окно предварит. просмотра документа
GET(WMFQue,NextEntry)         !Взять первый элемент списка
?ImageField{PROP:text} = WMFQue !Подготовить первую страницу документа
ACCEPT
CASE ACCEPTED()
OF ?NextPage
  NextEntry += 1              !Увеличить счетчик элементов
  IF NextEntry > RECORDS(WMFQue) THEN CYCLE .      !Конец документа?
  GET(WMFQue,NextEntry)       !Взять следующий элемент списка
  ?ImageField{PROP:text} = WMFQue !Подготовить след. страницу документа
  DISPLAY                     ! и показать на экране
OF ?PrintReport
  Report{PROP:Flushpreview} = ON !Сбросить файлы на принтер
  BREAK
      OF ?ExitReport
  BREAK !Завершить процедуру
...
CLOSE(ViewReport)            !Закрыть окно
FREE(WMFQue)                 !Освободить память, занимаемую списком
CLOSE(Report)                !Закрыть документ (удаляя все .WMF файлы)
RETURN ! и возвратиться в программу

```

**Смотри также:** Переход на новую страницу, PREVIEW

## OPEN (открыть структуру REPORT для работы)

**OPEN**(*документ*)

**OPEN**                      Активизирует структуру REPORT.  
*документ*                  Метка структуры REPORT.

Оператор **OPEN** активизирует структуру REPORT. Перед тем, как печатать какую-либо структуру, нужно открыть структуру REPORT.

### Пример:

OPEN(CustRpt) !Открыть документ

## PRINT (напечатать структуру)

**PRINT**([ *структура* ] )  
           [ *документ, номер* ]

**PRINT**                      Печатает структуры документа DETAIL, HEADER или FOOTER.  
*структура*                  Метка структуры DETAIL.  
*документ*                  Метка структуры REPORT.  
*номер*                      Номер или метка соответствия структуры документа, которую  
                                  требуется напечатать (имеет смысл только с параметром документ).

Оператор **PRINT** печатает структуру документа на “устройство”, указанное пользователем в окне Print... системы Windows. Если необходимо, PRINT автоматически обрабатывает ситуации завершения раздела и перехода на новую страницу.

### Пример:

```
BuildRpt PROCEDURE
CustRpt REPORT
    HEADER,USE(?PageHeader) !Верхний колонтитул страницы элементы структуры
    END
CustDetail DETAIL,USE(?Detail) !Верхний колонтитул страницы
    !элементы структуры
    END
END
CODE
PRINT(CustDetail) !Напечатать строку заказа
PrintRpt(CustRpt,?PageHeader) !Передать процедуре печати документ и метку соответствия структуры
PrintRpt PROCEDURE(RptToPrint,DetailNumber)
CODE
PRINT(RptToPrint,DetailNumber) !Напечатать структуру документа
```

**Смотри также:** Переход на новую страницу, BREAK

## Глава 10 Графические команды

### Предисловие

В этой главе определяются “графические примитивы” языка Clarion, которые позволяют “рисовать” в окнах и печатаемых документах.

Выводимые в окно графические данные всегда служат фоном для изображения управляющих полей. Графика позволяет нагляднее представить все управляющие поля окна, тем самым помогая пользователю при выборе требуемого поля.

### Текущий объект

---

Графические данные выводятся только в “текущий объект”. Если не было переустановки посредством SETTARGET, то “текущий объект” - это последнее открытое (и еще не закрытое) в текущем процессе окно, которое удерживает фокус ввода. Графические данные устойчивы в окне: они автоматически восстанавливаются библиотекой времени исполнения.

Графику (рисунки) можно включать и в документы. Для этого нужно использовать оператор SETTARGET, чтобы в качестве “текущего объекта” была установлена структура REPORT.

У каждого окна или документа есть свой текущий “фломастер” (pen), который определяет толщину, цвет и тип рисуемой линии. Поэтому, для согласованного применения в нескольких окнах одного и того же фломастера (параметры которого отличны от параметров по умолчанию) нужно в каждом окне произвести установки параметров фломастера, используя операторы SETPENWIDTH, SETPENCOLOR и SETPENSTYLE.

### Графика и система координат

---

Начало ( $x=0$ ,  $y=0$ ) системы координат для графики (графическая система координат) находится в верхнем левом углу окна. Координаты указываются в условных единицах (для REPORT условные единицы могут быть заменены атрибутами THOUS, MM или POINTS на соответствующие единицы). За условные единицы принимаются одна четверть усредненной ширины символа шрифта и одна восьмая его усредненной высоты, причем шрифт устанавливается FONT-атрибутом окна (или берется системный шрифт, если для окна не указан FONT атрибут).

Если графическая картинка не попадает в текущую видимую часть окна, то ее можно просмотреть, используя механизм прокрутки окна. Размеры виртуального экрана, по которому осуществляется прокрутка окна, автоматически так увеличиваются, чтобы

охватить все графические картинки окна. Отображение графики за пределы видимой части окна приводит к автоматическому появлению (если для окна указан один из атрибутов HSCROLL, VSCROLL или HVSCROLL) линеек прокрутки.

Графические процедуры

ARC (нарисовать дугу эллипса)

ARC( <i>x, y, ширина, высота, угол_нач, угол_кон</i> )	
ARC	Рисует дугу эллипса в текущем окне или документе.
<i>x</i>	Целочисленное выражение для указания горизонтального положения начальной точки.
<i>y</i>	Целочисленное выражение для указания вертикального положения начальной точки.
<i>ширина</i>	Целочисленное выражение для указания ширины.
<i>высота</i>	Целочисленное выражение для указания высоты.
<i>угол_нач</i>	Целочисленное выражение, определяющее начальную точку дуги. Указывается в десятых долях градуса (10=1 градус). Отсчет угла производится против часовой стрелки, начиная с трех часов.
<i>угол_кон</i>	Целочисленное выражение, определяющее конечную точку дуги. Указывается в десятых долях градуса (10=1 градус). Отсчет угла производится против часовой стрелки, начиная с трех часов.

Процедура **ARC** рисует дугу эллипса в текущем окне или документе.

Эллипс прорисовывается внутри “ограничивающего прямоугольника”, размеры которого определяются параметрами *x, y, ширина* и *высота*. Параметры *x* и *y* определяют начальную точку, а параметры *ширина* и *высота* определяют горизонтальный и вертикальный размеры “ограничивающего прямоугольника”.

Параметры *угол\_нач* и *угол\_кон* определяют ту часть линии эллипса, которую требуется изобразить.

Цвет линии эллипса - цвет, установленный оператором SETPENCOLOR для текущего фломастера. Цвет по умолчанию - цвет, установленный Windows для текста окна. Толщина линии - текущая толщина, установленная SETPENWIDTH, толщина по умолчанию - один пиксел. Тип линии - тип линии, установленный SETPENSTYLE для текущего фломастера, тип линии по умолчанию - сплошная линия.

Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
```

```

!управляющие поля окна
END
CODE
OPEN(MDIChild)
ARC(100,50,100,50,0,900)          !Изобразить дугу в 90 градусов от 3 до 12 часов -
! дуга верхнего правого квадранта эллипса

```

## BLANK (стереть графические изображения)

**BLANK**(*[x][, y][, ширина][, высота]*)

<b>BLANK</b>	Стирает все графические изображения, попадающие в указанную область окна или документа.
<i>x</i>	Целочисленное выражение для указания горизонтального положения начальной точки. Если параметр не указан, то по умолчанию равен нулю.
<i>y</i>	Целочисленное выражение для указания вертикального положения начальной точки. Если параметр не указан, то по умолчанию равен нулю.
<i>ширина</i>	Целочисленное выражение для указания ширины. Если параметр не указан, то по умолчанию равен ширине окна.
<i>высота</i>	Целочисленное выражение для указания высоты. Если параметр не указан, то по умолчанию равен высоте окна.

Процедура **BLANK** стирает все графические изображения, попадающие в указанную область окна или документа. Управляющие поля не стираются. **BLANK** без параметров стирает во всем окне или документе.

### Пример:

```

MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
!Управляющие поля окна
END
CODE
OPEN(MDIChild)
ARC(100,50,100,50,0,900)          !Нарисовать дугу
BLANK    !И стереть ее

```

## BOX (нарисовать прямоугольник)

**BOX**(*x, y, ширина, высота[,раскраска]*)

<b>BOX</b>	Рисует прямоугольник в текущем окне или документе.
<i>x</i>	Целочисленное выражение для указания горизонтального положения начальной точки.
<i>y</i>	Целочисленное выражение для указания вертикального положения начальной точки.

<i>ширина</i>	Целочисленное выражение для указания ширины.
<i>высота</i>	Целочисленное выражение для указания высоты.
<i>раскраска</i>	Целочисленная константа типа LONG или ULONG, символическое имя, переменная, в трех младших байтах которых находятся красная зеленая и синяя компоненты цвета или символическое имя для значения стандартного цвета Windows.

Процедура **BOX** рисует прямоугольник в текущем окне или документе. Положение и размеры прямоугольника определяются параметрами *x*, *y*, *ширина* и *высота*. Параметры *x* и *y* определяют начальную точку, а параметры *ширина* и *высота* определяют горизонтальный и вертикальный размеры прямоугольника. Прямоугольник распространяется вправо и вниз от начальной точки.

Цвет линии прямоугольника - цвет, установленный оператором SETPENCOLOR для текущего фломастера. Цвет по умолчанию - цвет, установленный Windows для текста окна. Толщина линии - текущая толщина, установленная SETPENWIDTH, толщина по умолчанию - один пиксел. Тип линии - тип линии, установленный SETPENSTYLE для текущего фломастера, тип линии по умолчанию - сплошная линия.

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
    !Управляющие поля окна
END
CODE
OPEN(MDIChild)
BOX(100,50,100,50,00FF0000h)      !Прямоугольник, раскрашенный в красный цвет
```

## CHORD (нарисовать сектор эллипса)

**CHORD**(*x*, *y*, *ширина*, *высота*, *угол\_нач*, *угол\_кон* [,*раскраска*])

<b>CHORD</b>	Рисует замкнутый сектор эллипса в текущем окне или документе.
<i>x</i>	Целочисленное выражение для указания горизонтального положения начальной точки.
<i>y</i>	Целочисленное выражение для указания вертикального положения начальной точки.
<i>ширина</i>	Целочисленное выражение для указания ширины.
<i>высота</i>	Целочисленное выражение для указания высоты.
<i>угол_нач</i>	Целочисленное выражение, определяющее начальную точку дуги. Указывается в десятых долях градуса (10=1 градус). Отсчет угла производится против часовой стрелки, начиная с трех часов.
<i>угол_кон</i>	Целочисленное выражение, определяющее конечную точку дуги. Указывается в десятых долях градуса (10=1 градус). Отсчет угла



производится против часовой стрелки, начиная с трех часов.

*раскраска*

Целочисленная константа типа LONG или ULONG, символическое имя, переменная, в трех младших байтах которых находятся красная зеленая и синяя компоненты цвета или символическое имя для значения стандартного цвета Windows.

Процедура **CHORD** рисует замкнутый сектор эллипса в текущем окне или документе. Эллипс прорисовывается внутри “ограничивающего прямоугольника”, размеры которого определяются параметрами x, y, ширина и высота. Параметры x и y определяют начальную точку, а параметры ширина и высота определяют горизонтальный и вертикальный размеры “ограничивающего прямоугольника”. Параметры угол\_нач и угол\_кон определяют ту часть линии эллипса (дугу), сектор которой требуется изобразить. Конечные точки дуги соединяются отрезком прямой линии (хордой).

Цвет линии границы - цвет, установленный оператором SETPENCOLOR для текущего фломастера. Цвет по умолчанию - цвет, установленный Windows для текста окна. Толщина линии - текущая толщина, установленная SETPENWIDTH, толщина по умолчанию - один пиксел. Тип линии - тип линии, установленный SETPENSTYLE для текущего фломастера, тип линии по умолчанию - сплошная линия.

**Пример:**

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
    !управляющие поля окна
END
CODE
OPEN(MDIChild)
CHORD(100,50,100,50,0,900,00FF0000h) !Красный “полумесяц” с дугой в 90 градусов
```

## ELLIPSE (нарисовать эллипс)

**ELLIPSE**(*x, y, ширина, высота [,раскраска]*)

**ELLIPSE**

Рисует эллипс в текущем окне или документе.

*x*

Целочисленное выражение для указания горизонтального положения начальной точки.

*y*

Целочисленное выражение для указания вертикального положения начальной точки.

*ширина*

Целочисленное выражение для указания ширины.

*высота*

Целочисленное выражение для указания высоты.

*раскраска*

Целочисленная константа типа LONG или ULONG, символическое имя, переменная, в трех младших байтах которых находятся красная зеленая и синяя компоненты цвета или символическое имя для значения стандартного цвета Windows.

Процедура **ELLIPSE** рисует эллипс в текущем окне или документе. Эллипс прорисовывается внутри “ограничивающего прямоугольника”, размеры которого определяются параметрами *x*, *y*, ширина и высота. Параметры *x* и *y* определяют начальную точку, а параметры ширина и высота определяют горизонтальный и вертикальный размеры “ограничивающего прямоугольника”.

Цвет линии границы - цвет, установленный оператором **SETPENCOLOR** для текущего фломастера. Цвет по умолчанию - цвет, установленный Windows для текста окна. Толщина линии - текущая толщина, установленная **SETPENWIDTH**, толщина по умолчанию - один пиксел. Тип линии - тип линии, установленный **SETPENSTYLE** для текущего фломастера, тип линии по умолчанию - сплошная линия.

**Пример:**

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
    !управляющие поля окна
END
CODE
OPEN(MDIChild)
ELLIPSE(100,50,100,50,00FF0000h)    !Эллипс раскрашенный красным цветом
```

**IMAGE (нарисовать эллипс)**

**IMAGE**(*x*, *y*, *ширина*, *высота*, *имяфайла*)

<b>IMAGE</b>	Размещает графическую картинку в текущем окне или документе.
<i>x</i>	Целочисленное выражение для указания горизонтального положения начальной точки.
<i>y</i>	Целочисленное выражение для указания вертикального положения начальной точки.
<i>ширина</i>	Целочисленное выражение для указания ширины. Может принимать отрицательные значения. Если этот параметр опущен, то берется ширина оригинала изображения.
<i>высота</i>	Целочисленное выражение для указания высоты. Может принимать отрицательные значения. Если этот параметр опущен, то берется высота оригинала изображения.
<i>имяфайла</i>	Строковая константа или переменная, указывающие имя отображаемого файла

Процедура **IMAGE** размещает графическую картинку в текущем окне или документе в области, местоположение которой и размеры указываются параметрами *x*, *y*, ширина и высота. В качестве графической картинкой может выступать битовое отображение (.BMP), пиктограмма (.ICO), PaintBrush-объект (.PCX), GIF-объект (.GIF), JPEG-объект (.JPG)

или метафайл Windows (.WMF).

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
    !управляющие поля окна
END
CODE
OPEN(MDIChild)
IMAGE(100,50,100,50,'LOGO.BMP')    !Вывести графическую картинку
```

## LINE (изобразить прямую линию)

**LINE**(*x, y, ширина, высота*)

<b>LINE</b>	Рисует прямую линию в текущем окне или документе.
<i>x</i>	Целочисленное выражение для указания горизонтального положения начальной точки.
<i>y</i>	Целочисленное выражение для указания вертикального положения начальной точки.
<i>ширина</i>	Целочисленное выражение для указания ширины. Может принимать отрицательные значения.
<i>высота</i>	Целочисленное выражение для указания высоты. Может принимать отрицательные значения.

Процедура **LINE** помещает в текущее окно или документ прямую линию. Начальная точка, наклон и длина линии определяются параметрами *x*, *y*, *ширина* и *высота*. Параметры *x* и *y* определяют начальную точку прямой, а параметры *ширина* и *высота* указывают расстояние по горизонтали и по вертикали до конечной точки. Если их значения - положительные числа, то линия распространяется вправо и вниз от начальной точки. Если значение параметра *ширина* отрицательно, то линия распространяется влево от начальной точки; если значение параметра *высота* отрицательно, то линия распространяется влево от начальной точки. Если либо *ширина*, либо *высота* - нулевые, то линия либо вертикальна, либо горизонтальна.

<u>Ширина</u>	<u>Высота</u>	<u>Результат</u>
положительная	положительная	вправо и вниз от начальной точки
отрицательная	отрицательная	влево и вниз от начальной точки
положительная	отрицательная	вправо и вверх от начальной точки
отрицательная	положительная	влево и вверх от начальной точки
нулевая	положительная	вертикально вниз от начальной точки
нулевая	отрицательная	вертикально вверх от начальной точки
положительная	нулевая	горизонтально вправо от начальной точки
отрицательная	нулевая	горизонтально влево от начальной точки

Цвет линии - цвет, установленный оператором SETPENCOLOR для текущего фломастера. Цвет по умолчанию - цвет, установленный Windows для текста окна. Толщина линии - текущая толщина, установленная SETPENWIDTH, толщина по умолчанию - один пиксел. Тип линии - тип линии, установленный SETPENSTYLE для текущего фломастера, тип линии по умолчанию - сплошная линия.

**Пример:**

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
!управляющие поля окна
END
CODE
OPEN(MDIChild)
LINE(100,50,100,50)           !Изобразить линию
```

**PIE (изобразить секторную диаграмму)**

**PIE**(*х, у, ширина, высота, секции, цвета* [*, глубина*][*, значениеобщ*][*, угол\_нач*])

<b>PIE</b>	Рисует секторную диаграмму в текущем окне или документе.
<i>х</i>	Целочисленное выражение для указания горизонтального положения начальной точки.
<i>у</i>	Целочисленное выражение для указания вертикального положения начальной точки.
<i>ширина</i>	Целочисленное выражение для указания ширины.
<i>высота</i>	Целочисленное выражение для указания высоты.
<i>секции</i>	Массив числовых значений, определяющих относительный размер каждой секции диаграммы.
<i>цвета</i>	Целочисленный массив для задания цвета раскраски каждой секции диаграммы.
<i>глубина</i>	Целочисленное выражение, задающее глубину трехмерной диаграммы. Если параметр не указан, то диаграмма - двумерная.
<i>значениеобщ</i>	Числовая константа или переменная, определяющие суммарную величину, необходимую для создания полной диаграммы. Если параметр не указан, то используется сумма значений элементов массива секциию
<i>угол_нач</i>	Числовая константа или переменная, определяющие начальное положение первой секции диаграммы. Значение параметра рассматривается как доля (пропорциональная) значения параметра значениеобщ. Если параметр не указан (или равен нулю), то начало первой секции диаграммы соответствует 12 часам на циферблате часов.

Процедура **PIE** создает секторную диаграмму в текущем окне или документе. Диаграмма (эллипс) прорисовывается внутри “ограничивающего прямоугольника”, размеры которого

определяются параметрами *x*, *y*, ширина и высота. Параметры *x* и *y* определяют начальную точку, а параметры ширина и высота определяют горизонтальный и вертикальный размеры “ограничивающего прямоугольника”.

Секции диаграммы, как пропорциональные части значенияобщ, создаются по ходу часовой стрелки, начиная с положения, заданного в *угол\_нач*. Задание значениеобщ большего чем сумма значений элементов массива секции приводит к созданию диаграммы с отсутствующим сектором.

Цвет линий - цвет, установленный оператором **SETPENCOLOR** для текущего фломастера. Цвет по умолчанию - цвет, установленный Windows для текста окна. Толщина линий - текущая толщина, установленная **SETPENWIDTH**, толщина по умолчанию - один пиксел. Тип линий - тип линии, установленный **SETPENSTYLE** для текущего фломастера, тип линии по умолчанию - сплошная линия.

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
!управляющие поля окна
END
SliceSize BYTE,DIM(4)
SliceColor LONG,DIM(4)
CODE
SliceSize[1] = 90
SliceColor[1] = 0 !Черный цвет
SliceSize[2] = 90
SliceColor[2] = 00FF0000h !Красный цвет
SliceSize[3] = 90
SliceColor[3] = 0000FF00h !Зеленый цвет
SliceSize[4] = 90
SliceColor[4] = 000000FFh !Голубой цвет
OPEN(MDIChild)
PIE(100,50,100,50,SliceSize,SliceColor)
!Изобразить секторную диаграмму, содержащую четыре равные
!секции, начинающуюся с 12 часов. Секторы Черный, Красный,
!Зеленый и Голубой следуют против часовой стрелки.
```

## POLYGON (изобразить многозвенную фигуру)

**POLYGON**(массив [,раскраска])

### POLYGON

Рисует многозвенную фигуру в текущем окне или документе.

*массив*

Целочисленный массив для задания *x* и *y* координат каждой “угловой точки” многоугольника.

*раскраска*

Целочисленная константа типа **LONG** или **ULONG**, символическое

имя, переменная, в трех младших байтах которых находятся красная зеленая и синяя компоненты цвета или символическое имя для значения стандартного цвета Windows.

Процедура **POLYGON** помещает многозвенную фигуру в текущее окно или документ. Многоугольник всегда замкнут.

Массив содержит x и y координаты каждой “угловой точки” многоугольника. Число угловых точек многоугольника равно половине числа элементов массива. Для каждой угловой точки в общей последовательности точек координата x выбирается из нечетного элемента массива, координата y - из расположенного за ним четного элемента.

Цвет линии границы - цвет, установленный оператором SETPENCOLOR для текущего фломастера. Цвет по умолчанию - цвет, установленный Windows для текста окна. Толщина линии - текущая толщина, установленная SETPENWIDTH, толщина по умолчанию - один пиксел. Тип линии - тип линии, установленный SETPENSTYLE для текущего фломастера, тип линии по умолчанию - сплошная линия.

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
    !управляющие поля окна
END
Corners SHORT,DIM(8)
CODE
Corners[1] = 0           !x координата 1-ой точки
Corners[2] = 90          !y координата 1-ой точки
Corners[3] = 90          !x координата 2-ой точки
Corners[4] = 190         !y координата 2-ой точки
Corners[5] = 100         !x координата 3-ей точки
Corners[6] = 200         !y координата 3-ей точки
Corners[7] = 50          !x координата 4-ой точки
Corners[8] = 60          !y координата 4-ой точки
OPEN(MDIChild)
POLYGON(Corners,000000FFh) !Четырехугольник, закрашенный голубым цветом
```

## ROUNDBOX (нарисовать прямоугольник с округлыми углами)

**ROUNDBOX**(*x, y, ширина, высота[,раскраска]*)

**ROUNDBOX** Рисует прямоугольник с округлыми углами в текущем окне или документе.

*x* Целочисленное выражение для указания горизонтального положения начальной точки.

<i>у</i>	Целочисленное выражение для указания вертикального положения начальной точки.
<i>ширина</i>	Целочисленное выражение для указания ширины.
<i>высота</i>	Целочисленное выражение для указания высоты.
<i>раскраска</i>	Целочисленная константа типа LONG или ULONG, символическое имя, переменная, в трех младших байтах которых находятся красная зеленая и синяя компоненты цвета или символическое имя для значения стандартного цвета Windows.

Процедура **ROUNDBOX** рисует прямоугольник с округлыми углами в текущем окне или документе. Положение и размеры прямоугольника определяются параметрами *х*, *у*, *ширина* и *высота*.

Параметры *х* и *у* определяют начальную точку, а параметры *ширина* и *высота* определяют горизонтальный и вертикальный размеры прямоугольника. Прямоугольник распространяется вправо и вниз от начальной точки.

Цвет линии прямоугольника - цвет, установленный оператором **SETPENCOLOR** для текущего фломастера. Цвет по умолчанию - цвет, установленный Windows для текста окна. Толщина линии - текущая толщина, установленная **SETPENWIDTH**, толщина по умолчанию - один пиксел. Тип линии - тип линии, установленный **SETPENSTYLE** для текущего фломастера, тип линии по умолчанию - сплошная линия.

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
    !Управляющие поля окна
END
CODE
OPEN(MDIChild)
ROUNDBOX(100,50,100,50,00FF0000h)    !"Сглаженный" прямоугольник,
                                        ! раскрашенный в красный цвет
```

## SETPENCOLOR (установить цвет линий)

**SETPENCOLOR**([*цвет*])

**SETPENCOLOR** Устанавливает цвет текущего фломастера.  
*цвет* Целочисленная константа типа LONG или ULONG, символическое имя, переменная, в трех младших байтах которых находятся красная зеленая и синяя компоненты цвета или символическое имя для значения стандартного цвета Windows. Если параметр опущен, то выбирается цвет, установленный системой Windows для вывода текста в окно.

Процедура **SETPENCOLOR** устанавливает для всех графических процедур цвет текущего фломастера. По умолчанию - выбирается цвет, установленный системой Windows для вывода текста в окно.

Для каждого окна можно установить свой цвет текущего фломастера. Поэтому, для согласованного применения в нескольких окнах одного и того же фломастера (цвет от которого устанавливается не по умолчанию) нужно в каждом окне произвести установку параметра цвета фломастера, используя оператор SETPENCOLOR.

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
    !Управляющие поля окна
END
CODE
OPEN(MDIChild)
SETPENCOLOR(000000FFh)          !Выбрать голубой фломастер
ROUNDBOX(100,50,100,50,00FF0000h)!”Сглаженный” прямоугольник
    ! с голубым контуром,
    ! раскрашенный в красный цвет
```

## SETPENSTYLE (установить тип линий)

**SETPENSTYLE**(*[mun]*)

### SETPENSTYLE

Устанавливает тип линии для текущего фломастера.

*mun* Целочисленная константа, символическое имя, переменная для указания типа линии. Если параметр опущен, то устанавливаемый тип - сплошная линия.

Процедура **SETPENSTYLE** устанавливает для всех графических процедур тип линии, вычерчиваемой текущим фломастером. Тип по умолчанию - сплошная линия.

Для каждого окна можно установить свой тип линии. Поэтому, для согласованного применения в нескольких окнах одного и того же фломастера (тип линии которого устанавливается не по умолчанию) нужно в каждом окне произвести установку параметра типа линии, используя оператор SETPENSTYLE.

В файле EQUATES.CLW есть операторы EQUATE для типов линий. Ниже приведена их представительная выборка (полный список смотри в EQUATES.CLW):

```
PEN:solid   Сплошная линия
PEN:dash    Штриховая линия
PEN:dot     Пунктирная линия
PEN:dashdot Штрих-пунктирная линия
```



**Пример:**

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
    !Управляющие поля окна
END
CODE
OPEN(MDIChild)
SETPENCOLOR(000000FFh)           !Выбрать голубой фломастер
SETPENSTYLE(PEN:dash)            !Выбрать пунктирную линию
ROUNDBOX(100,50,100,50,00FF0000h)!"Сглаженный" прямоугольник
    ! с голубым пунктирным контуром,
    ! раскрашенный в красный цвет
```

**SETPENWIDTH (установить толщину линий)**

**SETPENWIDTH**([толщина])

**SETPENWIDTH**

Устанавливает толщину линии текущего фломастера.

*толщина*

Целочисленное выражение, определяющее толщину линии.

Толщина указывается в условных единицах при условии, что единица измерения не была переустановлена одним из атрибутов THOUSINCH, MILLIMETERS или POINTS. Если параметр опущен, то толщина (один пиксел) устанавливается по умолчанию.

Процедура **SETPENWIDTH** устанавливает для всех графических процедур толщину вычерчиваемой линии. Толщина по умолчанию - один пиксел; такая же толщина устанавливается когда параметр толщина равен 0.

Для каждого окна можно установить свою толщину линии. Поэтому, для согласованного применения в нескольких окнах одного и того же фломастера (толщина линии которого устанавливается не по умолчанию) нужно в каждом окне произвести установку параметра толщины линии, используя оператор **SETPENWIDTH**.

**Пример:**

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
    !Управляющие поля окна
END
CODE
OPEN(MDIChild)
SETPENCOLOR(000000FFh)           !Выбрать голубой фломастер
SETPENSTYLE(PEN:dash)            !Выбрать пунктирную линию
SETPENWIDTH(2)                   ! толщиной в две условных единицы
ROUNDBOX(100,50,100,50,00FF0000h)!"Сглаженный" прямоугольник
```

! с жирным голубым пунктирным контуром,  
! раскрашенный в красный цвет

## SHOW (отобразить на экран)

**SHOW**(*x, y, строка*)

**SHOW** Выводит строку в текущее окно или документ.

*x* Целочисленное выражение для указания горизонтального положения начальной точки. Задается в условных единицах.

*y* Целочисленное выражение для указания вертикального положения начальной точки. Задается в условных единицах.

*строка* Строковая константа, переменная или выражение, содержащие отформатированный текст, предназначенный для вывода его в текущее окно или в документ.

**SHOW** выводит строку текста в текущее окно или в документ. Используемый при этом шрифт текущий шрифт для окна или для документа.

Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
!Управляющие поля окна
END
CODE
OPEN(MDIChild)
SHOW(100,100,FORMAT(TODAY(),@D3))      !Отобразить дату
SHOW(20,20,'Press Any Key to Continue') !Вывести сообщение
```

## TYPE (вывести строку на экран)

**TYPE**(*строка*)

**TYPE** Выводит строку в текущее окно или документ.

*строка* Строковая константа, переменная или выражение.

**TYPE** выводит строку текста в текущее окно или в документ. Вывод строки в окне или документе начинается с текущего положения курсора и, если строка переходит правую границу, продолжается слева. Используемый при этом шрифт - текущий шрифт для окна или для документа. Перед выводом строки оператором **TYPE** для установки курсора можно воспользоваться оператором **SHOW**.

Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
    !Управляющие поля окна
END
CODE
OPEN(MDIChild)
TYPE(Cus:Notes)                !Вывести на экран поле примечаний
```

## Графические функции

### PENCOLOR (возвратить цвет линии)

#### PENCOLOR()

Функция **PENCOLOR** возвращает установленный процедурой SETPENCOLOR цвет текущего фломастера.

Тип возвращаемых данных: LONG

#### Пример:

```
Proc1    PROCEDURE
MDIChild1 WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
    !управляющие поля окна
END
CODE
OPEN(MDIChild1)
SETPENCOLOR(000000FFh)        !Выбрать голубой фломастер
Proc2    !Вызов процедуры

Proc2    PROCEDURE
MDIChild2 WINDOW('Child Two'),AT(0,0,320,200),MDI,MAX,HVSCROLL
    !управляющие поля окна
END
ColorNow LONG
CODE
ColorNow = PENCOLOR()         !Прочитать цвет текущего фломастера
OPEN(MDIChild2)
SETPENCOLOR(ColorNow)         !Установить такой же цвет фломастера
SETPENSTYLE(PEN:dash)         !Установить тип линии - штриховая
SETPENWIDTH(2)                !Установить толщину линии в 2 условных единицы
BOX(100,50,100,50,00FF0000h) !Прямоугольник
    ! с жирным голубым штриховым контуром,
    ! раскрашенный в красный цвет
```



**PENWIDTH (возвратить толщину линии)****PENWIDTH()**

Функция **PENWIDTH** возвращает установленную процедурой **SETPENWIDTH** толщину линии текущего фломастера. Значение возвращается в условных единицах (если не было переопределено в **REPORT** атрибутами **THOUS**, **MM** или **POINTS**).

Тип возвращаемых данных: **LONG**

**Пример:**

```
Proc1    PROCEDURE
MDIChild1  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
    !управляющие поля окна
    END
CODE
OPEN(MDIChild1)
SETPENCOLOR(000000FFh)          !Выбрать голубой фломастер
SETPENSTYLE(PEN:dash)          !Выбрать тип линии - штриховая
SETPENWIDTH(2)                 !Выбрать толщину линии в 2 условных единицы
Proc2    !Вызов процедуры

Proc2    PROCEDURE
MDIChild2  WINDOW('Child Two'),AT(0,0,320,200),MDI,MAX,HVSCROLL
    !управляющие поля окна
    END
ColorNow  LONG
StyleNow  LONG
WidthNow  LONG
CODE
ColorNow = PENCOLOR()          !Прочитать цвет текущего фломастера
StyleNow = PENSTYLE()          !Прочитать тип линии текущего фломастера
OPEN(MDIChild2)
SETPENCOLOR(ColorNow)          !Установить такой же цвет фломастера
SETPENSTYLE(StyleNow)          !Установить такой же тип линии
SETPENWIDTH(WidthNow)          !Установить такую же толщину линии
BOX(100,50,100,50,00FF0000h)   !Прямоугольник
    ! с жирным голубым штриховым контуром,
    ! раскрашенный в красный цвет
```



## Глава 11 Файлы данных

### Структуры для работы с файлами данных

#### FILE (объявить структуру файла данных)

<i>метка</i>	<b>FILE,DRIVER()</b> [ <b>,CREATE</b> ] [ <b>,RECLAIM</b> ] [ <b>,OWNER()</b> ] [ <b>,ENCRYPT</b> ]
	[ <b>,NAME()</b> ] [ <b>,PRE()</b> ] [ <b>,BINDABLE</b> ] [ <b>,THREAD</b> ]
	[ <b>,EXTERNAL</b> ][ <b>,DLL</b> ] [ <b>,OEM</b> ]
<i>метка</i>	[ <b>INDEX()</b> ]
<i>метка</i>	[ <b>KEY()</b> ]
<i>метка</i>	[ <b>MEMO()</b> ]
<i>label</i>	[ <b>BLOB</b> ]
[ <i>метка</i> ]	<b>RECORD</b>
	<i>поля</i>
.	.

<b>FILE</b>	Объявляет файл данных.
<b>DRIVER</b>	Указывает тип файла данных. Атрибут DRIVER обязателен для объявления любого файла.
<b>CREATE</b>	Разрешает создание файла во время выполнения программы оператором CREATE.
<b>RECLAIM</b>	Задаёт повторное использование пространства, занимаемого логически удалёнными записями.
<b>OWNER</b>	Задаёт ключевое слово для шифрования данных.
<b>ENCRYPT</b>	Задаёт шифрование данных в файле.
<b>NAME</b>	Указывает спецификацию файла (путь в структуре каталогов и имя).
<b>PRE</b>	Объявляет префикс для имен в данной структуре.
<b>BINDABLE</b>	Указывает, что все переменные из структуры RECORD можно использовать в динамических выражениях.
<b>THREAD</b>	Указывает, что, при открытии файла в исполняемом процессе, память для буфера записи файла выделяется отдельно для каждого процесса.
<b>EXTERNAL</b>	Указывает, что структура FILE описывается во внешней библиотеке и там же для нее выделяется буфер
<b>DLL</b>	Указывает, данная структура FILE описывается в библиотеке DLL. Этот атрибут дополнительно требует атрибута EXTERNAL
<b>OEM</b>	Указывает, что строковые данные при чтении с диска преобразуются из OEM ASCII в ANSI, а при записи, наоборот, из ANSI в OEM ASCII.
<b>MEMO</b>	Объявляет текстовое поле переменной длины (до 64 К байт).
<b>BLOB</b>	Объявляет мемо-поле переменной длины, которое может быть больше 64К.

**RECORD**

Объявляет структуру записи файла, состоящую из полей. Структура RECORD обязательна для любого объявления файла.

*поля*

Элементы данных в структуре RECORD.

Оператор **FILE** объявляет структуру файла данных, которая представляет собой точное описание находящегося на диске файла. Метка структуры FILE используется в операторах и функциях, которые осуществляют операции с дисковым файлом. Структура FILE заканчивается точкой или оператором END.

Все атрибуты операторов FILE, KEY, INDEX, MEMO, операторов объявления полей и типы данных, которые может содержать структура FILE, зависят от поддержки файловым драйвером (значения атрибута DEVICE). Любые атрибуты и типы данных в объявлении файла, которые не поддерживаются файловой системой, указанной атрибутом DRIVER, приведут к появлению сообщений об ошибке, выдаваемого файловым драйвером при открытии файла. Исключения в атрибутах и/или типах данных перечисляются в документации по файловым драйверам.

Во время выполнения программы структура RECORD назначается области памяти, используемой в качестве буфера данных, в котором прочитанные с диска записи могут обрабатываться исполняемыми операторами. В структуре FILE обязательно должна присутствовать структура RECORD.

Структура FILE с атрибутом BINDABLE предполагает, что все переменные из структуры RECORD доступны для использования в динамических выражениях, без обязательного выполнения оператора BIND для каждого поля (позволяя выполнить один оператор BIND(файл), чтобы сделать все доступными все поля файла). Содержимое параметра NAME для каждой переменной является логическим именем, используемым в динамическом выражении. Если атрибут NAME не указан, то используется имя переменной (включая префикс). Для имен всех переменных структуры в EXE-модуле резервируется память. Это увеличивает размер программы и затраты оперативной памяти. Поэтому атрибут BINDABLE следует применять только когда большая часть составляющих структуру полей планируется использовать в динамических выражениях.

Атрибут THREAD указывает, что для каждого исполняемого процесса, в котором открывается файл, выделяется отдельный буфер для записи и блок управления файлом (FCB). Если в процессе файл не открывается, то для него и не выделяется буфер записи. Файл с атрибутом EXTERNAL объявляется и его можно использовать в исполняемых операторах, но память для него не выделяется. Память для буфера записи этого файла резервируется во внешней библиотеке. Это позволяет программе на языке Clarion использовать файлы, объявленные как общие во внешней библиотеке.



**Пример:**

Names	FILE,DRIVER('Clarion')	!Объявить структуру файла
Rec	RECORD	!Обязательная структура RECORD
Name	STRING(20)	! содержащая один или более элемент данных
..		!Конец объявлений записи и файла

**CREATE (разрешить создание файла)****CREATE**

Атрибут **CREATE** в описании файла разрешает создание оператором CREATE дискового файла в программе, в которой этот файл объявляется. Тем самым привносятся некоторые накладные расходы, поскольку вся необходимая для этого информация должна содержаться в исполняемом модуле.

**Пример:**

Names	FILE,DRIVER('Clarion'),CREATE	!Объявить файл, разрешить создание его
Rec	RECORD	
Name	STRING(20)	
..		!Конец объявления файла

**DRIVER (указать тип файловой системы)****DRIVER**(тип файла[,строка])

<b>DRIVER</b>	Указывает какую файловую систему этот файл использует.
<i>тип файла</i>	Строковая константа, содержащая имя файлового менеджера (драйвера) (Btrieve, Clarion и т.д.)
<i>строка</i>	Строковая константа или переменная, содержащая любые дополнительные команды файловому драйверу.

Атрибут **DRIVER** указывает какой файловый драйвер используется для доступа к этому файлу данных. Это обязательный атрибут в объявлении любого файла.

Для осуществления физического доступа к файлу Clarion-программа использует файловый драйвер. Файловый драйвер действует как транслятор между Clarion-программой и файловой системой, исключая несвойственные для данной файловой системы команды. Файловые драйверы позволяют обращаться к файлам различных файловых систем, оставаясь в рамках синтаксиса языка Clarion.

Конкретный способ реализации каждой команды доступа к файлу в Clarion зависит от файлового драйвера. Из-за ограничений конкретной файловой системы некоторые команды

могут быть не реализованы в драйвере. Каждый файловый драйвер описывается в Руководстве пользователя. Там перечисляются все неподдерживаемые команды доступа к файлу, атрибуты объявления файла, типы данных и/или особенности файловой системы.

Пример:

```
Names  FILE,DRIVER('Clarion')           !Начало объявления файла
Record  RECORD
Name    STRING(24)
. .     !Конец объявления файла
```

NAME (задать имя файла)

NAME	([	<i>константа</i>		]
		<i>переменная</i>		

- NAME**                      Задает имя файла в файловой системе DOS.
- константа*                Строковая константа.
- переменная*              Метка статической строковой переменной. Переменная может быть объявлена как глобальная, в секции данных модуля или локальных данных, но с атрибутом **STATIC**.

Атрибут **NAME** указывает в операторе **FILE** имя файла в DOS для данного файлового драйвера. Если константа или переменная не содержит имени диска и пути по файловой системе, то подразумевается текущий диск и текущий каталог. Если опущено расширение, то подразумевается расширение, используемое для данного файлового драйвера по умолчанию.

Для некоторых файловых драйверов требуется, чтобы ключи, индексы и мемо-поля находились в отдельных файлах. Следовательно атрибут **NAME** можно также указывать в операторах **KEY**, **INDEX** и **МЕМО**. По умолчанию, атрибут **NAME** без параметров принимает значение метки оператора объявления структуры, в которой он помещается (включая любой заданный префикс).

Атрибут в форме **NAME(константа)** может использоваться для любого поля, объявленного в структуре **RECORD**. Таким образом обеспечивается для файлового драйвера такое имя поля, какое можно использовать в файловой системе данного драйвера.

Пример:

```
Cust      FILE,PRE(Cus),NAME(CustName) !Имя файла в переменной CustName
CustKey   KEY('Name'),NAME('c:\data\cust.idx') !Объявить ключ в файле cust.idx
Record    RECORD
Name      STRING(20)                      !Значение атрибута NAME по умолчанию равно 'Cus:Name'
```

..

Смотри также: FILE, KEY, INDEX

## ENCRYPT (шифрование файла данных)

### ENCRYPT

Атрибут **ENCRYPT** используется с атрибутом OWNER для того, чтобы скрыть информацию в файле данных. Этот атрибут допустим только вместе с атрибутом OWNER. Даже с помощью утилиты просмотра в шестнадцатеричном формате крайне трудно расшифровать данные в зашифрованном файле.

Пример:

```
Names  FILE,DRIVER('Clarion'),OWNER('Clarion'),ENCRYPT
Record  RECORD
Name    STRING(24)
```

..

Смотри также: OWNER

## OWNER (объявить пароль для шифрования данных)

### OWNER(пароль)

**OWNER**                      Задает пароль для шифрования данных.  
*пароль*                      Строковая константа или переменная.

Атрибут **OWNER** задает пароль, который используется для шифрования данных атрибутом ENCRYPT. Точная реализация этого атрибута зависит от файлового драйвера.

Пример:

```
Customer  FILE,DRIVER('Clarion'),OWNER('abCdeF'),ENCRYPT
           !Зашифровать данные с помощью ключевого слова "abCdeF"
Record    RECORD
Name      STRING(24)
```

..

Смотри также: атрибут ENCRYPT

**RECLAIM (использовать пространство удаленных записей)**

**RECLAIM**

Атрибут **RECLAIM** указывает, что файловый драйвер помещает новые записи в файл на место, ранее занимаемое удаленными теперь записями (если, конечно такое пространство имеется). В противном случае записи добавляются после последней записи файла. Реализация этого механизма зависит от файлового драйвера и не во всех файловых системах может быть осуществлена.

**Пример:**

```
Names FILE,DRIVER('Clarion'),RECLAIM !Использовать место удаленных записей
Record RECORD
Name STRING(20)
..
```

**PRE (задать префикс для переменных структуры)**

**PRE([префикс])**

**PRE** Обеспечивает префикс для переменных в составных структурах данных.

*префикс* Допустимы символы букв, цифр от 0 до 9 и символ подчеркивания. Префикс должен начинаться с буквы или символа подчеркивания. Хотя префикс может быть длинным, по традиции он состоит из 1-3-х букв.

Этот атрибут обеспечивает префикс для составных структур данных. Он используется для того, чтобы различать одноименные переменные в различных структурах. При использовании в исполняемых операторах, операторах присваивания и в списках параметров префикс присоединяется к имени переменной с помощью двоеточия (префикс:имя).

Для идентификации одноименных переменных, которые могут быть объявлены в структурах, не имеющих атрибута PRE, используется другой, более гибкий метод - синтаксис уточнения имен. При упоминании в исполняемых операторах, присвоениях и списках параметров к имени переменной через двоеточие спереди присоединяется имя содержащей данное поле структуры (GroupName:Label). В случае описания файла, которое содержит еще одну структуру - RECORD (у которой есть метка) к отдельному полю адресуются как FileLabel:RecordLabel:FieldName. Если же у структуры RECORD нет имени, то к отдельному полю обращаются так: FileLabel:FieldName.

**Пример:**

```

MasterFile  FILE,DRIVER('Clarion'),PRE(Mst)  !Объявить структуру главного файла
Record      RECORD
AcctNumber  LONG

..
Detail      FILE,DRIVER('Clarion'),PRE(Dtl)  !Объявить структуру файла Detail
Record      RECORD
AcctNumber  LONG

..
Message     GROUP,PRE(Mem)                   !Объявить переменные в памяти
            STRING(30)
            END
            CODE
            IF Dtl:AcctNumber <> Mst:AcctNumber  !Это новый счет
                Mem:Message = 'New Account'  !Вывести сообщение
                DO MatchMaster                !взять другую запись
            END
            IF Detail:Record:AcctNumber <> Masterfile:Record:AcctNumber !выражение
                Mem:Message = 'New Account'  !сообщение
                DO MatchMaster                ! получить новую запись
            END

```

**Смотри также :** Зарезервированные слова, Синтаксис уточнения имен.

**BINDABLE (переменная, используемая в динамических выражениях)****BINDABLE**

Атрибут **BINDABLE** оператора FILE, объявляет, что переменные, составляющие структуру RECORD можно использовать в динамических выражениях во время выполнения программы. Значение атрибута NAME каждой из переменных является логическим именем, используемым в динамических выражениях. Если атрибут NAME отсутствует, то используется имя переменной (включая префикс). Для имен всех переменных структуры в исполняемом модуле выделяется память. Таким образом программа становится больше и использует больше памяти чем обычно. Поэтому атрибут BINDABLE следует использовать, только если большая часть переменных структуры используется в динамических выражениях.

Перед тем, как можно будет использовать отдельное поле из структуры RECORD в динамическом выражении, нужно выполнить в программе оператор BIND в форме BIND(группа).

Пример:

```
Names      FILE,DRIVER('Clarion'),BINDABLE !Переменные из RECORD могут
использоваться
Record      RECORD                        ! в динамических выражениях
FileName    STRING(8),NAME('FILE')      !Динамическое имя FILE
Dot         STRING('.')                 !Динамическое имя: Dot
Extension   STRING(3),NAME('EXT')       !Динамическое имя: EXT
END
END
```

Смотри также: BIND, UNBIND, EVALUATE

**THREAD (отдельный буфер записи для каждого исполняемого процесса)**

**THREAD**

Атрибут **THREAD** указывает, что память для буфера записи (и управляющий блок файла) распределяется отдельно для каждого исполняемого процесса в программе. Таким образом значения, содержащиеся в буфере зависят от того, какой процесс выполняется.

Как только начат новый исполняемый процесс, файл для него должен быть открыт заново, чтобы получить новую копию буфера записи. При закрытии процесса память, выделенная для буфера записи файла, освобождается.

Пример:

```
PROGRAM
MAP
  Thread1
  Thread2
END
Names      FILE,DRIVER('Clarion'),PRE(Nam),THREAD      !Файл раздельно используемый
процессами
NbrNdx      INDEX(Nam:Number),OPT
Rec         RECORD
Name        STRING(20)
Number      SHORT
..
CODE
START(Thread1)
START(Thread2)

Thread1     PROCEDURE
CODE
OPEN(Names)                                !OPEN создает новую копию буфера записи
```

GET(Names,1)	! содержащую 1-ю запись файла
Thread2    PROCEDURE	
CODE	
OPEN(Names)	! OPEN создает другую копию буфера записи
GET(Names,5)	! содержащую 5-ю запись файла

**Смотри также:**    START, Объявление данных и распределение памяти

## EXTERNAL (файл объявлен во внешнем модуле)

### EXTERNAL( *member-модуль* )

**EXTERNAL**                    Указывает, что структура FILE описывается во внешней библиотеке.  
*member-модуль*            Строковая константа. Допустима только в объявлении файла. Содержит имя файла (без расширения) - member-модуля, содержащего описание файла без атрибута EXTERNAL. Если структура FILE описывается в программном модуле, то требуется чтобы параметр member-модуль был пустой строкой.

Атрибут **EXTERNAL** указывает что файл, к которому он относится, определен во внешней библиотеке. Таким образом структура FILE с атрибутом EXTERNAL объявляется и может использоваться в Clarion-программе, но память для буфера записи не выделяется. Память для буфера записи такого файла выделяется во внешней библиотеке. Этот атрибут позволяет Clarion-программе получить доступ к файлам, объявленным во внешней библиотеке как "public" - общие.

При использовании атрибута EXTERNAL(member-модуль) для объявления файла, совместно используемого несколькими библиотеками (.LIB , .DLL и .EXE) только в одной из них этот файл должен объявляться без атрибута EXTERNAL. Во всех других библиотеках и программах следует объявлять этот файл с атрибутом EXTERNAL. Это обеспечит уверенность в том, что для него распределен только один буфера записи и во всех библиотеках и программах при обращении к нему будут ссылки на одну и ту же область памяти.

Объявления файлов во всех библиотеках (или .EXE - модулях), которые ссылаются на эти общие файлы, должны содержать в точности одинаковые ключи, мемо и поля, объявленные точно в таком же порядке. Если объявления отличаются, то может произойти разрушение данных. Реальные последствия несовместимости объявлений файла зависят от драйвера для данной файловой системы. Ответственность за соблюдение идентичности объявлений лежит на программисте, поскольку ни компилятор не компоновщик не могут определить несоответствия объявлений в различных программах и библиотеках.

В структуре FILE с атрибутом EXTERNAL не должно быть атрибутов OWNER, NAME, or PASSWORD. Они должны быть в том объявлении файла, которое не имеет атрибута EXTERNAL.

Можно посоветовать при разработке больших систем, использующих много библиотек DLL и/или EXE модулей, которые совместно используют одни и те же файлы, собирать реальные описания разделяемых глобальных переменных и файлов в одну библиотеку DLL. Таким образом создается одна “главная” библиотека DLL, в которой происходит отслеживание всех действительных объявлений файлов. Эта главная библиотека связывается со всеми программами, которые используют общие файлы и переменные. Во всех других библиотеках и программах в этой системе общие переменные должны объявляться с атрибутами EXTERNAL и DLL.

### Пример:

```

PROGRAM
MAP
  MODULE('LIB.LIB')
    AddCount                !Внешняя библиотечная процедура
  ..
TotalCount LONG,EXTERNAL    !Переменная объявлена во внешней библиотеке
Cust       FILE,PRE(Cus),EXTERNAL('') !Структура файла определена в программном
модуле
CustKey    KEY('Cus:Name')    ! библиотека которого включается в эту программу
Record     RECORD
Name       STRING(20)

Contact    FILE,PRE(Con),EXTERNAL('LIB01')    !Структура файла определена в MEM-
BER модуле
ContactKey KEY('Con:Name')    ! библиотека которого включается в эту программу
Record     RECORD
Name       STRING(20)

!        файл LIB.CLW содержит:
PROGRAM
MAP
  MODULE('LIB01')
    AddCount                !
  ..
TotalCount LONG              !Объявление переменной TotalCount
Cust       FILE,PRE(Cus)     !Объявление файла Cust, для которого здесь
CustKey    KEY('Cus:Name')   !   же выделяется буфер записи
Record     RECORD
Name       STRING(20)
  ..
CODE
!Исполняемые операторы ...
!        файл LIB01.CLW содержит:
MEMBER('LIB')

```



Contact	FILE,PRE(Con)	!Объявление файла Contact, для которого здесь
ContactKey	KEY('Con:Name')	! же выделяется буфер записи
Record	RECORD	
Name	STRING(20)	
..		
AddCount	PROCEDURE	
	CODE	
	TotalCount += 1	

## DLL (процедура определена внешне, в библиотеке DLL)

**DLL( [ флаг ] )**

**DLL**                      Объявляет файл, определенный внешне, в библиотеке DLL.  
*флаг*                      Числовая константа, метка соответствия, или определение системы поддержки проекта, которое задает активен ли данный атрибут. Если флаг установлен в 0, то этот атрибут неактивен, как если бы его вообще не было. Если флаг имеет отличное от нуля значение, то атрибут активен.

Атрибут **DLL** указывает, что структура FILE, в которой имеется этот атрибут, определена в библиотеке с динамическими связями (DLL). Структура FILE, имеющая атрибут DLL, должна иметь и атрибут EXTERNAL. В 32-х разрядных приложений атрибут DLL обязателен, так как такие библиотеки являются настраиваемыми (перемещаемыми) в 32-битовом адресном пространстве, которое требует от компилятора еще одного дополнительного разыменовывания (преобразования адреса) при обращении к файлу.

Объявления файлов во всех библиотеках (или .EXE - модулях), которые ссылаются на общие файлы, должны быть в точности одинаковыми (с соответствующим добавлением атрибутов EXTERNAL и DLL). Если объявления отличаются, то может произойти разрушение данных. Ответственность за соблюдение идентичности объявлений лежит на программисте, поскольку ни компилятор не компоновщик не могут определить несоответствия объявлений в различных программах и библиотеках.

При использовании атрибутов EXTERNAL и DLL для объявления файла, совместно используемого несколькими библиотеками (.LIB , .DLL и .EXE) только в одной из них этот файл должен объявляться без атрибутов EXTERNAL и DLL. Во всех других библиотеках и программах следует объявлять этот файл с этими атрибутами. Это обеспечит уверенность в том, что для него распределен только один буфера записи и во всех библиотеках и программах при обращении к нему будут ссылки на одну и ту же область памяти.

Можно посоветовать при разработке больших систем, использующих много библиотек DLL и/или EXE модулей, которые совместно используют одни и те же файлы, собирать реальные описания разделяемых глобальных переменных и файлов в одну библиотеку DLL.

Таким образом создается одна “главная” библиотека DLL, в которой происходит отслеживание всех действительных объявлений файлов. Эта главная библиотека связывается со всеми программами, которые используют общие файлы и переменные. Во всех других библиотеках и программах в этой системе общие переменные должны объявляться с атрибутами EXTERNAL и DLL.

### Пример:

```
Cust      FILE,PRE(Cus),EXTERNAL(''),DLL(1)      !Файл определен в программном
модуле .DLL
CustKey   KEY('Cus:Name')
Record    RECORD
Name      STRING(20)
..
```

Смотри также:    EXTERNAL

## ОЕМ (установить поддержку международной кодировки)

### ОЕМ

Атрибут **ОЕМ** указывает, что данный файл содержит строковые данные не на английском языке. Эти строки автоматически перекодируются из кодировки OEM ASCII, в которой они содержатся в файле в набор символов ANSI, в котором они отображаются в Windows. Перед выводом на диск все строковые данные в записи файла автоматически перекодируются из кодировки ANSI в кодировку OEM ASCII. Конкретный набор символов OEM ASCII, используемый при перекодировании, берется из кодовой страницы DOS, загруженной драйвером country.SYS. Это “привязывает” файл данных к конкретному языку, использующему эту кодовую страницу и значит, что эти данные нельзя использовать на компьютере, на котором загружена другая кодовая страница. Этот атрибут может поддерживаться не всеми файловыми системами. За уточнением обращайтесь к документации по конкретному файловому драйверу.

### Пример:

```
Cust      FILE,DRIVER('TopSpeed'),PRE(Cus),OEM      !Содержит не английские строки
CustKey   KEY('Cus:Name')
Record    RECORD
Name      STRING(20)
..

Screen    WINDOW('Window')
           ENTRY(@S20),USE(Cus:Name)
           BUTTON('&OK'),USE(?Ok),DEFAULT
           BUTTON('&Cancel'),USE(?Cancel)
```

END

CODE

OPEN(Cust)

!Открыть файл Cust

SET(Cust)

NEXT(Cust)

!Прочитать запись, строки в кодировке ASCII  
!автоматически перекодируются в ANSI

OPEN(Screen)

!Открыть окно и вывести данные в коде ANSI

ACCEPT

CASE FIELD()

OF ?Ok

CASE EVENT()

OF EVENT:Accepted

PUT(Cust)

!Вывести запись, строки в кодировке ANSI  
!автоматически перекодируются в OEM ASCII  
!по загруженной в DOS кодовой странице

BREAK

END

END

END

CLOSE(Screen)

CLOSE(Cust)

## Операторы, описывающие структуру файла

### INDEX (объявить статический ключ доступа к записям файла)

метка	INDEX([-][поле],...,-[поле]) [,NAME()][,NOCASE][,OPT]
<b>INDEX</b>	Объявляет статический ключ доступа для файла данных
-/+	Знак “-” (минус) перед полем - компонентом индекса указывает убывающий порядок сортировки для этого компонента. Если минус опущен или указан символ + (плюс), то упорядочение по этой составляющей производится в порядке возрастания значений.
поле	Метка поля в структуре RECORD структуры FILE, в которой объявляется индекс. Поле представляет собой компонент индекса. В качестве составляющей индекса не может использоваться поле, объявленное с атрибутом DIM (массив).
<b>NAME</b>	Задаёт спецификацию дискового файла для индекса.
<b>OPT</b>	Исключает из индекса те записи, в которых все поля-компоненты имеют пустые (нулевые или пробельные) значения.
<b>NOCASE</b>	Указывает независимость сортировки индекса от регистра букв (прописные - строчные).

Оператор **INDEX** объявляет “статический” ключ в структуре FILE. Индексный файл строится и обновляется только посредством выполнения оператора BUILD. Индекс используется для доступа к записям данных в логической последовательности отличной от физического расположения их в файле. Он может использоваться или для последовательной обработки файла, или прямого (произвольного) доступа к записям. В индексе всегда допускается наличие дублирующихся значений. Индекс может иметь несколько составляющих полей. Порядок следования поле-компонент в индексе определяет порядок его упорядочения. Первый компонент наиболее значим, последний - наименее. В общем случае файл данных может иметь до 255 индексов (и/или ключей), а каждый индекс может состоять из полей, сумма длин которых не может превышать 255 байт, но точное значение этих величин зависит от конкретного файлового драйвера. Индекс, объявленный без составляющих его полей создает “динамический индекс. В динамическом индексе в качестве составляющих может использоваться любое поле (или поля) из структуры RECORD за исключением массивов. Поля - компоненты динамического индекса определяются во время выполнения программы вторым параметром оператора BUILD. Один и тот же динамический индекс можно строить, используя каждый раз различные поля-компоненты.

#### Пример:

```
Names FILE, DRIVER('Clarion'), PRE(Nam)
NameNdx INDEX(Nam:Name), NOCASE !Объявить индекс по полю Name
NbrNdx INDEX(Nam:Number), OPT !Объявить индекс по полю Number
```

Rec RECORD  
Name STRING(24)  
Number SHORT  
..

Смотри также: KEY, BUILD

## KEY (объявить динамический ключ доступа к записям файла)

*метка* **KEY**(**[-][поле]**,...,**[-][поле]**) [**DUP**][**NAME()**][**NOCASE**][**OPT**] [**PRIMARY**]

<b>KEY</b>	Объявляет динамически обновляемый ключ доступа к файлу данных.
<b>-/+</b>	Знак “-” (минус) перед полем - компонентом ключа указывает убывающий порядок сортировки для этого компонента. Если минус опущен или указан символ + (плюс), то упорядочение по этой составляющей производится в порядке возрастания значений.
<i>поле</i>	Метка поля в структуре RECORD структуры FILE, в которой объявляется ключ. Поле представляет собой компонент ключа. В качестве составляющей ключа не может использоваться поле, объявленное с атрибутом DIM (массив).
<b>NAME</b>	Задаёт спецификацию дискового файла для ключа.
<b>DUP</b>	Допускает наличие в файле записей, имеющих одинаковые значения полей, составляющих ключ.
<b>OPT</b>	Исключает из ключа те записи, в которых все поля-компоненты имеют пустые (нулевые или пробельные) значения.
<b>NOCASE</b>	Указывает независимость сортировки ключа от регистра букв (прописные - строчные).
<b>PRIMARY</b>	Указывает, что данный ключ является первичным ключом в связи между файлами (уникальным ключом содержащимся в каждой записи файла).

Ключ представляет собой указатель на записи данных, который автоматически обновляется во время добавления, удаления или изменения записей файла данных. Он используется для доступа к записям в логической последовательности, отличной от физического расположения записей в файле. Ключ может использоваться либо для последовательной обработки файла, либо прямого (произвольного) доступа к записям.

Ключ может содержать несколько составляющих его полей. Порядок следования поле-компонент в ключе определяет порядок его упорядочения. Первый компонент ключа самый старший с точки зрения сортировки данного ключа, а последний - самый младший. В общем случае файл данных может иметь до 255 ключей (и/или индексов), а каждый ключ может длиной до 255 байт, но точное значение этих величин зависит от конкретного файлового драйвера.

Пример:

```
Names  FILE,DRIVER('Clarion'),PRE(Nam)
NameKey  KEY(Nam:Name),NOCASE,DUP  !Объявить ключ по полю Name
NbrKey   KEY(Nam:Number),OPT      !Объявить ключ по полю Number
Rec      RECORD
Name     STRING(20)
Number   SHDRT
...
CODE
Nam:Name = 'Clarion Software'      !Присвоить значение ключевому полю
GET(Names,Nam:NameKey)            !Прочитать запись
SET(Nam:NbrKey)                   !Установить последовательную обработку по номеру
```

Смотри также: SET, GET, INDEX

МЕМО (объявить текстовое поле)

метка	МЕМО(длина)[,BINARY][,NAME()]
-------	-------------------------------

- МЕМО

Объявляет строку фиксированной длины, которая хранится на диске в виде строки переменной длины.
- длина

Числовая константа, которая определяет максимальное число символов в поле. Диапазон возможных значений - от 1 до 65536 байт в шестнадцатиразрядном приложении и неограничен в 32-х разрядном.
- BINARY

Объявляет, что МЕМО-поле предназначено для хранения двоичных данных.
- NAME

Задаёт имя файла, в котором хранится МЕМО-поле (использование этого атрибута зависит от файлового драйвера).

Оператор **МЕМО** объявляет строковое поле фиксированной длины, которое на диске хранится в виде записей переменной длины. Параметр длина определяет максимальный размер данных в этом поле. МЕМО-поле должно объявляться перед структурой RECORD. Память для буфера МЕМО-поля выделяется при открытии файла, а при закрытии она освобождается.

В общем случае в структуре FILE может объявляться до 255 МЕМО-полей, но точное их число и способ хранения на диске зависят от файлового драйвера. МЕМО-поле обычно отображается в поле типа TEXT структуры SCREEN или REPORT.

Пример:

```
Names  FILE,DRIVER('Clarion'),PRE(Nam)
NameKey  KEY(Nam:Name)
```

```
NbrKey    KEY(Nam:Number)
Notes     MEMO(4800)                !Мемо-поле длиной 4800 байт
Rec       RECORD
Name      STRING(24)
Number    SHORT
..
```

## BLOB (объявить мемо-поле переменной длины)

метка	BLOB [,BINARY] [,NAME( )]
-------	---------------------------

**BLOB**                      Объявляет мемо-поле переменной длины, которое может быть больше 64K (и в 16-ти разрядном, и в 32-х разрядном приложении).

**BINARY**                    Объявляет, что в этом поле хранятся двоичные данные.

**NAME**                      Указывает, имя дискового файла для этого BLOB-поля (использование этого параметра зависит от файлового драйвера).

Оператор **BLOB** (Binary Large Object - большой двоичный объект) объявляет строковое поле переменной длины, и которое может быть более чем 64K (и в 16-ти разрядном, и в 32-х разрядном приложении). Поле BLOB должно объявляться до структуры RECORD. Память для него выделяется и освобождается по мере необходимости. В общем случае в структуре FILE может быть объявлено до 255-ти полей типа BLOB. Точное число таких полей и способ их хранения на диске зависят от файлового драйвера.

Поле BLOB нельзя обрабатывать “целиком”; нужно использовать синтаксис “части строки” для обращения к одной части (до 64K) за раз. Такое поле нельзя использовать так же как обычную переменную (нельзя указать ее в качестве USE-переменной экранного объекта). Можно использовать свойство PROP:Handle, чтобы получить идентификационный номер (handle) Windows для объекта BLOB. Таким образом обеспечивается единственный способ присвоения значения одного поля BLOB другому: получить для обоих поле идентификационные номера объектов Windows, а затем присвоить один номер другому. Для текущей прочитанной в память записи длину поля BLOB можно получить с помощью функции SIZE. Кроме того, можно получить (и установить) размер BLOB-поля с помощью свойства PROP:BlobSize.

### Пример:

```
Names     FILE,DRIVER('TopSpeed')
NbrKey    KEY(Names:Name)
Notes     BLOB                      !Может быть больше 64K
Rec       RECORD
Name      STRING(20)
..
ArcNames  FILE,DRIVER('TopSpeed')
NbrKey    KEY(ArcNames:Name)
```

Notes	BLOB
Rec	RECORD
Name	STRING(20)

```
..
CODE
OPEN(Names)
CREATE(ArcNames)
SET(Names)
LOOP
  NEXT(Names)
  IF ERRORCODE() THEN BREAK.
  ArcNames.Rec = Names.Rec !Занести данные в запись архивного файла
  ArcNames.Notes{PROP:Handle} = Names.Notes{PROP:Handle}
                                !Занести данные в BLOB поле файла Archive
  ADD(ArcNames)
END
```



**RECORD (объявить структуру записи файла)**

<i>метка</i>	<b>RECORD</b> [,PRE][,NAME())
<i>поля</i>	
.	

<b>RECORD</b>	Объявляет начало структуры данных внутри объявления файла.
<i>поля</i>	Множественное объявление переменных.
<b>PRE</b>	Указывает префикс для переменных в данной структуре.
<b>NAME</b>	Задаёт внешнее имя для данной структуры RECORD (использование этого параметра зависит от файлового драйвера).

Оператор **RECORD** объявляет начало структуры данных в объявлении файла. Структура RECORD обязательно должна присутствовать. Каждое поле является элементом структуры RECORD. Длина структуры RECORD представляет собой сумму длин составляющих ее полей. Когда метка структуры RECORD используется в операторе присвоения, выражении или списке параметров, то эта структура рассматривается как группа (тип данных GROUP).

Во время выполнения программы в качестве буфера для структуры RECORD выделяется статическая память. Поля в буфере записи доступны независимо от того, открыт файл или закрыт.

Если поля представляют собой объявления переменных с присвоением начального значения, то эти начальные значения используются для определения размера переменных, а в буфер записи эти значения не заносятся.

Операторами NEXT, PREVIOUS или GET записи из файла данных на диске считываются в буфер записи. Данные в полях обрабатываются, а затем как единая запись заносятся операторами ADD или PUT в файл данных на диске или удаляются из него оператором DELETE.

**Пример:**

Names	FILE,DRIVER('Clarion')	!Объявить структуру файла
Record	RECORD	! начало объявления структуры записи
Name	STRING(20)	! объявить поле Name
Number	SHORT	! объявить поле Number
..	!Конец объявления структуры записи и файла	

## Атрибуты операторов INDEX, KEY и MEMO

### BINARY ( мемо-поле содержит двоичные данные)

#### BINARY

Атрибут BINARY в объявлении MEMO-поля указывает, что поле будет содержать данные, которые являются не только символами ASCII. Обычно этот атрибут используется для того, чтобы хранить в MEMO-поле небольшие графические изображения, предназначенные для отображения в полях типа IMAGE структуры SCREEN.

#### Пример:

```
Names  FILE,DRIVER('Clarion'),PRE(Nam)
NameKey KEY(Nam:Name)
NbrKey  KEY(Nam:Number)
Picture MEMO(48000),BINARY      !Двоичное мемо-поле длиной 48000 байт
Rec     RECORD
Name    STRING(20)
Number  SHORT
..
```

Смотри также: MEMO, IMAGE

### DUP (допускается повторение значений ключа)

#### DUP

Атрибут DUP в объявлении ключа допускает наличие в файле нескольких записей с одинаковым значением ключа. Если же атрибут DUP опущен, то попытка добавить запись (или изменить данные в существующей записи) со значением ключа, уже имеющимся в файле, приведет к выдаче сообщения об ошибке “Creates Duplicate Key”, а запись в файл не запишется. Во время последовательной обработки записей с использованием последовательности ключа записи с одинаковым значением ключа обрабатываются в том порядке, в котором соответствующие им ключи располагаются в файле ключей. Операторы SET и GET обращаются к первой записи из набора имеющих одинаковый ключ. В объявлении индексов атрибут DUP не нужен, поскольку индекс всегда допускает повторение значения элементов.

#### Пример:

```
Names  FILE,DRIVER('Clarion'),PRE(Nam)
NameKey KEY(Nam:Name),DUP      !Объявить ключ по полю имя, допустить
совпадающие имена
```

```

NbrKey  KEY(Nam:Number)          !Объявить ключ по полю номер, повтор номеров
недопустим
Rec     RECORD
Name    STRING(20)
Number  SHORT
..

```

## **NOCASE (независимость ключа или индекса от регистра букв)**

### **NOCASE**

Атрибут **NOCASE** в объявлении ключа или индекса делает независимой от регистра букв последовательность упорядочения ключа или индекса. Все буквы значения ключа при записи в файл ключей преобразуются в прописные. Это преобразование не влияет на регистр букв в полях записи в файле данных. Не влияет оно и на символы, не являющиеся буквами.

#### **Пример:**

```

Names  FILE,DRIVER('Clarion'),PRE(Nam)
NameKey KEY(Nam:Name),NOCASE  !Объявить ключ по полю имя, допустить
совпадающие имена
NbrKey  KEY(Nam:Number)          !Объявить ключ по полю номер
Rec     RECORD
Name    STRING(24)
Number  SHORT
..

```

**Смотри также:** INDEX, KEY

## **OPT (исключить нулевые значения ключа или индекса)**

### **OPT**

Атрибут **OPT** позволяет исключить из индекса или ключа элементы для тех записей, в которых все поля, составляющие ключ или индекс, имеют “пустое” значение. В контексте использования этого атрибута “пустым” значением является нуль для числовых полей и заполненное пробелами строковое поле.

#### **Пример:**

```

Names  FILE,DRIVER('Clarion'),PRE(Nam)  !Объявить структуру FILE
NameKey KEY(Nam:Name),OPT  !Объявить ключ по полю имя, не допуская пустых имен
NbrKey  KEY(Nam:Number),OPT  !Объявить ключ по номеру, не допуская нулевых номеров
Rec     RECORD

```

Name     STRING(20)  
 Number    SHORT

..

Смотри также: INDEX, KEY

## PRIMARY (установить первичный ключ)

### PRIMARY

Атрибут **PRIMARY** указывает, что данный ключ является уникальным ключом содержащимся в каждой записи файла и для него недопустимы “пустые” значения любого из составляющих его полей. Это определение “по Кодду” первичного ключа в теории реляционных баз данных.

#### Пример:

```
Names      FILE,DRIVER('TopSpeed'),PRE(Nam)      !Объявить структуру файла
NameKey     KEY(Nam:Name),OPT
            !Объявить ключ Name, пробельные значения недопустимы
NbrKey      KEY(Nam:Number),OPT,PRIMARY
            !Объявить ключ Number нулевые значения недопустимы
Rec         RECORD
Name        STRING(20)
Number      SHORT
..
```

Смотри также:     KEY

## NAME (задать внешнее имя)

NAME([*константа*]  
       |*переменная*])

**NAME**                    Задает “внешнее” имя.  
*константа*                Строковая константа.  
*переменная*               Метка статической строковой переменной. Переменная может быть объявлена как глобальная, в секции данных модуля или локальных данных, но с атрибутом STATIC.

Атрибут **NAME** указывает в операторе KEY, INDEX или MEMO “внешнее” имя файла ключей, индексов или тето-полей для данного файлового драйвера. Для некоторых файловых драйверов требуется, чтобы ключи, индексы и тето-поля находились в отдельных файлах, которые и указываются атрибутом NAME.

Атрибут в форме NAME(константа) может использоваться для любого поля, объявленного в структуре RECORD. Таким образом обеспечивается для файлового драйвера такое имя поля, какое можно использовать в файловой системе данного драйвера.

По умолчанию, атрибут NAME без параметров принимает значение метки оператора объявления структуры, в которой он помещается (включая любой заданный префикс).

**Пример:**

```
Cust      FILE,PRE(Cus),NAME(CustName) !Имя файла в переменной CustName
CustKey   KEY('Name'),NAME('c:\data\cust.idx') !Объявить ключ в файле cust.idx
Record    RECORD
Name      STRING(20) !Значение атрибута NAME по умолчанию равно 'Cus:Name'
..
```

**Смотри также:** FILE, KEY, INDEX

## Команды для работы с файлами

### BUILD (построить ключ или индекс)

BUILD(	<div>ключ</div> <div>индекс[,составляющие [,фильтр ]])</div> <div>файл</div>
<b>BUILD</b>	Построить ключи или индексы.
<i>ключ</i>	Метка объявления ключа.
<i>индекс</i>	Метка объявления индекса.
<i>файл</i>	Метка объявления файла.
<i>составляющие</i>	Строковая константа или переменная, содержащая список полей, которым строится динамический индекс. Если файл имеет атрибут CREATE, то в качестве составляющих можно использовать метки объявления полей записи. Если же у файла нет атрибута CREATE, то в качестве составляющих должны использоваться значения атрибутов NAME соответствующих полей. Имена полей должны разделяться запятыми. Перед именем поля ставится символ “+” или “-” для того, чтобы обозначить возрастающую или убывающую последовательность сортировки.
<i>фильтр</i>	Строковая константа, переменная или выражение, содержащее логическое выражение, с помощью которого отфильтровываются ненужные записи из динамического индекса. Для этого нужно, чтобы были компонентам индекса были присвоены имена. Затем нужно оператором BIND сделать возможным использование этих переменных в выражении фильтра.

Оператор **BUILD** строит ключи и индексы. В формах: BUILD(ключ), BUILD(индекс) и BUILD(файл) он требует исключительного доступа к файлу. Поэтому должен быть закрыт, заблокирован оператором LOCK или открываться с режимом доступа 12h (чтение/запись, полный запрет) или 22h (чтение/запись, запрет записи). В форме BUILD(индекс, составляющие) этот оператор не требует исключительного доступа к файлу.

**BUILD(ключ)/BUILD(индекс)** Строится только заданный ключ или индекс. Файл должен быть закрыт, или открыт и заблокирован оператором LOCK, или открыт с режимом доступа 12h или 22h.

**BUILD(файл)** Строятся все ключи, объявленные для данного файла. Файл должен быть закрыт, или открыт и заблокирован оператором LOCK, или открыт с режимом доступа 12h или 22h.

**BUILD(индекс, составляющие)** Позволяет построить динамический индекс. Эта форма оператора не требует исключительного доступа к файлу, однако файл должен быть открыт (в любом допустимом режиме доступа). Динамический индекс создается в виде временного файла и исключительно для того

пользователя, который выдал команду BUILD. Во время закрытия файла временный файл индекса автоматически удаляется..

**BUILD**(индекс, составляющие, фильтр)      Позволяет построить динамический индекс, содержащий только записи, которые удовлетворяют критерию фильтра. Фильтр должен иметь форму, поддерживаемую файловым драйвером.

Выдаваемые сообщения об ошибках:

- 37    File Not Open (файл не открыт)
- 40    Creates Duplicate Key (создается второй экземпляр ключа)
- 63    Exclusive Access Required (требуется исключительный доступ)
- 76    Invalid Index String (ошибка в строке составляющих индекса)

**Пример:**

```
Names    FILE,DRIVER('TopSpeed '),PRE(Nam)      !Объявить структуру FILE
NameKey   KEY(Nam:Name),OPT                    !Объявить ключ по полю "имя"
NbrNdx   INDEX(Nam:Number),OPT                !Объявить индекс по полю "номер"
DynNdx   INDEX()                                !Объявить динамический индекс
Rec       RECORD
Name       STRING(24)
Number      SHORT
...
CODE
OPEN(Names,12h)                                !Открыть файл в режиме исключительного доступа
BUILD(Names)                                    !Построить все ключи для файла
BUILD(Nam:NbrNdx)                               !Построить индекс по полю "номер"
BUILD(Nam:DynNdx,'-Nam:Number,+Nam:Name')
          ! Построить динамический индекс по убыванию
          !номеров и возрастанию имен
BIND('Nam:Name',Nam:Name)
BUILD(Nam:DynNdx,'+Nam:Name','UPPER(Nam:Name[1]) = A')
          !Построить динамический индекс имен, начинающ. с A
UNBIND('Nam:Name')
```

**Смотри также:** OPEN, SHARE

## CLOSE (закрыть файл данных)

**CLOSE**(файл)

**CLOSE**            Закрывает файл данных  
*файл*              Метка оператора FILE.

Оператор CLOSE закрывает файл. Обычно при этом буферы DOS для этого файла записываются на диск и освобождается вся использованная при работе с файлом память

(за исключением буфера данных в структуре RECORD). Конкретные действия, выполняемые оператором CLOSE, зависят от используемого файлового драйвера.

Пример:

CLOSE(Customer)

!Заккрыть файл клиентов

**COPY** (скопировать файл данных)

**COPY**(*файл, новый файл*)

**COPY**

Копирует файл данных

*файл*

Метка оператора FILE, описывающего файл, который подлежит скопировать.

новый файл

Строковая константа или переменная, содержащая спецификацию файла в файловой системе DOS. Если эта спецификация не содержит имени диска и пути к файлу, то подразумевается текущий диск и текущий каталог. Если указан только путь к каталогу нового файла, то для нового файла используются имя и расширение исходного файла.

Оператор **COPY** копирует файл и добавляет спецификацию нового файла в файловую систему DOS. Файл который подлежит копированию должен быть закрыт, иначе выдается сообщение об ошибке “File Already Open” (файл уже открыт). Если спецификация нового файла совпадает со спецификацией исходного, то оператор COPY игнорируется.

Так как некоторые файловые драйверы используют для реализации одной структуры FILE несколько физических дисковых файлов, подразумеваемые по умолчанию имена и расширения этих файлов зависят от файлового драйвера. Если выдается какое-либо сообщение об ошибке, то файл не копируется.

### Выдаваемые сообщения об ошибках:

02 File Not Found (файл не найден)

03 Path Not Found (путь к файлу неверен)

05 Access Denied (доступ к файлу запрещен)

52 File Already Open (файл уже открыт)

### Пример:

```
COPY(Names,'A:\')
```

!Скопировать файл Names на дискету

COPY(CompText,Filename)

## !Скопировать один текстовый файл в другой



**CREATE (создать пустой файл данных)****CREATE**(*файл*)**CREATE**

Создает пустой файл данных.

*файл*

Метка структуры FILE, описывающей файл, который должен быть создан.

Оператор **CREATE** добавляет пустой файл в файловую структуру DOS. Если файл уже существует, то он удаляется и создается пустой файл. В этом случае существующий файл должен быть закрыт, иначе выдается сообщение об ошибке “File Already Open” (файл уже открыт). Оператор **CREATE** не открывает файл после создания.

Выдаваемые сообщения об ошибках:

- 03 Path Not Found (путь к файлу неверен)
- 04 Too Many Open Files (слишком много открытых файлов)
- 05 Access Denied (доступ к файлу запрещен)
- 52 File Already Open (файл уже открыт)
- 54 No Create Attribute (файл не имеет атрибута CREATE)

**Пример:**

CREATE(Master)

!Создать главный файл

CREATE(Detail)

!Создать дополнительный файл

**EMPTY (очистить файл данных)****EMPTY**(*файл*)**EMPTY**

Очищает файл данных

*файл*

Метка структуры FILE.

Оператор **EMPTY** удаляет из заданного файла все записи. Этот оператор требует исключительного доступа к файлу. Поэтому файл должен быть открыт в режиме доступа 12h (Чтение/Запись, Полный запрет) или 22h (Чтение/Запись, Запрет записи).

Выдаваемые сообщения об ошибках:

- 63 Exclusive Access Required (требуется исключительный доступ)

**Пример:**

OPEN(Master,18)

!Открыть главный файл

EMPTY(Master)

! и очистить его



ожидать до тех пор, пока другой пользователь не разблокирует его.

**LOCK(файл,секунд)** После безуспешных попыток в течение заданного числа секунд заблокировать файл выдает сообщение “File Already Locked” (файл уже заблокирован)

Наиболее часто встречающаяся проблема, которую следует избегать при блокировании файлов - взаимная блокировка. Эта ситуация возникает, когда две рабочие станции, используя оператор **LOCK** в форме **LOCK(файл)**, пытаются заблокировать один и тот же набор файлов, но в различной последовательности. Одна рабочая станция уже заблокировала файл, который другая пытается заблокировать в данный момент и наоборот. Избежать этой проблемы можно используя вторую форму оператора **LOCK**: **LOCK(файл,секунд)** и блокируя файлы всегда в одном и том же порядке.

#### Выдаваемые сообщения об ошибках:

32 File Already Locked (файл уже заблокирован)

#### **Пример:**

```

LOOP !Цикл для исключения взаимной блокировки
LOCK(Master,1)      !В течение 1 секунды пытаться заблокировать файл Master
IF ERRORCODE() = 32  !Если кто-то уже сделал это
    CYCLE            ! попытаться еще раз
END
LOCK(Detail,1)      !В течение 1 секунды пытаться заблокировать файл Detail
IF ERRORCODE() = 32  !Если кто-то уже сделал это
    UNLOCK(Master)   ! разблокировать файл Master
    CYCLE            ! попытаться повторить все сначала
..                  !Конец структуры IF и цикла

```

## **OPEN (открыть файл данных)**

**OPEN(файл[,режим доступа])**

<b>OPEN</b>	Открывает файл данных.
<i>файл</i>	Метка структуры FILE
<i>режим доступа</i>	Числовая константа, переменная или выражение, которая определяет уровень доступа, обуславливаемый и для пользователя, открывающего файл, и для других пользователей в многопользовательской среде. По умолчанию устанавливается значение 22h (чтение/запись + запрет записи).

Оператор **OPEN** открывает структуру FILE для работы и устанавливает режим доступа. Поддержка различных режимов доступа зависит от файлового драйвера. Перед тем, как начать обращение к файлу, его надо открыть явно. Режим доступа представляет собой битовую комбинацию, сообщаящую операционной системе, какой уровень доступа

назначить пользователю, открывающему файл, и на каком уровне установить другим пользователям ограничение в использовании файла. Имеются следующие фактические значения для каждого уровня доступа.

	Десятич.	Шестнадц.	Доступ
Доступ Пользователя	0	0h	Только Чтение
	1	1h	Только Запись
	2	2h	Чтение/Запись
Доступ других пользователей	16	10h	Полный Запрет
	32	20h	Запрет Записи
	48	30h	Запрет Чтения
	64	40h	Нет Запрета

#### Выдаваемые сообщения об ошибках:

- 02 File Not Found (файл не найден)
- 04 Too Many Open Files (слишком много открытых файлов)
- 05 Access Denied (доступ к файлу запрещен)
- 52 File Already Open (файл уже открыт)
- 75 Invalid Field Type Descriptor (неправильный описатель типа поля)

#### **Пример:**

```

ReadOnly EQUATE(0)      !Метки соответствия режимов доступа
WriteOnly EQUATE(1)
ReadWrite EQUATE(2)
DenyAll EQUATE(10h)
DenyWrite EQUATE(24h)
DenyRead EQUATE(30h)
DenyNone EQUATE(40h)
CODE
    OPEN(Names,ReadWrite+DenyNone)
    !Открыть в режиме полного совместного использования

```

**Смотри также:** SHARE

## **RACK (исключить удаленные записи)**

**RACK(файл)**

**RACK** Исключает удаленные записи  
*файл* Метка структуры FILE.

Оператор **RACK** исключает логически удаленные записи из файла данных и заново создает ключи для него. Получающийся в результате выполнения этого оператора файл

является максимально компактным. Для своего выполнения оператор RACK требует дискового пространства, превышающего по крайней мере вдвое объем, занимаемый файлом данных, ключами и MEMO-файлами. Из старого файла создается новый, а старый удаляется только после завершения процесса. Этот оператор требует исключительного доступа к файлу. Поэтому файл должен быть открыт в режиме доступа 12h (чтение/запись, полный запрет) или 22h (чтение/запись, запрет записи).

Выдаваемые сообщения об ошибках:

63 Exclusive Access Required (требуется исключительный доступ)

**Пример:**

OPEN(Trans, 12h)      !Открыть файл в режиме исключительного доступа  
RACK(Trans)          ! и упаковать его

**Смотри также:** OPEN, SHARE

## **REMOVE (удалить файл данных)**

**REMOVE**(*файл*)

**REMOVE**      Удаляет файл данных  
*файл*          Метка структуры FILE.

Оператор **REMOVE** удаляет спецификацию файла из каталога DOS точно таким же образом как это делает команда DELETE операционной системы. Файл должен быть закрыт, в противном случае выдается сообщение об ошибке “File Already Open”. Если выдается какое-либо сообщение об ошибке, то файл не удаляется.

Выдаваемые сообщения об ошибках:

02 File Not Found (файл не найден)  
05 Access Denied (доступ к файлу запрещен)  
52 File Already Open (файл уже открыт)

**Пример:**

REMOVE(OldFile)                      !Удалить старый файл  
REMOVE(Changes)                    !Удалить файл изменений

**RENAME (изменить имя файла и/или каталог)****RENAME**(*файл, новый файл*)

**RENAME**                      Переименовывает файл.  
*файл*                              Метка структуры FILE, описывающей файл, который должен быть переименован.  
*новый файл*                      Строковая константа или переменная, содержащая спецификацию файла.

Если эта спецификация файла не содержит диска и пути, то подразумевается текущий диск и каталог. Если указывается только путь к файлу, то в качестве имени и расширения для нового файла используется имя и расширение исходного файла. Файл нельзя “переименовать” на диск, отличный от старого.

Оператор RENAME изменяет спецификацию файла в файловой системе DOS на заданную параметром новый файл. Файл, который переименовывается должен быть закрыт, в противном случае выдается сообщение “File Already Open”. Если спецификация нового файла совпадает с исходной спецификацией файла, то оператор RENAME игнорируется. Если выдается какое-либо сообщение об ошибке, то файл не переименовывается.

Поскольку некоторые файловые драйверы используют для одной логической структуры FILE несколько физических дисковых файлов, используемые по умолчанию имена файлов и расширения зависят от конкретного файлового драйвера.

Выдаваемые сообщения об ошибках:

- 02    File Not Found (файл не найден)
- 03    Path Not Found (путь к файлу неверен)
- 05    Access Denied (доступ к файлу запрещен)
- 52    File Already Open (файл уже открыт)

**Пример:**

RENAME(Text,'text.bak')	!Сделать резервную копию
RENAME(Master,'\\newdir')	!Переместить в другой каталог

**SHARE (открыть файл данных)****SHARE**(*файл[,режим доступа]*)

**SHARE**                      Открывает структуру данных для обработки.  
*файл*                              Метка структуры FILE  
*режим доступа*                      Числовая константа, переменная или выражение, которая определяет уровень доступа, обуславливаемый и для пользователя, открывающего

файл, и для других пользователей в многопользовательской среде. По умолчанию устанавливается значение 42h (чтение/запись + нет запрета).

Оператор **SHARE** открывает структуру FILE для работы и устанавливает режим доступа. Он аналогичен оператору OPEN за исключением режима доступа, используемого по умолчанию. Режим доступа представляет собой битовую комбинацию, сообщаемую операционной системе, какой уровень доступа назначить пользователю, открывающему файл, и на каком уровне установить другим пользователям ограничение в использовании файла. Имеются следующие фактические значения для каждого уровня доступа.

	Десятич.	Шестнадц.	Доступ
Доступ Пользователя	0	0h	Только Чтение
	1	1h	Только Запись
	2	2h	Чтение/Запись
Доступ других пользователей	16	10h	Полный Запрет
	32	20h	Запрет Записи
	48	30h	Запрет Чтения
	64	40h	Нет Запрета

#### Выдаваемые сообщения об ошибках:

- 02 File Not Found (файл не найден)
- 04 Too Many Open Files (слишком много открытых файлов)
- 05 Access Denied (доступ к файлу запрещен)
- 52 File Already Open (файл уже открыт)
- 75 Invalid Field Type Descriptor (неправильный описатель типа поля)

#### **Пример:**

```

ReadOnly EQUATE(0)
WriteOnly EQUATE(1)
ReadWrite EQUATE(2)
DenyAll EQUATE(10h)
DenyWrite EQUATE(24h)
DenyRead EQUATE(30h)
DenyNone EQUATE(40h)
CODE
SHARE(Master,ReadOnly+DenyWrite) !Метки соответствия режимов доступа
!Открыть в режиме только чтения

```

**Смотри также:** OPEN

**STATUS (получить состояние файла)****STATUS( *файл* )**

**STATUS** Возвращает текущее состояние файла  
*файл* Метка оператора FILE.

Если файл не открыт, функция STATUS возвращает ноль (0), а если открыт, то она возвращает режим доступа к файлу. Если режим доступа на самом деле ноль (Только чтение/доступ всем), то возвращается 40h (только чтение/нет запрета) см. оператор OPEN.

Тип возвращаемого значения: LONG

**Пример:**

```
IF STATUS(DataFile) % 16 = 0      !Открыто в режиме "только чтение"?
RETURN                          ! выйти
ELSE                              !Иначе
EXECUTE DiskAction              ! Записать запись на диск
ADD(DataFile)
PUT(DataFile)
DELETE(DataFile)
..
```

Смотри также: OPEN

**STREAM (включить буферизацию)****STREAM( *файл* )**

**STREAM** Выключает автоматический сброс буферов на диск после каждой операции.  
*файл* Метка структуры FILE.

Некоторые файловые системы записывают буферы DOS при каждой операции записи в файл. Оператор STREAM выключает этот механизм автоматической записи буферов. Буферы DOS выделяются оператором BUFFERS= в файле CONFIG.SYS . До тех пор, пока есть свободное место, записываемые данные хранятся в буферах, затем записываются на диск все буферы разом. Элементы каталога для данного файла обновляются только при записи буферов на диск. Буферизация выключается при закрытии файла, когда буферы автоматически записываются на диск, или при выполнении оператора FLUSH.

Поддержка этого оператора зависит от файловой системы и описывается в документации по соответствующему файловому драйверу.



## Смотри также: FLUSH

UNLOCK(*файл*)

<b>UNLOCK</b> <i>файл</i>	Разблокирует ранее заблокированный файл данных Метка структуры FILE.
------------------------------	---

Оператор **UNLOCK** разблокирует ранее заблокированный файл данных. Если файл не заблокирован, или заблокирован другим пользователем, то оператор UNLOCK игнорируется. При выполнении этого оператора не генерируется никаких сообщений об ошибках. Конкретные действия, выполняемые оператором зависят от файлового драйвера.

```

LOOP                !Цикл для исключения взаимной блокировки
LOCK(Master,1)      !В течение 1 секунды пытаться заблокировать файл Master
IF ERRORCODE() = 32  !Если кто-то уже сделал это
    CYCLE            ! попытаться еще раз
END
LOCK(Detail,1)      !В течение 1 секунды пытаться заблокировать файл Detail
IF ERRORCODE() = 32  !Если кто-то уже сделал это
    UNLOCK(Master)   ! разблокировать файл Master
    CYCLE             ! попытаться повторить все сначала
...                  !Конец структуры IF и цикла

```

## Команды для работы с записями

### ADD (добавить новую запись)

**ADD(файл[,длина])**

<b>ADD</b>	Добавляет новую запись в файл
<i>файл</i>	Метка структуры FILE, описывающей файл.
<i>длина</i>	Целочисленная константа, переменная или выражение, которая содержит число байтов, которые следует записать в файл. Значение этого параметра должно быть больше 0 и меньше или равно длине структуры RECORD. В случае, если параметр <i>длина</i> опущен или выходит за границы допустимого диапазона, он устанавливается равным длине структуры RECORD.

Оператор **ADD** записывает новую запись из буфера структуры RECORD в файл данных. При каждом выполнении оператора ADD также изменяются все ключи, относящиеся к этому файлу данных. Если сгенерировано какое-либо сообщение об ошибке, то запись в файл не добавляется. Конкретные действия, выполняемые оператором ADD, зависят от файлового драйвера.

В случае отсутствия свободного пространства на диске генерируется сообщение об ошибке “Access Denied” (доступ запрещен).

#### Выдаваемые сообщения об ошибках:

- 05 Access Denied (доступ к файлу запрещен)
- 37 File Not Open (файл не открыт)
- 40 Creates Duplicate Key (создается второй экземпляр ключа)

#### Пример:

```
ADD(Customer)                                !Добавить запись о новом клиенте
IF ERRORCODE() THEN STOP(ERROR()).          ! и проверить безошибочность добавления
```

### APPEND (добавить новую запись)

**APPEND(файл[,длина])**

<b>APPEND</b>	Добавляет новую запись к концу файла.
<i>файл</i>	Метка структуры FILE, описывающей файл.
<i>длина</i>	Целочисленная константа, переменная или выражение, которая содержит число байтов, которые следует записать в файл. Значение этого параметра должно быть больше 0 и меньше или равно длине структуры RECORD. В случае, если параметр <i>длина</i> опущен или выходит за границы допустимого

диапазона, он устанавливается равным длине структуры RECORD.

Оператор **APPEND** записывает новую запись из буфера структуры RECORD в файл данных. Во время его выполнения ключи относящиеся к этому файлу не обновляются. После добавления записей оператором APPEND ключи нужно перестроить оператором BUILD. Обычно оператор APPEND используется для “пакетного” добавления за один раз целого ряда записей.

Если сгенерировано какое-либо сообщение об ошибке, то запись в файл не добавляется. Конкретные действия, выполняемые оператором ADD, зависят от файлового драйвера. В случае отсутствия свободного пространства на диске генерируется сообщение об ошибке “Access Denied” (доступ запрещен).

#### Выдаваемые сообщения об ошибках:

- 05 Access Denied (доступ к файлу запрещен)
- 37 File Not Open (файл не открыт)

#### **Пример:**

LOOP UNTIL EOF(InFile)	!Цикл по обработке входного файла
NEXT(InFile)	! чтение каждой записи
IF ERRORCODE() THEN STOP(ERROR()).	! и проверка ошибок
Cus:Record = Inf:Record	!Копировать данные в запись о клиенте
APPEND(Customer)	! и добавить запись
IF ERRORCODE() THEN STOP(ERRDR()).	! проверить ошибки
.	!Конец цикла
BUILD(Customer)	!Построить ключи

**Смотри также:** BUILD

## **DELETE (удалить запись)**

### **DELETE(файл)**

**DELETE** Удаляет запись файла  
*файл* Метка структуры FILE, описывающей файл.

Оператор DELETE удаляет запись, к которой было последнее обращение оператором NEXT, PREVIOUS, GET, ADD или PUT. Также удаляются ключи для этой записи. Оператор DELETE не очищает буфер структуры RECORD. Поэтому данные из только что удаленной записи продолжают существовать и доступны до тех пор, пока в буфер записи не будут записаны новые данные.

Если предварительной выборки записи не было, или она заблокирована другой рабочей

станцией, то выдается сообщение “Record Not Available”, а запись не удаляется.

Выдаваемые сообщения об ошибках:

- 05 Access Denied (доступ к файлу запрещен)
- 37 Record Not Available (запись недоступна))

**Пример:**

```
Customer FILE, DRIVER('Clarion'), PRE(Cus)
NameKey KEY(Cus:Name), OPT
NbrKey KEY(Cus:Number), OPT
Rec RECORD
Name STRING(20)
Number SHORT

..
CODE
Cus:Number = 12345 !Присвоить значение ключевому полю
GET(Customer, Cus:NbrKey) !Получить запись с заданным ключом
IF ERRORCODE() THEN STOP(ERROR()).
DELETE(Customer) !Удалить запись о клиенте
```

**Смотри также:** ADD, GET, HOLD, NEXT, PREVIOUS, PUT

**GET (прочитать запись с помощью прямого доступа)**

*файл, ключ*

**GET**(*файл, указатель[длина]*)

*ключ, указатель ключа*

<b>GET</b>	Считывает запись из файла.
<i>файл</i>	Метка структуры FILE, описывающей файл.
<i>ключ</i>	Метка оператора объявления ключа или индекса.
<i>указатель</i>	Числовая константа, переменная или выражение, представляющая значение, возвращаемое для данной записи функцией POINTER(файл). Конкретное значение зависит от файлового драйвера.
<i>указатель ключа</i>	Числовая константа, переменная или выражение, представляющая значение, возвращаемое для данной записи функцией POINTER(ключ). Конкретное значение зависит от файлового драйвера.
<i>длина</i>	Целочисленная константа, переменная или выражение, содержащее число байт, которые следует прочитать из файла. Длина должна быть больше нуля и не больше длины структуры RECORD. Если этот параметр опущен или выходит за границы допустимого диапазона, то он устанавливается равным длине структуры RECORD.

Оператор **GET** находит конкретную запись в файле данных и считывает ее в буфер данных структуры RECORD. Прямой доступ к определенной записи осуществляется по относительному положению ее внутри файла или соответствию значения ключа.

**GET(файл,ключ)** Получает первую запись (из той последовательности, в которой они располагаются в файле ключей), которая содержит значения полей, совпадающие со значениями составляющих ключа.

**GET(файл, указатель[,длина])** Получает запись, основываясь на заданном указателе относительном положении записи внутри файла данных. Если указатель равен нулю, то текущее значение указателя на запись обнуляется, а из файла ничего не считывается.

**GET(ключ, указатель ключа)** Получает запись, основываясь на заданном указателе ключа относительном положении значения ключа в файле ключей.

Значения указателя и указателя ключа зависят от файлового драйвера. Это могут быть: номер записи, байтовое смещение в файле или какая-либо другая “поисковая позиция” внутри файла. Если значение указателя или указателя ключа выходит за границы допустимого диапазона, или в файле нет соответствующего значения ключа, то выдается сообщение “Record Not Found” (запись не найдена).

#### Выдаваемые сообщения об ошибках:

- 35 Record Not Found (запись не найдена)
- 37 File Not Open (файл не открыт)
- 43 Record Is Already Held (запись уже заблокирована)

#### **Пример:**

Customer FILE,DRIVER('Clarion'),PRE(Cus)

NameKey KEY(Cus:Name),OPT

NbrKey KEY(Cus:Number),OPT

Rec RECORD

Name STRING(20)

Number SHORT

..  
CODE

```
Cus:Name = 'Clarion'           !Присвоить значение ключевому полю
GET(Customer,Cus:NameKey)      ! получить запись с соответствующим значением
IF ERRORCODE() THEN STOP(ERROR()).
GET(Customer,3)                !Получить 3-ю запись в физической последовательности
IF ERRORCODE() THEN STOP(ERROR()).
GET(Cus:NameKey,3)             !Получить 3-ю запись в последовательности ключа
IF ERRORCODE() THEN STOP(ERROR()).
```

**Смотри также:** POINTER, DUPLICATE

**HOLD (исключительный доступ к записи)****HOLD(файл[,секунд])****HOLD**

Включает блокировку записи.

*файл*

Метка структуры FILE, открытой в режиме совместного доступа.

*секунд*

Числовая константа или переменная, которая задает максимальное время ожидания в секундах.

Оператор **HOLD** включает блокировку записи последующими операторами GET, NEXT и PREVIOUS при работе в многопользовательской среде. При успешном получении записи операторы GET, NEXT и PREVIOUS отмечают эту запись как “захваченную”. В общем случае это запрещает другим пользователям изменять эту запись, но чтение ее возможно. Конкретные действия, выполняемые оператором HOLD, зависят от файлового драйвера.

**HOLD(файл)**

Включает процесс безусловного захвата записей; последующие операторы GET, NEXT и PREVIOUS до тех пор предпринимают попытки захватить запись, пока попытка не завершится успешно. Если запись захвачена другой рабочей станцией, то попытки захватить запись будут продолжаться до тех пор, пока другой пользователь не освободит ее.

**HOLD(файл,секунд)** Включает процесс условного удержания т.е. после безуспешных попыток захватить запись в течение заданного числа секунд операторы GET, NEXT и PREVIOUS выдают сообщение об ошибке “Record Is Already Held” (запись уже кем-то захвачена).

В каждом файле пользователь может за раз захватить одну запись. Если в том же файле происходит обращение ко второй записи, то ранее захваченная запись автоматически освобождается. Как и при блокировке файлов, при захвате записей существует ситуация, которую следует избегать, - взаимная блокировка. Эта ситуация возникает, когда две рабочие станции пытаются захватить один и тот же набор записей в разной последовательности и обе используют форму HOLD(файл) этого оператора. Одна станция уже захватила запись одного файла и пытается захватить запись другого, а другая рабочая станция, наоборот, захватила запись второго файла и пытается захватить запись первого. Этой проблемы можно избежать, используя форму HOLD(файл, секунд) и отслеживая ситуацию “Record Is Already Held”.

**Пример:**

```

LOOP          !Цикл для предотвращения взаимной блокировки
  HOLD(Master,1)  !Включить захват записей главного файла в теч. 1 сек
  GET(Master,1)   !Прочитать и захватить запись
  IF ERRORCODE() = 43  !Если кто-то уже сделал это
    CYCLE         ! попытаться еще раз
  END

```

HOLD(Detail, 1)	!В теч. 1 сек пытаться захватить записи файла Detail
GET(Detail, 1)	!Прочитать и захватить запись
IF ERRORCODE() = 43	!Если кто-то уже сделал это
RELEASE(Master)	! освободить захваченную запись
CYCLE	! попытаться все сначала
END	!Конец цикла, конец структуры IF
BREAK	
END	

Смотри также: RELEASE, GET, NEXT, PREVIOUS

## NEXT (прочитать следующую запись)

### NEXT(файл)

**NEXT** считывает следующую запись файла в текущей последовательности.  
*файл* Метка объявления структуры FILE.

Оператор **NEXT** считывает следующую в заданной ранее последовательности запись файла и помещает ее в буфер структуры RECORD. Последовательность, в которой считываются записи, определяется оператором SET. Первое выполнение оператора NEXT считывает запись в позиции, указанной оператором SET. Последующие выполнения оператора NEXT считывают следующие записи в заданной оператором SET последовательности. Выполнение операторов GET, ADD, PUT и DELETE не влияет на установленную последовательность обработки записей.

Выполнение оператора NEXT без предварительной установки оператором SET положения в файле и последовательности обработки или попытка считывания записи после конца файла приводит к возникновению ошибочной ситуации “Record Not Available” (запись недоступна)

#### Выдаваемые сообщения об ошибках:

33	Record Not Available (запись недоступна)
37	File Not Open (файл не открыт)
43	Record Is Already Held (запись уже заблокирована)

#### Пример:

SET(Cus:NameKey)	!Встать на начало файла в последовательности
ключа	
LOOP UNTIL EOF(Customer)	!Читать все записи до конца файла
NEXT(Customer)	! считать очередную запись
IF ERRORCODE() THEN STOP(ERROR()).	
DO PostTrans	! обратиться к подпрограмме начала транзакции

!Конец цикла

Смотри также: SET, PREVIOUS, EOF, HOLD

## **NOMEMO (включить режим выборки без MEMO-полей)**

### **NOMEMO(файл)**

**NOMEMO**                      Включает режим выборки записей без MEMO-полей.  
*файл*                              Метка файла

Оператор **NOMEMO** включает режим выборки записей данных последующими операторами GET, NEXT и PREVIOUS без выборки значения MEMO-поля. В таком режиме эти операторы считывают запись данных, но не выбирают соответствующее ей MEMO-поле. Обычно это ускоряет доступ к записи, если содержимое MEMO-поля не требуется в процессе работы процедуры.

### **Пример:**

```
SET(Master)
LOOP
  NOMEMO(Master)      !Включить режим без MEMO-полей
  NEXT(Master)        !Получить запись без MEMO-поля
  IF ERRORCODE() THEN BREAK.
  Queue = Mst:Record   !Заполнить очередь
  ADD(Queue)
  IF ERRORCODE() THEN STOP(ERROR()).
  ..                  !Конец IF, конец цикла
  DISPLAY(?ListBox)   !Вывести очередь
```

Смотри также: GET, NEXT, PREVIOUS

## **PREVIOUS (прочитать предыдущую запись)**

### **PREVIOUS(файл)**

**PREVIOUS**                      Считывает предыдущую запись файла в установленной последовательности.  
*файл*                              Метка объявления структуры FILE.

Оператор **PREVIOUS** считывает предыдущую в заданной ранее последовательности запись файла и помещает ее в буфер структуры RECORD. Последовательность, в которой считываются записи, определяется оператором SET. Первое выполнение оператора PREVIOUS считывает запись в позиции, указанной оператором SET. Последующие выполнения оператора PREVIOUS считывают следующие записи в обратной последовательности.



Выполнение операторов GET, ADD, PUT и DELETE не влияет на установленную последовательность обработки записей.

Выполнение оператора PREVIOUS без предварительной установки оператором SET положения в файле и последовательности обработки или попытка считывания записи после прочтения первой записи файла приводит к возникновению ошибочной ситуации “Record Not Available” (запись недоступна)

Выдаваемые сообщения об ошибках:

- 33 Record Not Available (запись недоступна)
- 37 File Not Open (файл не открыт)
- 43 Record Is Already Held (запись уже заблокирована)

**Пример:**

SET(Trn:Datekey)	!Начало/конец файла в последовательности ключа
LOOP UNTIL BOF(Trans)	!Читать все записи в обратном порядке
PREVIOUS(Trans)	! читать записи последовательно
IF ERRORCODE() THEN STOP(ERROR()).	
DO LastInFirstOut	! обратиться к подпрограмме организации
очереди-стека	
.	!Конец цикла

**Смотри также:** SET, NEXT, BOF, HOLD

## PUT (записать данные обратно в файл)

**PUT(файл[,указатель[,длина)])**

<b>PUT</b>	Записывает данные обратно в файл.
<i>файл</i>	Метка объявления структуры FILE.
<i>указатель</i>	Числовая константа, переменная или выражение, представляющее значение, возвращаемое для данной записи функцией POINTER(файл). Конкретное значение зависит от файлового драйвера.
<i>длина</i>	Целочисленная константа, переменная или выражение, которая содержит число байтов, которые следует записать в файл. Длина должна быть больше нуля и не больше длины структуры RECORD. Если этот параметр опущен или выходит за границы допустимого диапазона, то он устанавливается равным длине структуры RECORD. Оператор <b>PUT</b> записывает находящиеся в данный момент в буфере RECORD данные на место ранее прочитанной в файле записи.
PUT(файл)	Заносит запись на место последней полученной оператором NEXT, PREVIOUS, GET или ADD. Если в буфере изменились значения полей, входящих в ключи, то обновляются и файлы ключей.

PUT(файл, указатель)      Заносит запись в файл, в заданной указателем место и обновляет ключи.

PUT(файл, указатель, длина)      Записывает указанное параметром длина число байт по заданному указателем адресу в файл и обновляет ключи.

Если к записи не было обращения с помощью операторов NEXT, PREVIOUS, GET, ADD или запись удалена, то выдается сообщение “Record Not Available” (запись недоступна). Кроме этого сообщения, оператор PUT выдает сообщение “Creates Duplicate Key” (создается повторяющееся значение ключа). Если выдано какое-либо сообщение об ошибке, то запись не заносится в файл.

#### Выдаваемые сообщения об ошибках:

05      Access Denied (доступ запрещен)

33      Record Not Available (запись недоступна)

40      Creates Duplicate Key (создается повторяющееся значение ключа)

#### **Пример:**

```
SET(Trn:Datekey)                      !Начало/конец файла в последовательности ключа
LOOP UNTIL BOF(Trans)                !Читать все записи в обратном порядке
  PREVIOUS(Trans)                    ! читать записи последовательно
  IF ERRORCODE() THEN STOP(ERROR()).
DO LastInFirstOut
  ! обратиться к подпрограмме организации очереди-стека
  PUT(Trans)                          !Занести запись транзакции обратно в файл
  IF ERRORCODE() THEN STOP(ERROR()).
.                                      !Конец цикла
```

**Смотри также:** NEXT, PREVIOUS, GET, ADD

## **RELEASE (освободить захваченную запись)**

### **RELEASE(файл)**

**RELEASE**                      Освобождает захваченную запись  
*файл*                              Метка объявления структуры FILE.

Оператор **RELEASE** освобождает ранее захваченную запись. Но он не освободит запись, захваченную другим пользователем. Если запись не захвачена, или захвачена другим пользователем, то оператор RELEASE игнорируется.

#### **Пример:**

```
LOOP                              !Цикл для предотвращения взаимной блокировки
```

```

HOLD(Master,1)    !Включить захват записей главного файла в теч. 1 сек
GET(Master,1)     !Прочитать и захватить запись
IF ERRORCODE() = 43 !Если кто-то уже сделал это
    CYCLE         !попытаться еще раз

END
HOLD(Detail,1)    !В теч. 1 сек пытаться захватить записи файла Detail
GET(Detail,1)     !Прочитать и захватить запись
IF ERRORCODE() = 43 !Если кто-то уже сделал это
    RELEASE(Master) ! освободить захваченную запись
    CYCLE           ! попытаться все сначала
END               !Конец цикла, конец структуры IF
BREAK
END

```

## REGET (повторно прочитать запись файла)

**REGET**(последовательность, строка)

**REGET** Повторно прочитывает заданную запись файла.

*последовательность* Метка оператора FILE, KEY, или INDEX.

*строка* Строка, возвращенная ранее функцией POSITION.

Оператор **REGET** считывает запись, идентифицированную строкой, возвращенной ранее функцией POSITION. Содержащаяся в строке значение, возвращенное функцией POSITION, и его длина зависят от файлового драйвера.

### Выдаваемые сообщения об ошибках:

33 Record Not Available (запись недоступна)

### Пример:

```

RecordQue QUEUE,PRE(Dsp)
QueFields LIKE(Trn:Record),PRE(Dsp)
END
SavPosition STRING(260)
CODE
SET(Trn:DateKey) !Начало файла в последовательности ключа.
LOOP             !Прочитать все записи файла
    NEXT(Trans)  ! читать записи последовательно
    IF ERRORCODE() THEN BREAK.
    RecordQue = Trn:Record !Поместить запись в буфер очереди
    ADD(RecordQue)        ! и добавить ее в очередь
    IF ERRORCODE() THEN STOP(ERROR()).
    IF RECORDS(RecordQue) = 20 OR EOF(Trans)
        !20 записей в очереди или конец файла?
        SavPosition = POSITION(Trn:DateKey) !Запомнить положение записи

```

DO DisplayQue	!Вывести на экран очередь
FREE(RecordQue)	! и очистить ее
REGET(Trn:DateKey,SavPosition)	! и прочитать запись снова

..

Смотри также: POSITION, RESET

## RESET (восстановить положение в последовательности записей)

**RESET**(*последовательность, строка*)

**RESET** Восстанавливает указатель последовательной обработки на заданную запись.

*последовательность* Метка объявления файла, ключа или индекса.

*строка* Строка, возвращенная ранее функцией POSITION.

Оператор **RESET** восстанавливает указатель на запись, идентифицированную строкой, возвращенной ранее функцией POSITION. Поскольку оператор RESET восстановил указатель на запись, то либо оператор NEXT, либо PREVIOUS считают эту запись.

Содержащаяся в строке значение, возвращенное функцией POSITION, и его длина зависят от файлового драйвера. Оператор RESET в сочетании с функцией POSITION используется для временной приостановки и продолжения последовательной обработки файла.

### Пример:

RecordQue QUEUE,PRE(Dsp)

QueFields LIKE(Trn:Record),PRE(Dsp)

.

SavPosition STRING(260)

CODE

SET(Trn:DateKey) !Начало файла в последовательности ключа

LOOP !Читать все записи файла

NEXT(Trans) ! читать записи последовательно

IF ERRORCODE() THEN STOP(ERROR()).

RecordQue = Trn:Record !Поместить запись в буфер

ADD(RecordQue) ! и добавить в очередь

IF ERRORCODE() THEN STOP(ERROR()).

IF RECORDS(RecordQue) >= 20 OR EOF(Trans)

!В очереди 20 записей ?

SavPosition = POSITION(Trn:DateKey) !Запомнить положение в файле

DO DisplayQue !Вывести на экран очередь

FREE(RecordQue) ! и очистить очередь

IF EOF(Trans) THEN BREAK. !Выйти из цикла после обработки всех записей

```

RESET(Trn:DateKey,SavPosition) !Восстановить указатель на запись
... !Конец цикла

```

**Смотри также:** POSITION, NEXT, PREVIOUS

## SET (начать последовательную обработку файла)

*файл*

**SET**(файл, ключ)

*файл, указатель в файле*

*ключ*

*ключ, ключ*

*ключ, указатель в ключах*

*ключ, ключ, указатель в ключах*

**SET**

Начинает последовательную обработку файла

*файл*

Метка объявления структуры FILE. Этот параметр задает обработку в порядке физического расположения записей в файле данных.

*ключ*

Метка объявления ключа или индекса. При использовании в качестве первого параметра ключ задает обработку в логической последовательности упорядочения ключа или индекса.

*указатель в файле*

Числовая константа, переменная или выражение представляющее собой значение возвращенное функцией POINTER(файл). Конкретное значение зависит от файлового драйвера.

*указатель в ключах*

Числовая константа, переменная или выражение представляющее собой значение возвращенное функцией POINTER(ключ). Конкретное значение зависит от файлового драйвера.

Оператор **SET** начинает последовательную обработку файла. Запись при его выполнении не считывается, а только устанавливается последовательность обработки и начальная точка для последующих операторов NEXT и PREVIOUS. Первый параметр оператора определяет последовательность, в которой обрабатываются записи. Второй и третий параметры определяют начальную точку обработки. Если второй и третий параметры опущены, то обработка начинается с начала (или с конца) файла.

SET(файл)

Задает обработку в последовательности физического расположения записей и устанавливает текущий указатель на начало (SET...NEXT) или конец (SET...PREVIOUS) заданного файла,

SET(файл, ключ)

Задает обработку в последовательности физического расположения записей и устанавливает текущий указатель на первую запись, которая содержит значения полей, совпадающие со значениями полей-компонент ключа в буфере RECORD.

SET(файл, указатель в файле)

Задает обработку в последовательности физического

расположения записей и устанавливает текущий указатель на заданную указателем в файле запись файла данных.

**SET(ключ)**           Задаёт обработку в последовательности ключа и устанавливает текущий указатель на начало (SET...NEXT) или конец (SET...PREVIOUS) файла в этой логической последовательности.

**SET(ключ, ключ)**   Задаёт обработку в последовательности ключа и устанавливает текущий указатель на первую запись, которая содержит значения полей, совпадающие со значениями полей-компонент ключа в буфере RECORD. Оба параметра должны быть одинаковыми.

**SET(ключ, указатель в ключах)**   Задаёт обработку в последовательности ключа и устанавливает текущий указатель в файле ключей в соответствии со вторым параметром.

**SET(ключ, ключ, указатель в файле)**   Задаёт обработку в последовательности ключа и устанавливает текущий указатель на запись, которая содержит значения полей, совпадающие с значениями компонент ключа, и у которой номер записи равен значению параметра указатель в файле. Первые два параметра должны совпадать.

Когда вторым параметром оператора является ключ, обработка начинается с первой записи, удовлетворяющей заданным в буфере RECORD значениям полей-компонент ключа или индекса. Если запись, с соответствующими значениями найдена, то оператор NEXT или PREVIOUS прочитают эту запись. Если же запись с точным соответствием значений компонент ключа не найдена, то оператором NEXT считывается запись со следующим, большим заданного значением ключа, а оператором PREVIOUS считывается запись с предыдущим, меньшим заданного значением.

Значения параметров указатель в файле и указатель в ключах зависят от файлового драйвера. Это может быть номер записи, относительное смещение в файле или некий другой тип “поисковой позиции” внутри файла. Эти параметры используются для того, чтобы начать обработку с заданной записи в файле.

Для всех файловых драйверов попытка выполнить оператор SET на запись “после конца” файла установит значение функции EOF “истина”, а попытка установить запись “перед началом” файла установит значение истина для функции BOF.

### Пример:

```
SET(Customer)      !Физический порядок обработки с начала файла
Cus:Name = 'Smith'
SET(Customer,Cus:NameKey) !Физический порядок обработки, с записи с Name = 'Smith'
SavePtr = POINTER(Customer)
SET(Customer,SavePtr)   !Физический порядок обработки, начав с номера = SavePtr
SET(Cus:NameKey)       !В порядке ключа NameKey, с начала файла (по ключу)
SavePtr = POINTER(Cus:NameKey)
```

```

SET(Cus:NameKey,SavePtr)    !В порядке NameKey, с номера записи в ключах = SavePtr
Cus:Name = 'Smith'
SET(Cus:NameKey,Cus:NameKey) !В порядке NameKey, с записи с Name = 'Smith'
Cus:Name = 'Smith'
SavePtr = POINTER(Customer)
SET(Cus:NameKey,Cus:NameKey.SavePtr)
    !В порядке NameKey, с записи с Name = 'Smith' и номером записи = SavePtr

```

**Смотри также:** NEXT, PREVIOUS, KEY, RECORD, POINTER

## SKIP (пропустить запись при последовательной обработке)

**SKIP**(*файл, число*)

### SKIP

*файл*

*число*

Пропускает запись при последовательной обработке файла.

Метка объявления структуры FILE.

Числовая константа или переменная. Параметр число задает число записей, которое следует пропустить. Если значение числа положительное, то записи пропускаются в “прямой” последовательности (как если бы выполнялся оператор NEXT). Если же число отрицательное, то записи пропускаются в “обратной” последовательности (как если бы выполнялся оператор PREVIOUS).

Оператор **SKIP** используется для того, чтобы в процессе последовательной обработки файла пропускать записи. Этот оператор пропускает записи в последовательности, заданной оператором SET, изменяя указатель текущий записи на указанное число записей. Для пропуска записей оператор SKIP более эффективен, чем NEXT или PREVIOUS, потому что он перемещает данные из файла в буфер структуры RECORD.

Если при пропуске записей оператором SKIP происходит выход за начало или конец файла, то функции EOF() или BOF() возвращают значение “истина”. Если последовательность обработки файла предварительно не была установлена оператором SET, то оператор SKIP игнорируется.

### Пример:

```

SET(ltm:InvoiceKey)                !Установить начало файла
LOOP UNTIL EOF(ltems)              !Обработать все записи
NEXT(ltems)                        ! Прочитать запись
IF ERRORCODE() THEN STOP(ERROR()).
IF ltm:InvoiceNo <> SavInvNo        ! Проверить на первый пункт в заказе
Hea:InvoiceNo = ltm:InvoiceNo      ! Присвоить значение ключевому полю
GET(Header,Hea:InvoiceKey)        ! Получить соответствующую запись заголовка
IF ERRORCODE() THEN STOP(ERROR()).
IF Hea:InvoiceStatus = 'Cancel'    ! Это отмененный заказ ?

```

```

SKIP(Items,Hea:ItemCount-1)      ! отступить на один пункт назад
CYCLE ! и обработать следующий заказ
.. ! Конец двух IF
DO ItemProcess                    ! обработать пункт
SavInvNo = Itm:InvoiceNo          ! запомнить номер счета
. !Конец цикла

```

## WATCH (автоматическая проверка совместного использования)

### WATCH(*файл*)

**WATCH** Включает автоматическую проверку совместного использования базы данных по оптимистической стратегии.

*файл* Метка объявления файла.

Оператор **WATCH** включает автоматическую проверку файловым драйвером совместного использования базы данных для последующих операторов NEXT и PREVIOUS в многопользовательской среде по оптимистической стратегии. Обычно файловый драйвер сохраняет копию значений полей, прочитанных из каждого файла при успешном выполнении операторов NEXT и PREVIOUS. При записи оператором PUT обратно в виртуальный файл поля на диске сравниваются с первоначально выбранными данными. В случае изменения их другим пользователем при выполнении оператора PUT выдается код ошибки. Конкретные действия, выполняемые оператором WATCH, зависят от файлового драйвера.

### Пример:

```

SET(Itm:InvoiceKey)      !Встать на начало файла деталей
LOOP                     !Обработать все записи
  WATCH(Items)           !Включить проверку совместного использования
  NEXT(Items)             ! прочитать запись
  IF ERRORCODE() THEN BREAK.
  DO ItemProcess          ! обработать элемент
  PUT(Items)              ! и записать его обратно
  IF ERRORCODE() THEN STOP(ERROR()).
                           !Остановиться по любой ошибке, включая
                           ! то, что запись изменена другим пользователем
END

```



## Функции для работы с файлами

### BOF (сигнализировать о достижении начала файла)

#### BOF(*файл*)

**BOF** Сигнализирует о достижении начала файла во время последовательной обработки.

*файл* Метка объявления структуры FILE.

Функция **BOF** возвращает отличное от нуля значение (“истина”), когда оператором PREVIOUS прочитана (или пропущена оператором SKIP) первая в последовательности обработки запись файла. Во всех других случаях функция возвращает ноль (“ложь”).

Наиболее часто функция BOF используется в качестве условия в операторе LOOP UNTIL. Поскольку условие выполнения вычисляется в начале цикла, функция BOF возвращает значение “истина” после того, как при обработке файла в обратной последовательности прочитана и обработана последняя запись.

Функция BOF может не поддерживаться некоторыми файловыми драйверами или быть в этих файловых системах очень неэффективной. Перед ее использованием сверьтесь с документацией по данному файловому драйверу.

Тип возвращаемого значения: LONG

#### Пример:

SET(Trn:DateKey)	!Начало/конец файла в последовательности ключа
LOOP UNTIL BOF(Trans)	!Обработать файл в обратной последовательности
PREVIOUS(Trans)	! последовательно читать записи
IF ERRORCODE() THEN STOP(ERROR()).	
DO LastInFirstOut	! обратиться к подпрограмме
.	!Конец цикла

Смотри также: PREVIOUS, SKIP, LOOP

### BYTES (получить размер файла в байтах)

#### BYTES(*файл*)

**BYTES** Возвращает размер файла или последней прочитанной записи в байтах

*файл* Метка объявления структуры FILE.

Функция **BYTES** возвращает размер файла в байтах или число байт в записи, к которой

было последнее обращение. Сразу после выполнения оператора OPEN эта функция возвращает размер файла. А после того, как к этому файлу были обращения с помощью операторов GET, NEXT, ADD и PUT, функция BYTES возвращает число байт в записи, к которой осуществлялся доступ. Эту функцию можно использовать для того, чтобы получать число байт, прочитанных в записи переменной длины.

Тип возвращаемого значения: LONG

### Пример:

```

OPEN(DosFile)                !Открыть файл
IF (BYTES(DosFile) % 80) > 4  !Проверить на неполную последнюю запись
    SavPtr = INT(BYTES(DosFile) % 80) + 1) ! вычислить номер неполной записи
ELSE
    SavPtr = BYTES(DosFile) / 80          ! вычислить номер последней записи
.
GET(DosFile,SavPtr)           !Прочитать последнюю запись
LastRec = BYTES(DosFile)      !Запомнить размер неполной записи

```

## DUPLICATE (проверить на повторение значения ключа)

**DUPLICATE**( *ключ* )  
                  *файл*

<b>DUPLICATE</b>	Проверяет значения ключа на повторение.
<i>ключ</i>	Метка объявления ключа
<i>файл</i>	Метка объявления структуры FILE.

Функция **DUPLICATE** возвращает отличное от нуля значение (“истина”), занесение текущей записи в файл данных приведет к возникновению ошибочной ситуации “CREATES DUPLICATE KEY” (создается повторяющееся значение ключа). Если указан параметр ключ, то заданный ключ и проверяется на возможность повторения. Если задан параметр файл, то проверяются все ключи, не имеющие атрибута DUP.

Функция **DUPLICATE** подразумевает, что содержимое структуры RECORD дублируется на место в файле, определяемое текущим указателем. Поэтому, при использовании функции **DUPLICATE** перед оператором **ADD** указатель должен быть обнулен с помощью оператора **GET(файл,0)**

Тип возвращаемого значения: LONG

### Пример:

```

IF Action = 'ADD' THEN GET(Vendor,0).    !Если добавление, очистим указатель

```

```

IF DUPLICATE(Vendor)           !Если этот продавец уже существует
SCR:MESSAGE "Vendor Number already assigned" ! высветить сообщение
SELECT(?)                     ! и оставаться в этом поле
.                               !Конец структуры IF

```

Смотри также: GET

## EOF (сигнализировать о достижении конца файла)

### EOF(*файл*)

**EOF** *файл*                      Сигнализирует о достижении конца файла  
                                     Метка объявления структуры FILE.

Функция **EOF** возвращает отличное от нуля значение (“истина”), когда оператором NEXT прочитана (или пропущена оператором SKIP) последняя в последовательности обработки запись файла. Во всех других случаях функция возвращает ноль (“ложь”). Наиболее часто функция EOF используется в качестве условия в операторе LOOP UNTIL. Поскольку условие выполнения вычисляется в начале цикла, функция EOF возвращает значение “истина” после того, как прочитана и обработана последняя запись.

Функция EOF может не поддерживаться некоторыми файловыми драйверами или работать в их фаловых системах крайне неэффективно. Перед ее использованием сверьтесь с документацией по данному файловому драйверу.

Тип возвращаемого значения: LONG

### Пример:

```

SET(Trn:DateKey)               !Начало/конец файла в последовательности ключа
LOOP UNTIL EOF(Trans)          !Обработать все записи файла
NEXT(Trans)                    ! последовательно читать записи
IF ERRORCODE() THEN STOP(ERROR()).
DO LastInFirstOut              ! обратиться к подпрограмме
.                               !Конец цикла

```

Смотри также: NEXT, SKIP, LOOP

**NAME (получить имя файла в DOS или имя устройства)****NAME**(метка)

**NAME** Возвращает имя файла.  
*метка* Метка оператора объявления файла.

Функция **NAME** возвращает строковую константу содержащую имя устройства DOS для структуры, указанной меткой. Для структуры FILE при условии, что файл открыт, возвращается полная спецификация (диск, путь, имя и расширение). Если файл закрыт, то возвращается содержимое атрибута NAME структуры файл.

Тип возвращаемого значения: STRING

**Пример:**

OpenFile = NAME(Customer) !Сохранить имя открытого файла

**POINTER (получить относительное положение записи)****POINTER**( файл )  
ключ

**POINTER** Возвращает относительное положение записи  
*файл* Метка объявления структуры FILE. Этот параметр задает физическую последовательность записей в файле.  
*ключ* Метка объявления ключа или индекса. Задает логическую последовательность записей упорядоченную по заданному ключу или индексу.

Функция **POINTER** возвращает относительное положение в файле данных (в физической последовательности) или относительное положение в файле ключей или индексов (в логической последовательности) последней записи, к которой было обращение. Смысл значения, возвращаемого функцией, зависит от файлового драйвера. Этот может быть номер записи, байтовое смещение в файле или некая другая разновидность “поискового” положения в файле.

Некоторыми файловыми драйверами функция **POINTER** не поддерживается. Поэтому ее следует использовать только, если вы знаете, что она поддерживается в используемой файловой системе и не будете заменять ее в будущем. Предпочтительный способ позиционирования на запись, который разработан для с тем расчетом, чтобы функционировать во всех файловых системах, это использование функции **POSITION** в сочетании с **RESET** и **REGET**.

Тип возвращаемого значения: LONG

### Пример:

SavePtr# = POINTER(Customer)      !Запомнить положение в файле

## Смотри также: SET, POSITION, RESET, REGET

**POSITION** (получить положение в последовательности обработки файла)

**POSITION**(*последовательность*)

<b>POSITION</b>	Однозначно идентифицирует положение в файле.
<i>последовательность</i>	Метка объявления структуры FILE, ключа или индекса.

Функция **POSITION** возвращает строку, которая однозначно идентифицирует положение записи в заданной последовательности. Эта функция возвращает положение записи файла, к которой было последнее обращение (записи, находящейся в данный момент в буфере RECORD). Функция POSITION используется в сочетании с оператором RESET для того, чтобы временно приостанавливать и возобновлять последовательную обработку файла.

Значение, находящееся в возвращаемой функцией строке, и длина этой строки зависят от файлового драйвера. Как правило для файловых систем, в которых записи имеют номера, размер строки, возвращаемой функцией POSITION(файл) равен 4 байта. Размер строки, возвращаемой функцией POSITION(ключ) равен 4 плюс сумма длин полей, составляющих ключ. Для файловых систем, в которых записи не имеют номеров, размер строки, возвращаемой функцией POSITION(файл) обычно равен сумме длин полей первичного ключа (первый ключ файла, не имеющий атрибута DUP и OPT). Размер строки, возвращаемой в этом случае функцией POSITION(ключ) равен сумме длин полей первичного ключа плюс сумме длин полей заданного ключа.

Тип возвращаемого значения: **STRING**

### Пример:

```
RecordQue  QUEUE,PRE(Dsp)
QueFields  LIKE(Trn:Record),PRE(Dsp)
.
SavPosition STRING(260)
CODE
SET(Trn:DateKey)                !Начало файла в последовательности ключа
LOOP !Читать все записи файла
    NEXT(Trans)                  ! читать последовательно запись
```

```

IF ERRDRCODE() THEN STOP(ERROR()).
RecordQue = Trn:Record      !Поместить запись в буфер очереди
ADD(RecordQue)              ! и добавить ее в очередь
IF ERRORCODE() THEN STOP(ERROR()).
IF RECORDS(RecordQue) >= 20 OR EOF(Trans) ! в очереди 20 записей ?
SavPosition = POSITION(Trn:DateKey)    !Запомнить положение записи в файле
DO DisplayQue                        !Вывести очередь на экран
FREE(RecordQue)                     ! и очистить ее
IF EOF(Trans) THEN BREAK.           !Выйти из цикла, когда все записи обработаны
RESET(Trn:DateKey,SavPosition)      !Восстановить указатель на запись
. . .                               !Конец цикла

```

Смотри также: RESET, REGET

## RECORDS (получить число записей или значений ключа)

**RECORDS**( *файл*)  
*ключ*

<b>RECORDS</b>	Возвращает число записей или значений ключа
<i>файл</i>	Метка объявления структуры FILE.
<i>ключ</i>	Метка объявления ключа или индекса.

Функция **RECORDS** возвращает число записей в файле или ключе. Поскольку атрибут OPT операторов KEY или INDEX задает исключение “пустых” значений, функция RECORDS для ключа или индекса может возвращать меньшее значение чем для файла данных.

Тип возвращаемого значения: LONG

### Пример:

```

SaveCount = RECORDS(Master)      !Запомнить число записей
SaveNameCount = RECORDS(Nam:NameKey) !Число записей с заполненным полем Name

```

Смотри также: KEY, INDEX, OPT

**SEND (послать сообщение файловому драйверу)****SEND**(*файл*, *сообщение*)**SEND**

Посылает сообщение файловому драйверу

*файл*

Метка объявления структуры FILE. Атрибут DRIVER этой структуры идентифицирует файловый драйвер, который должен получить сообщение.

*сообщение*

Строковая константа или переменная, содержащая информацию, которая предназначена для передачи файловому драйверу.

Функция **SEND** позволяет программе во время выполнения передать любые параметры, характерные для какого-то файлового драйвера. Конкретные примеры допустимых сообщений перечисляются в документации на файловый драйвер.

Тип возвращаемого значения: STRING

**Пример:**

FileCheck = SEND(ClarionFile, 'RECOVER=124')

!Включить восстановление файла данных Clarion

## **Обработка транзакций**

База данных сохраняет свою целостность, когда записи данных содержат правильные данные (целостность данных), и поля ключей точно отражают связи между записями и файлами (целостность связей). Только тщательное проектирование и программирование могут обеспечить целостность данных. Если важно значение конкретного элемента данных, то для обнаружения и исключения неверных данных нужно написать процедуры редактирования данных при вводе. Если запись с первичным ключом (Primary Key) не должна удаляться до тех пор, пока существуют записи со вторичным ключом (Foreign Key), то логика работы программы не должна допускать такого удаления.

Обработка транзакций представляет собой один из механизмов языка Clarion, которые помогают программисту обеспечивать целостность базы данных. Обработку транзакций еще часто называют “отслеживанием транзакций”, “регистрацией транзакций” или реализацией “фиксирования границ”. Неважно как это называется, важно что это технология программирования, которая позволяет значительно снизить ваши затраты на поддержание целостности базы данных.

Посредством своей технологии файловых драйверов Clarion поддерживает множество файловых систем. Некоторые из этих файловых систем не поддерживают обработку транзакций, другие реализуют ее, используя незначительно отличающиеся методы. Поэтому здесь обсуждается обработка транзакций в общем случае. По вопросу поддерживает ли файловый драйвер обработку транзакций и относительно отличий в реализации этой обработки следует обращаться к документации на конкретный файловый драйвер.

### **Определение транзакции**

---

Транзакцию можно определить как:

Единое логическое событие, в течение которого множество записей данных выводится на диск и любой сбой во время этих дисковых операций поставил бы под сомнение целостность базы данных.

В транзакцию могут включаться несколько записей одного файла или одна или несколько записей из нескольких файлов. Главным требованием транзакции является следующее: все записи должны быть успешно записаны на диск или никакая из них не должна быть записана вообще. Таким образом главный принцип транзакции “все или ничего”.



## Границы транзакции

У транзакции всегда есть явное начало и конец, которые определяют “границы транзакции”. Одним из главных правил обработки транзакций является стремление сделать протяженность транзакции как можно меньше. Тому есть несколько причин. Любой сбой во время транзакции ставит под сомнение целостность базы данных. Следовательно, период времени, в течение которого возможно нарушение целостности базы данных следует делать возможно меньшим, чтобы снизить вероятность такого события. Вторая причина - это то, что в некоторых файловых системах при работе в многопользовательском режиме на время обработки транзакции требуется исключительный доступ к файлам. Это означает, что на время транзакции другим пользователям запрещается доступ к файлам участвующим в транзакции. Ясно, что это убедительный аргумент в пользу того, чтобы уменьшать размеры транзакции.

В течение транзакции изменения в файлах, которые участвуют в этой транзакции, “отслеживаются” или “регистрируются” в специальном файле (файле регистрации или Pre-Image файле). В этом файле сохраняется столько информации, сколько нужно, чтобы, если потребуется, восстановить то состояние файлов, которое они имели перед началом транзакции.

Транзакция заканчивается, когда она или “откатывается” назад или “фиксируется”.

- Если во время транзакции произошли какие-либо ошибки, изменения базы данных, сохраненные в Pre-Image файле откатываются назад, к прежнему состоянию. Поскольку изменения убраны, состояние базы данных снова соответствует тому, которое было перед началом транзакции.
- Если во время транзакции ошибок не было, то это успешная транзакция. Следовательно, она фиксируется, и сделанные изменения нужно оставить в базе данных.

В любом случае в конце транзакции сохраняется внутренняя непротиворечивость базы данных.

Если транзакция прерывается из-за исчезновения электропитания или сбоя операционной системы общего характера, то частично выполненную транзакцию нужно откатить назад, в противном случае нельзя гарантировать целостность базы данных. В каждой файловой системе по разному определяется необходимость отката незавершенной транзакции. В файловых системах с архитектурой Клиент/Сервер, незавершенные транзакции обычно обнаруживаются при запуске программы - Сервера файловой системы на компьютере - сетевом файл-сервере. В файловых системах других типов это обнаруживается обычно при открытии файлов участвовавших в транзакции. Как бы это не определялось как только обнаруживается любая незавершенная транзакция она автоматически откатывается или ядром файловой системы, или драйвером Clarion для этой файловой системы.

## LOGOUT, COMMIT, ROLLBACK

---

В языке Clarion существует три оператора, с помощью которых реализуется обработка транзакций: LOGOUT, COMMIT, and ROLLBACK.

Оператор LOGOUT начинает транзакцию. В нем перечисляются все файлы, которые будут участвовать в ней - для всех этих файлов должен использоваться один и тот же файловый драйвер. Для этого имеется очень серьезная причина.

В рамках каждой файловой системы механизм поддержки обработки транзакций спроектирован таким образом, что в нем нет “окон уязвимости”. “Окно уязвимости” представляет собой участок программного кода, случись при выполнении которого авария (сбой электропитания или сбой системы) целостность базы данных будет безвозвратно нарушена. Поддержка обработки транзакций в каждой файловой системе специально спроектирована так, чтобы исключить наличие таких окон.

Если допустить включение в транзакцию файлов, относящихся к различным системам, то между драйверами возникло бы “окно уязвимости”. Ведь каждый из них независимо управляет обработкой транзакций для своих файлов. Предположим сбой происходит во время фиксации такой транзакции, в этом случае любой еще незафиксировавший свою часть транзакции драйвер решил бы, что она осталась незавершенной. Незавершенная же транзакция автоматически обнаруживается и откатывается. Случись это, целостность базы данных была бы нарушена из-за того, что одна часть транзакции зафиксирована, а для другой выполнен откат. Вот поэтому, все файлы транзакции должны использовать один и тот же драйвер.

Оператор COMMIT завершает успешную транзакцию. Причем только явно указанный, не может быть “подразумеваемой” фиксации. Если файл закрывается до фиксации транзакции, явно (оператором CLOSE), или неявно (операторами RUN, CHAIN, RUNSMALL и т.д.), транзакция считается незавершенной. В большинстве систем оператор COMMIT удаляет файл регистрации, который обеспечил бы осуществление отката транзакции.

Оператором ROLLBACK заканчивается неудавшаяся транзакция. Для того, чтобы восстановить файлы, входящие в транзакцию, используется информация, сохраненная в Pre-Image-файле. В большинстве файловых систем после отката транзакции оператор ROLLBACK удаляет файл регистрации.

## Вопросы многопользовательской работы

---

При обработке транзакций в многопользовательской среде в расчет должны приниматься некоторые дополнительные соображения. Первое - это то, что для сетевого файлового сервера абсолютно необходим бесперебойный источник питания. Он необходим потому,

что сетевые операционные системы “обманывают” прикладные программы, когда те просят записать данные на диск. Операционная система сообщает прикладной программе об успешной записи на диск, в то время как, фактически, данные все еще находятся в ее системном кэше или буфере, и физически еще не записаны на диск.

Без бесперебойного источника на сервере любой сбой питания в то время, когда записи еще находятся в системном буфере/кэше, может привести к нарушению целостности базы данных. В зависимости от установленных значений параметров сетевой операционной системы время нахождения записей в буфере может достигать тридцати секунд. Это немыслимо большое время для образования “окна уязвимости”. Итак, при обработке транзакций в многопользовательской среде необходим бесперебойный источник питания для файлового сервера.

В некоторых файловых системах требуется исключительный доступ к файлам (блокировка файлов), включенным в транзакцию. Это не проблема, так как оператор LOGOUT, если нужно, автоматически блокирует доступ к файлам. Необходимость блокировки файлов определяется параметрами режима доступа, с которыми файл открывался. Любые файлы, открытые в режиме “Нет запрета” или “Запрет чтения” для “других пользователей”, являются совместно используемыми ресурсами и заблокируются, если это требуется файловой системе. Если файлы заблокированы оператором LOGOUT, то разблокирует их или оператор COMMIT, или оператор ROLLBACK. Это делает обработку транзакций абсолютно одинаковой независимо от того, используется ли однопользовательская система или многопользовательская.

Важно знать, блокируются ли автоматически оператором LOGOUT файлы или нет. Если блокируются и, кроме того, если в какой-то программе, обращающейся в это время к тем же самым файлам, используется оператор HOLD, может возникнуть взаимная блокировка второго типа, обсуждающаяся в разделе Совместное использование файлов. Программа должна быть написана так, чтобы в транзакции обнаруживались заблокированные записи, транзакция “откатывалась”, а пользователю предоставлялась возможность повторить транзакцию. Кроме того, нужно выполнять “проверку вмешательства”, которая описывалась в разделе Совместное использование файлов. При корректировке записей транзакции нужно иметь возможность распознавать изменения, внесенные другими пользователями.

## COMMIT (завершить успешную транзакцию)

### COMMIT

Оператор **COMMIT** завершает активную транзакцию. Выполнение этого оператора предполагает, что данная транзакция была полностью успешной и нет необходимости в откате. После выполнения оператора COMMIT откат транзакции становится невозможен.

Оператор COMMIT сообщает драйверу, участвующему в транзакции, что можно удалить

временные файлы, содержащие информацию, необходимую для того, чтобы восстановить базу данных в предшествующее транзакции состояние. Затем, этот файловый драйвер выполняет действия, необходимые в данной файловой системе для того, чтобы успешно завершить транзакцию.

**Пример:**

```
LOGOUT(.1,OrderHeader.OrderDetail)    !Начать транзакцию
DO ErrHandler                          ! всегда проверяя ошибки
ADD(OrderHeader)                       !Добавить порождающую запись
DD ErrHandler                          ! всегда проверяя ошибки
LOOP X# = 1 TO RECORDS(Detail4ue)      !Процесс занесения порожденных записей
  GET(Detail4ue,X#)                    ! Взять одну из очереди
  DO ErrHandler                        ! проверяя ошибки
  Det:Record ' Detail4ue                !поместить в буфер записи
  ADD(OrderDetail)                      !и добавить в файл
  DO ErrHandler                        ! проверив ошибки
.
COMMIT                                 !Завершить успешную транзакцию

ErrHandler ROUTINE                     !Подпрограмма проверки ошибок
  IF NOT ERRORCODE() THEN EXIT.         !Выйти, если нет ошибок
  ROLLBACK                              !Откатить неуспешную транзакцию
.
  BEEP !Оповестить пользователя
  SHOW(25,1,'Transaction Error - ' & ERR4R())
  ASK  !и ждать его реакции
  RETURN                                !затем выйти
```

**LOGOUT (начать транзакцию)**

LOGOUT(*интервал*, *файл*, [*файл*, ..., *файл*])

<b>LOGOUT</b>	Начинает транзакцию.
<i>интервал</i>	Числовая константа или переменная, которая задает число секунд, в течение которых производится попытка начать транзакцию для файла, после чего транзакция отвергается и генерируется сообщение об ошибке.
<i>файл</i>	Метка объявления структуры FILE. В списке параметров может быть несколько параметров файл, разделенных запятыми. Должны быть перечислены все файлы, которые будут входить в набор файлов транзакции.

Оператор **LOGOUT** начинает отработку транзакции для заданного набора файлов. Все файлы в этом наборе должны использовать один и тот же файловый драйвер. Этот оператор сообщает файловому драйверу о том, что начинается транзакция. Затем драйвер выполняет действия, необходимые в этой файловой системе для того, чтобы начать отработку транзакции для заданного набора файлов. Если в этой файловой системе требуется, чтобы

для отработки транзакции были заблокированы файлы, то оператор LOGOUT блокирует их.

Одновременно может быть активна только одна транзакция. Второй оператор LOGOUT, выполняемый до оператора COMMIT или ROLLBACK, относящегося к предыдущей транзакции, останавливает программу с выдачей сообщения об ошибке, возвращая пользователя в DOS.

#### Выдаваемые сообщения об ошибках:

32 File Is Already Locked (файл уже заблокирован)

#### **Пример:**

```

LOGOUT(.1,OrderHeader.OrderDetail) !Начать транзакцию
DO ErrHandler                      ! всегда проверяя ошибки
ADD(OrderHeader)                   !Добавить порождающую запись
DD ErrHandler                      ! всегда проверяя ошибки
LOOP X# = 1 TO RECORDS(Detail4ue) !Процесс занесения порожденных записей
  GET(Detail4ue,X#)                ! Взять одну из очереди
  DO ErrHandler                    ! проверяя ошибки
  Det:Record ' Detail4ue           !поместить в буфер записи
  ADD(OrderDetail)                 !и добавить в файл
  DO ErrHandler                    ! проверив ошибки
.
COMMIT                             !Завершить успешную транзакцию
ErrHandler ROUTINE                 !Подпрограмма проверки ошибок
  IF NOT ERRORCODE() THEN EXIT.    !Выйти, если нет ошибок
  ROLLBACK                         !Откатить неуспешную транзакцию
.
  BEEP !Оповестить пользователя
  SHOW(25,1,'Transaction Error - ' & ERR4R())
  ASK  ! и ждать его реакции
  RETURN                           !затем выйти

```

**Смотри также:** COMMIT, ROLLBACK

### **ROLLBACK (завершить неуспешную транзакцию)**

#### **ROLLBACK**

Оператор **ROLLBACK** завершает активную транзакцию. Выполнение оператора ROLLBACK предполагает, что транзакция была неуспешной и база данных должна быть восстановлена в состояние, в котором она находилась перед началом транзакции.

ROLLBACK сообщает файловому драйверу, участвующему в транзакции, что для восстановления базы данных следует использовать временные файлы, содержащие

необходимую информацию. После этого, файловый драйвер выполняет действия, необходимые для отката транзакции в данной файловой системе.

### Пример:

```

LOGOUT(. 1,OrderHeader.OrderDetail) !Начать транзакцию
DO ErrHandler                       ! всегда проверяя ошибки
ADD(OrderHeader)                    !Добавить порождающую запись
DD ErrHandler                       ! всегда проверяя ошибки
LOOP X# = 1 TO RECORDS(Detail4ue)  !Процесс занесения порожденных записей
GET(Detail4ue,X#)                  ! Взять одну из очереди
DO ErrHandler                       ! проверяя ошибки
Det:Record ' Detail4ue              !поместить в буфер записи
ADD(OrderDetail)                    !и добавить в файл
DO ErrHandler                       ! проверив ошибки
.
COMMIT                              !Завершить успешную транзакцию

ErrHandler ROUTINE                  !Подпрограмма проверки ошибок
IF NOT ERRORCODE() THEN EXIT.      !Выйти, если нет ошибок
ROLLBACK                           !Откатить неуспешную транзакцию
.
BEEP !Оповестить пользователя
SHOW(25,1,'Transaction Error - ' & ERR4R())
ASK  ! и ждать его реакции
RETURN                             !затем выйти

```

Смотри также: LOGOUT, COMMIT

## Обработка фиктивных данных

“Фиктивное” значение в поле файла данных или виртуального файла означает, что пользователь совсем не ввел данные в это поле. Фиктивность реально означает, что значение поля “не известно”. Это в корне отличается от того, что поле имеет пробельное или нулевое значение, и делает возможным определить разницу между тем, что данные в поле не введены и тем, что поле имеет действительно пробельное или нулевое значение.

В выражении фиктивное значение не равно нулевому или пробельному. Поэтому любое выражение, в котором сравнивается значение поля файла данных или виртуального файла с каким-либо другим значением, всегда будет давать в результате фиктивное значение, если фиктивно значение сравниваемого поля. Это справедливо, даже если значения обоих сравниваемых элементов являются фиктивными. Например условное выражение `Pre:Field1 = Pre:Field2`, будет давать результат “истина” только, если оба поля содержат реальные, одинаковые значения. Если одно из полей содержит фиктивное значение, то результат выражения будет также фиктивным.

действительное = действительное	!Результат “Истина” или “Ложь”
действительное = фиктивное	!Результат фиктивен
фиктивное = фиктивное	!Результат фиктивен
фиктивное <> 10	!Результат фиктивен
1 + фиктивное	!Результат фиктивен

Четыре булевых выражения, использующие операции OR (ИЛИ) и AND (И), в которых только одна часть всего выражения представляет собой фиктивное значение, а вторая удовлетворяет приведенным ниже критериям, являются единственным исключением из вышеприведенных правил.

фиктивное OR Истина	!Результат Истина
Истина OR фиктивное	!Результат Истина
фиктивное AND Ложь	!Результат Ложь
Ложь AND фиктивное	!Результат Ложь

Поддержка фиктивных значений в файле данных или виртуальном файле целиком зависит от используемого файлового драйвера. Некоторые драйверы поддерживают концепцию фиктивных полей (SQL драйверы в большей части), другие - не поддерживают. Для того, чтобы определить поддерживает ли файловый драйвер фиктивные значения поле, используйте документацию на соответствующий драйвер.

## NULL (проверить значение поля на фиктивное)

### NULL(*поле*)

**NULL** определяет наличие фиктивного значения в поле.  
*поле* Метка (включая префикс) поля в структуре FILE или VIEW.

Функция **NULL** возвращает ненулевое значение (истина), если значение поля фиктивное, и возвращает ноль (ложь), если поле содержит реальное значение (включая пробелы и ноль). Поддержка фиктивных значений в файле данных или виртуальном файле целиком зависит от используемого файлового драйвера.

Тип возвращаемого значения: LONG

### Пример:

```
Customer  FILE,DRIVER('Clarion'),PRE(Cus) !Объявить структуру файла покупателей
AcctKey   KEY(Cus:AcctNumber)
Record    RECORD
AcctNumber
```

LONG

```

OrderNumber          LONG
Name                 STRING(20)
Addr                 STRING(20)
CSZ                  STRING(35)
..

Header               FILE,DRIVER('Clarion'),PRE(Hea) !Объявить структуру файла заголовков
AcctKey              KEY(Hea:AcctNumber)
Record              RECORD
AcctNumber           LONG
OrderNumber          LONG
ShipToName           STRING(20)
ShipToAddr           STRING(20)
ShipToCSZ            STRING(35)
..

CODE
OPEN(Header)
OPEN(Customer)
SET(Hea:AcctKey)
LOOP
NEXT(Header)
  IF ERRORCODE() THEN BREAK.
  IF NULL(Hea:ShipToName) !Проверить заполнен ли адрес доставки
    Cus:AcctNumber = Hea:AcctNumber
    GET(Customer,Cus:AcctKey) !Прочитать запись о покупателе
    IF ERRORCODE() THEN CLEAR(Cus:Record).
    Hea:ShipToName = Cus:Name ! и присвоить адрес покупателя
    Hea:ShipToAddr = Cus:Addr ! в качестве адреса поставки
    Hea:ShipToCSZ = Cus:CSZ
  END
  PUT(Header) !Записать обратно запись заголовка
END

```

## SETNULL (установить для поля фиктивное значение)

**SETNULL**(*поле*)

**SETNULL** Назначает полю фиктивное значение.  
*поле* Метка (включая префикс) поля в структуре FILE или VIEW.

Оператор **SETNULL** присваивает полю в файле данных или виртуальном файле фиктивное значение. Поддержка фиктивных значений в файле данных или виртуальном файле целиком зависит от используемого файлового драйвера.

Тип возвращаемого значения: LONG



**Пример:**

```

Customer  FILE,DRIVER('Clarion'),PRE(Cus)           !Объявить структуру файла покупателей
AcctKey    KEY(Cus:AcctNumber)
  Record   RECORD
AcctNumber                                LONG
OrderNumber                               LONG
Name      STRING(20)
Addr      STRING(20)
CSZ       STRING(35)

..
Header    FILE,DRIVER('Clarion'),PRE(Hea)           !Объявить структуру файла заголовков
AcctKey    KEY(Hea:AcctNumber)
OrderKey   KEY(Hea:OrderNumber)
Record     RECORD
AcctNumber                                LONG
OrderNumber                               LONG
ShipToName                                STRING(20)
ShipToAddr  STRING(20)
ShipToCSZ   STRING(35)

..
CODE
OPEN(Header)
OPEN(Customer)
SET(Hea:AcctKey)
LOOP
  NEXT(Header)
  IF ERRORCODE() THEN BREAK.
  Cus:AcctNumber = Hea:AcctNumber
  GET(Customer,Cus:AcctKey)      !Прочитать запись файла покупателей
  IF ERRORCODE() THEN CLEAR(Cus:Record).
  IF NOT NULL(Hea:ShipToName) AND Hea:ShipToName = Cus:Name
                                !Проверить адрес поставки
    SETNULL(Hea:ShipToName)      ! и присвоить фиктивное значение
    SETNULL(Hea:ShipToAddr)      ! полю "адрес поставки"
    SETNULL(Hea:ShipToCSZ)
  END
  PUT(Header)                    !Записать обратно запись файла заголовков
END

```

## SETNULL (установить нефиктивное значение поля)

**SETNULL(*поле*)**

<b>SETNONULL</b> <b>поле</b>	Присваивает полю реальное значение (пробельное или нулевое). Метка (включая префикс) поля в структуре FILE или VIEW.
---------------------------------	---

Оператор **SETNULL** присваивает полю в файле данных или виртуальном файле реальное значение (пробельное или нулевое). Поддержка фиктивных значений в файле данных или виртуальном файле целиком зависит от используемого файлового драйвера.

Тип возвращаемого значения: LONG

### Пример:

Customer	FILE, DRIVER('Clarion'), PRE(Cus)	!Объявить структуру файла покупателей
AcctKey	KEY(Cus:AcctNumber)	
Record	RECORD	
AcctNumber		LONG
OrderNumber		LONG
Name	STRING(20)	
Addr	STRING(20)	
CSZ	STRING(35)	
	..	
Header	FILE, DRIVER('Clarion'), PRE(Hea)	!Объявить структуру файла заголовков
AcctKey	KEY(Hea:AcctNumber)	
OrderKey	KEY(Hea:OrderNumber)	
Record	RECORD	
AcctNumber		LONG
OrderNumber		LONG
ShipToName		STRING(20)
ShipToAddr	STRING(20)	
ShipToCSZ	STRING(35)	
	..	
	CODE	
	OPEN(Header)	
	OPEN(Customer)	
	SET(Hea:AcctKey)	
	LOOP	
	NEXT(Header)	
	IF ERRORCODE() THEN BREAK.	
	Cus:AcctNumber = Hea:AcctNumber	
	GET(Customer, Cus:AcctKey)	!Прочитать запись файла покупателей
	IF ERRORCODE() THEN CLEAR(Cus:Record).	
	IF NULL(Hea:ShipToName) OR Hea:ShipToName = Cus:Name	
		!Проверить адрес поставки

```
Hea:ShipToName = 'Same as Customer Address'      !Пометить запись
SETNULL(Hea:ShipToAddr)                          ! и очистить "адрес поставки"
SETNULL(Hea:ShipToCSZ)
END

PUT(Header)      !Записать обратно запись файла заголовков
END
```

**Смотри также:** NULL, SETNULL

## ***Настройка специфики, связанной со страной***

### **Файлы переменных среды**

---

Файл, описывающий среду, содержит установки параметров, настраивающие приложение на специфику использования в конкретной стране. В начале выполнения программы, процедуры исполняющей системы пытаются обнаружить файл, описывающий среду, для данной программы, который имеет такое же как программа имя (appname.ENV) и расположен в том же каталоге. Если такого файла нет, то исполняющая система устанавливает по умолчанию стандартные установки на English/ASCII. Создав файл CW.ENV можно также использовать установки, связанные с национальной спецификой (Database Manager использует эти установки, когда выводит содержимое файлов данных).

Если переменная CLACHARSET установлена в OEM, то файл .ENV совместим с файлом .INI, используемым Clarion для DOS (и версии 3, и версии 3.1), потому что, вообще говоря, файл .INI подготавливается используя набор символов OEM ASCII, а не ANSI.

Для того, чтобы во время выполнения приложения изменить установки, связанные с национальной спецификой, загрузив файл, описывающий среду, можно использовать процедуру LOCALE. Эту же процедуру можно также использовать для установки значения отдельных элементов Среды. Поддержка особенностей конкретной страны зависит от используемого приложением файлового драйвера (от поддержки атрибута OEM), поэтому для выяснения всех связанных с этим вопросов используйте документацию на конкретный драйвер.

В файле установок среды можно задать следующие переменные:

**CLACHARSET=WINDOWS**  
**CLACHARSET=OEM**

Эта переменная определяет набор символов, используемый для элементов в файле .ENV. Если эта переменная опущена, то по умолчанию используется WINDOWS. Если для редактирования .ENV файла вы используете текстовый редактор в DOS или если этот файл должен быть совместим с Clarion для DOS, то указывайте OEM. В противном случае задавайте значение WINDOWS или опустите эту переменную. Эта переменная всегда должна быть первой переменной в файле установок среды.

**CLACOLSEQ=WINDOWS**  
**CLACOLSEQ="строка"**

Задаёт особую последовательность сортировки, используемую во время выполнения. Последовательность сортировки используется при построении ключевых и индексных файлов, а также при сортировке очередей и при всех сравнениях строк или символов.

Если используется значение **WINDOWS**, то последовательность сортировки, используемая по умолчанию, определяется установкой страны в Windows (в Панели управления). Если эта переменная в файле установок среды опущена, то по умолчанию используется последовательность сортировки в стандарте ANSI.

Используя значение **WINDOWS**, можно при упорядочении чередовать прописные и строчные буквы (AaBbCc ...), так что программа такого типа:

```
CASE SomeString[1]  
OF 'A' TO 'Z'
```

обработает также и буквы от 'а' до 'у' (буквы латинские - прим. перев). Если установлено значение **WINDOWS** (или другой отличный от умалчиваемого вида сортировки), то для такого типа проверок предпочтительнее использовать функции **ISUPPER** и **ISLOWER**.

Кроме значения **WINDOWS** можно указать строку символов (в двойных кавычках), чтобы прямо определить, какую последовательность сортировки следует использовать. В строку нужно включить только те символы, для которых надо указать новое положение при сортировке. Все остальные, не перечисленные символы остаются на своих прежних местах. Например, если задано **CLACOLSEQ="CA"** для стандартной последовательности английского алфавита (ABCD ...), то в результате последовательность станет "CBAD". В этом состоит отличие этого значения от версии Clarion для DOS, в которой требуется указать точно 222 символа (но обратная совместимость обеспечивается).

Замечание. Читать и писать файлы следует, всегда используя одну и ту же последовательность сортировки. Применение различных последовательностей может привести к тому, что ключи будут не отсортированы, а записи станут недоступны. Задание **CLACOLSEQ = WINDOWS** подразумевает, что последовательность сортировки может изменить пользователь, изменив страну в Управляющей панели Windows.

**CLAAMPM=WINDOWS**

**CLAAMPM="АМстрока", "РМстрока"**

Эта переменная задает текст, используемый для обозначения времени суток ("до полудня" или "после полудня") как части высвечиваемого времени. Значение **WINDOWS** указывает, что используется соответствующая установка, сделанная в Управляющей панели Windows. Значения **АМстрока** и **РМстрока** те же самые, что и в Clarion для DOS за исключением того, что учитывается значение переменной **CLACHARSET**.

**CLAMONTH="Месяц1", " Месяц2", ... , " Месяц12"**

Задает текст, возвращаемый функциями и шаблонами форматирования,

представляющий собой полное название месяцев.

**CLAMON=" СокрМесяц1", " СокрМесяц2", ... , " СокрМесяц12"**

Задаёт текст, возвращаемый функциями и шаблонами форматирования, представляющий собой сокращённое название месяцев.

**CLADIGRAPH="ДиграфСимв1Симв2, ... "**

Позволяет правильно сортировать диграфы. Диграф - это логически единый символ, представляющий собой сочетание двух символов (Симв1 and Симв2). Диграфы сортируются как два символа, их составляющие. Чаще всего диграфы имеются в других языках, не в английском. Например,

**CLADIGRAPH="(Ae,(ae"** задаёт, слово **"J(ger"** при сортировке стоит перед словом **"Jager"** (поскольку **"Jae"** идёт перед **"Jag"**).

Можно определить несколько сочетаний ДиграфСимв1Симв2, разделённых запятыми. При этом учитывается значение переменной CLACHARSET.

**CLACASE=WINDOWS**

**CLACASE="СтрокаПрописных", "СтрокаСтрочных"**

Позволяет задать пары из прописной и строчной букв.

Значение WINDOWS подразумевает использование таких пар, определённых установкой конкретной страны в Windows (в Управляющей панели). Если эта переменная в файле .ENV опущена, то используются не принятые в Windows пары, а задаваемые стандартом ANSI.

Параметры СтрокаПрописных и СтрокаСтрочных задают набор прописных букв и соответствующие им строчные буквы. Длины этих параметров должны быть равны. При установке этой переменной следует учитывать значение переменной CLACHARSET. На ANSI символы с кодами меньше 127 эта переменная не влияет.

**CLABUTTON="OK", "&Yes", "&No", "&Abort", "&Ignore", "&Retry", "Cancel", "&Help"**

Эта переменная определяет тексты, используемые кнопками, выполняющими функции сообщений. Тексты задаются в виде списка разделённых запятыми строк в следующем порядке: OK, YES, NO, ABORT, RETRY, IGNORE, CANCEL, HELP. По умолчанию используются тексты, приведённые выше.

**CLAMSGномер\_ошибки="Сообщение об ошибке"**

Customer	FILE,DRIVER('Clarion'),PRE(Cus)	!Объявить файл без атрибута OEM
AcctKey	KEY(Cus:AcctNumber)	
Record	RECORD	
AcctNumber		LONG
OrderNumber		LONG
Name	STRING(20)	

```

Addr      STRING(20)
..
Win        WINDOW,SYSTEM
           STRING(@s20),USE(Cus:Name)
           END
           CODE
           OPEN(Customer)
           SET(Customer)
           NEXT(Customer)
           CONVERTOEMTOANSI(Cus:Record)  !Преобразовать все стоки из ASCII в ANSI
           OPEN(Win)
           ACCEPT
           !Обработать объекты окна
           END
           CONVERTANSITOOEM(Cus:Record)  !Преобразовать обратно из ANSI в ASCII
           PUT(Customer)

```

Смотри также:      CONVERTOEMTOANSI

## CONVERTOEMTOANSI (преобразовать код ASCII в ANSI)

**CONVERTOEMTOANSI**( строка )

**CONVERTOEMTOANSI**      Преобразует строки из кодировки OEM ASCII в ANSI.

*строка*                      Имя строки, которую надо преобразовать. Это может быть отдельная переменная или любая структура, которая рассматривается как группа (RECORD, QUEUE, etc.).

Оператор **CONVERTOEMTOANSI** преобразует отдельную строку или строки внутри группы из кодировки OEM (набор символов ASCII, расширенный символами, определенными активной кодовой страницей) в кодировку ANSI (отображаемую в Windows).

Эта процедура не требуется для файлов, для которых установлен атрибут OEM.

### Пример:

```

Customer  FILE,DRIVER('Clarion'),PRE(Cus)      ! Объявить файл без атрибута OEM
AcctKey    KEY(Cus:AcctNumber)
Record     RECORD
AcctNumber                                LONG
OrderNumber                                LONG
Name       STRING(20)
Addr       STRING(20)
..

```



```

Win      WINDOW,SYSTEM
        STRING(@s20),USE(Cus:Name)
        END
        CODE
        OPEN(Customer)
        SET(Customer)
        NEXT(Customer)
        CONVERTOEMTOANSI(Cus:Record)  !преобразовать все строки из ASCII в ANSI
        OPEN(Win)
        ACCEPT
        !Обработать объекты окна
        END
        CONVERTANSITOOEM(Cus:Record)  !Преобразовать обратно из ANSI в ASCII
        PUT(Customer)

```

Смотри также:      CONVERTANSITOOEM

## ISALPHA (проверить символ на букву)

**ISALPHA**( строка )

### ISALPHA

*строка*

Проверяет содержит ли переданная ей строка букву.  
Имя строки символов, которую надо проверить. Если параметр строка состоит более чем из одного символа, то проверяется только первый символ.

Функция **ISALPHA** возвращает значение ИСТИНА, если переданная ей строка представляет собой букву алфавита (прописную или строчную), а противном случае она возвращает значение ЛОЖЬ. Работа этой функции не зависит от установленного языка и последовательности сортировки.

Тип возвращаемого значения:      LONG

### Пример:

```

SomeString STRING(1)
CODE
SomeString = 'A'                !Функция ISALPHA возвратит значение ИСТИНА
IF ISALPHA(SomeString)
    X# = MESSAGE('Буква')
END
SomeString = '1'                !Функция ISALPHA возвратит значение ЛОЖЬ
IF ISALPHA(SomeString)
    X# = MESSAGE('Буква')
ELSE
    X# = MESSAGE('Не буква')
END

```

Смотри также: ISUPPER, ISLOWER

## ISLOWER (проверить символ на строчную букву)

**ISLOWER**( строка )

**ISLOWER**  
*строка*                      Проверяет, содержит ли переданная ей строка строчную букву.  
Имя строки символов, которую надо проверить. Если параметр строка состоит более чем из одного символа, то проверяется только первый символ.

Функция **ISLOWER** возвращает значение ИСТИНА, если переданная ей строка представляет собой строчную букву, противном случае она возвращает значение ЛОЖЬ. Работа этой функции не зависит от установленного языка и последовательности сортировки.

Тип возвращаемого значения: LONG

### Пример:

```
SomeString STRING(1)
CODE
SomeString = 'a'           !Функция ISLOWER возвратит значение ИСТИНА
IF ISLOWER(SomeString)
    X#= MESSAGE('Строчная буква')
END
SomeString = 'A'           ! Функция ISLOWER возвратит значение ЛОЖЬ
IF ISLOWER(SomeString)
    X#= MESSAGE('Строчная буква')
ELSE
    X#= MESSAGE('Не строчная буква')
END
```

Смотри также: ISUPPER, ISALPHA

## ISUPPER (проверить символ на прописную букву)

**ISUPPER**( строка )

**ISUPPER**  
*строка*                      Проверяет, содержит ли переданная ей строка прописную букву.  
Имя строки символов, которую надо проверить. Если параметр строка состоит более чем из одного символа, то проверяется только первый символ.

Функция **ISUPPER** возвращает значение ИСТИНА, если переданная ей строка представляет собой прописную букву, противном случае она возвращает значение ЛОЖЬ.

Работа этой функции не зависит от установленного языка и последовательности сортировки.

Тип возвращаемого значения: LONG

### Пример:

```
SomeString STRING(1)
CODE
SomeString = 'A' !Функция ISUPPER возвратит значение ИСТИНА
IF ISUPPER(SomeString)
X# = MESSAGE('Прописная буква')
END
SomeString = 'a' !Функция ISUPPER возвратит значение ЛОЖЬ
IF ISUPPER(SomeString)
X# = MESSAGE('Прописная буква')
ELSE
X# = MESSAGE('Не прописная буква')
END
```

Смотри также: ISLOWER, ISALPHA

## LOCALE (загрузить файл с переменными среды)

LOCALE(	файл	)
	переменная, значение	

**LOCALE** Позволяет во время выполнения приложения загрузить конкретный файл переменных среды (.ENV) и кроме того, и к тому же установить значения отдельных переменных.

*файл* Строковая константа или переменная, содержащая имя (с расширением) файла переменных среды, который следует загрузить или ключевое слово WINDOWS. Это может быть и полный путь в файловой системе DOS.

*переменная* Строковая константа или переменная, содержащая имя переменной среды, значение которой нужно установить. Допустимые значения перечисляются в разделе “Файлы переменных среды”

*значение* Строковая константа или переменная, которая содержит значение переменной среды.

Процедура **LOCALE** Позволяет во время выполнения приложения загрузить конкретный файл переменных среды (.ENV) и кроме того, и к тому же установить значения отдельных переменных. Таким образом можно в приложении загрузить другой файл переменных Среды, отличный от используемого по умолчанию файла arpname.ENV, или установить значения отдельный переменных, если файла переменный среды не существует.

Ключевое слово **WINDOWS**, использованное в качестве параметра файл, указывает, что для переменных **CLACOLSEQ**, **CLACASE** и **CLAAMP** используются принятые по умолчанию в Windows значения. При присвоении значений отдельным переменным для параметра значение не требуются двойные кавычки, в отличие от синтаксиса, принятого для файла переменных среды.

Сообщения об ошибках:

- 02 File Not Found (файл не найден)
- 05 Access Denied (доступ запрещен)

### Пример:

```

LOCALE('MY.ENV')           !Загрузить файл переменных среды
LOCALE('WINDOWS')         !Установить значения CLACOLSEQ, CLACASE и
CLAAMP                    ! используемые в Windows
LOCALE('CLABUTTON','OK,&Si,&No,&Abortar,&Ignora,&Volveratratar,Cancelar,&Ayuda')
                           !Установить испанские надписи на кнопках
LOCALE('CLACOLSEQ','АБВГДЕЖЗЙИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдежзйиийклмннопрпсстфхцчшщъыьэюя')
                           !Установить последовательность сортировки
LOCALE('CLACASE','ДЕЖЗЙСЦЬ,дежзйсть') !Установить пары прописных/строчных букв
LOCALE('CLAMSG2','No File Found')      !Установить текст сообщения для ERRORCODE(=2

```

**Смотри также:**      Файлы параметров среды

## Глава 12 Виртуальные файлы

### Структуры для организации виртуального файла

#### VIEW (объявить “виртуальный” файл)

<i>метка</i>	<b>VIEW</b> ( <i>первичный файл</i> ) [ <b>FILTER</b> ( )] [ <b>ORDER</b> ( )] [ <b>PROJECT</b> ( )] [ <b>JOIN</b> ( ) [ <b>PROJECT</b> ( )] [ <b>JOIN</b> ( ) [ <b>PROJECT</b> ( )] <b>END</b> ] <b>END</b> ] <b>END</b>
--------------	---

<b>VIEW</b>	Объявляет “виртуальный” файл, как агрегат из связанных файлов.
<i>метка</i>	Имя виртуального файла.
<i>первичный файл</i>	Метка первичной структуры FILE для виртуального файла.
<b>FILTER</b>	Объявляет выражение, используемое для отбора записей в виртуальный файл.
<b>ORDER</b>	Объявляет выражение или список выражений, используемых для того, чтобы определить порядок сортировки записей виртуального файла.
<b>PROJECT</b>	Указывает поля из первичного файла или из связанного файла, указанного структурой JOIN, которые будут входить в состав виртуального файла. Если этот атрибут опущен, то доступны все поля.
<b>JOIN</b>	Объявляет вторичный, связанный файл.

Структура **VIEW** объявляет виртуальный файл, как агрегат из связанных файлов. Так как структура **VIEW** представляет собой логическую конструкцию, элементы данных, объявленные в ней физически в этой структуре не существуют. **VIEW** представляет собой особый способ обращения к данным физически располагающимся в нескольких связанных структурах **FILE**. Во время выполнения программы структуре **VIEW** не выделяется память для буфера данных, так что поля, используемые в ней размещаются в буферах записей соответствующих структур **FILE**.

Перед использованием структура **VIEW** должна быть явным образом открыта, а предварительно должны быть открыты все файлы: и первичный и вторичные. В то время, когда открыта структура **VIEW**, не допускаются никакие файловые операции ввода/вывода, которые непосредственно оперируют с первичным или связанными в структуре **VIEW** файлами.

Структура VIEW позволяет осуществлять только последовательный доступ к данным. Для того, чтобы установить последовательность обработки и начальную точку, перед открытием структуры VIEW должен быть выполнен оператор SET для первичного файла этой структуры, затем операторы NEXT(view) и PREVIOUS(view) осуществляют последовательный доступ к данным. Со структурой VIEW допустим еще оператор REGET, но только для того, чтобы задать записи, которые должны быть текущими в соответствующих буферах первичного и связанных файлов после закрытия структуры VIEW. Если непосредственно перед закрытием структуры VIEW не выполнялся оператор REGET, то устанавливается, что буферы записей первичного и связанных файлов не содержат текущих записей. Последовательность обработки первичного и связанных файлов после закрытия структуры VIEW неопределена. Поэтому, если необходимо после закрытия структуры VIEW устанавливать последовательность обработки, для этого нужно использовать операторы SET или RESET.

Структура данных VIEW предназначена для того, чтобы облегчить доступ к базе данных в системах с архитектурой клиент-сервер. Она выполняет две реляционные операции одновременно: реляционные операции “соединение” и “проекция”. В клиент-серверных системах эти операции выполняются на файловом сервере, а клиенту пересылается только результат. Это может коренным образом улучшить производительность сетевой прикладной программы.

Реляционная операция “соединение” выбирает данные из нескольких файлов основываясь на определенных связях между файлами. Операцию “соединение” определяет структура JOIN в структуре VIEW. В структуре VIEW может быть несколько структур JOIN, и они могут быть вложенными одна в другую, выполняя многоуровневую операцию соединения.

Реляционная операция “проекция” делает доступными только указанные элементы данных из соответствующих файлов, а не всю запись. Доступны только те поля, которые явно объявлены в операторах PROJECT. Таким образом структурой VIEW реляционная операция “проекция” реализуется автоматически. Содержимое полей, которые не описаны в PROJECT, не определено.

Атрибут FILTER ограничивает виртуальный файл подмножеством записей. Выражение в FILTER может включать любые поля, явно объявленные в структуре VIEW, и накладывает ограничения на виртуальный файл, основываясь на содержимом этих полей. Таким образом атрибут FILTER работает на всех уровнях операции соединения.

### Пример:

```
Customer  FILE,DRIVER('Clarion'),PRE(Cus) !Объявить структуру файла покупателей
AcctKey   KEY(Cus:AcctNumber)
Record                                         RECORD
```

```

AcctNumber          LONG
OrderNumber         LONG
Name                STRING(20)
Addr                STRING(20)
City                STRING(20)
State               STRING(20)
Zip                 STRING(20)

..
Header              FILE,DRIVER('Clarion'),PRE(Hea) !Объявить структуру файла заголовков
AcctKey              KEY(Hea:AcctNumber)
OrderKey             KEY(Hea:OrderNumber)
Record              RECORD
AcctNumber           LONG
OrderNumber          LONG
ShipToName           STRING(20)
ShipToAddr           STRING(20)
ShipToCity           STRING(20)
ShipToState          STRING(20)
ShipToZip            STRING(20)

..
Detail              FILE,DRIVER('Clarion'),PRE(Dtl) !Объявить структуру файла накладных
OrderKey             KEY(Dtl:OrderNumber)
Record              RECORD
OrderNumber          LONG
Item                 LONG
Quantity            SHORT

..
Product             FILE,DRIVER('Clarion'),PRE(Pro) !Объявить структуру файла товаров
ItemKey              KEY(Pro:Item)
Record              RECORD
Item                 LONG
Description           STRING(20)
Price                DECIMAL(9,2)

..
ViewOrder           VIEW(Customer) !Объявить структуру VIEW
                    PROJECT(Cus:AcctNumber,Cus:Name)
                    JOIN(Hea:AcctKey,Cus:AcctNumber) !Соединить с файлом заголовков
                    PROJECT(Hea:OrderNumber)
                    JOIN(Dtl:OrderKey,Hea:OrderNumber) !Соединить с файлом накладных
                    PROJECT(Det:Item,Det:Quantity)
                    JOIN(Pro:ItemKey,Dtl:Item) !Соединить с файлом товаров
                    PROJECT(Pro:Description,Pro:Price)
                    END
                    END
                    END
                    END

```

**FILTER (установить ограничивающее выражение)****FILTER** (*выражение*)

**FILTER**                      Задает выражение, используемое для оценки того, включать ли запись в виртуальный файл.

*выражение*                      Строковая константа, содержащая логическое выражение.

Атрибут **FILTER** Задает выражение, используемое для оценки того, включать ли запись в виртуальный файл.

В выражении могут иметься ссылки на любые поля из структуры VIEW из структур JOIN любого уровня. Для того, чтобы запись была включена в виртуальный файл, все выражение в целом должно быть истиной. Во время выполнения программы выражение вычисляется (совсем также как функцией EVALUATE), поэтому все входящие в выражение переменные должны быть указаны оператором BIND.

**Пример:**

```
!Взять только заказы начиная с 100-го для покупателя 9999
ViewOrder VIEW(Customer),FILTER('Cus:AcctNumber = 9999 AND Hea:OrderNumber > 100')
    PROJECT(Cus:AcctNumber,Cus:Name)
    JOIN(Hea:AcctKey,Cus:AcctNumber)      !Соединить с файлом заголовков
    PROJECT(Hea:OrderNumber)
    JOIN(Dtl:OrderKey,Hea:OrderNumber)    !Соединить с файлом накладных
    PROJECT(Det:Item,Det:Quantity)
    JOIN(Pro:ItemKey,Dtl:Item) !Соединить с файлом товаров
    PROJECT(Pro:Description,Pro:Price)
    END
    END
    END
    END
    CODE
    OPEN((Customer,22h)
    OPEN((Header,22h)
    OPEN((Product,22h)
    OPEN(Detail,22h)
    BIND('Cus:AcctNumber',Cus:AcctNumber)
    BIND('Hea:AcctNumber',Hea:AcctNumber)
    SET(Cus:AcctKey)
    OPEN(ViewOrder)
    LOOP
        NEXT(ViewOrder)
        IF ERRORCODE() THEN BREAK.
        !Обработать подошедшие записи
    END
```



```

UNBIND('Cus:AcctNumber',Cus:AcctNumber)
UNBIND('Hea:AcctNumber',Hea:AcctNumber)
CLOSE(Header)
CLOSE(Customer)
CLOSE(Product)
CLOSE(Detail)

```

Смотри также: BIND, UNBIND, EVALUATE

## ORDER (выражение, определяющее порядок сортировки)

**ORDER**(*список\_выражений*)

**ORDER**                    Задает список выражений, используемый при сортировке записей виртуального файла.

*список\_выражений*        Строковая константа содержащая одно или более выражений. Каждое выражение в списке должно отделяться запятой.

Атрибут **ORDER** задает список выражений, используемый при сортировке записей виртуального файла. Выражения в списке вычисляются слева направо; приоритет сортировки убывает слева направо. Выражений, которому предшествует знак минус (-) определяет убывающую последовательность сортировки.

В выражении могут быть ссылки на любые поля структуры VIEW с любых уровней структур JOIN. Выражения в списке могут содержать в себе любые допустимые в языке Clariion выражения. Во время выполнения программы выражение вычисляется (совсем также как функцией EVALUATE), поэтому все входящие в выражение переменные должны быть указаны оператором BIND.

### Пример:

! Порядок сортировки: по убыванию даты, затем по фамилии покупателя (в рамках каждой даты

```

ViewOrder VIEW(Customer),ORDER('-Hea:OrderDate,Cus:Name')
        PROJECT(Cus:AcctNumber,Cus:Name,Cus:Zip)
        JOIN(Hea:AcctKey,Cus:AcctNumber)      !Присоединить файл Header
        PROJECT(Hea:OrderNumber,Hea:OrderDate)
        JOIN(Dtl:OrderKey,Hea:OrderNumber) ! Присоединить файл Detail
        PROJECT(Det:Item,Det:Quantity)
        JOIN(Pro:ItemKey,Dtl:Item) ! Присоединить файл Product
        PROJECT(Pro:Description,Pro:Price)
        END
    END
END
END

```

!Порядок сортировки по величине скидки внутри каждой даты

```
ViewOrd2  VIEW(Customer),ORDER(' -Hea:OrderDate,Pro:Price-Det:DiscountPrice')
          PROJECT(Cus:AcctNumber,Cus:Name,Cus:Zip)
          JOIN(Hea:AcctKey,Cus:AcctNumber)      ! Присоединить файл Header
          PROJECT(Hea:OrderNumber,Hea:OrderDate)
          JOIN(Dtl:OrderKey,Hea:OrderNumber)    ! Присоединить файл Detail
          PROJECT(Det:Item,Det:Quantity,Det:DiscountPrice)
          JOIN(Pro:ItemKey,Dtl:Item) ! Присоединить файл Product
          PROJECT(Pro:Description,Pro:Price)
          END
        END
      END
    END
```

Смотри также: BIND, UNBIND, EVALUATE

## PROJECT (задать поля виртуального файла)

### PROJECT (поля)

#### PROJECT

*поля*

Объявляет поля, доступные в структуре VIEW.

Разделенный запятыми список полей (включая префиксы) из первичного файла или связанного файла, обозначенного в структуре JOIN, содержащей данный оператор PROJECT.

Оператор PROJECT объявляет в структуре VIEW поля выбираемые в реляционной операции “проекция”. Эта операция делает доступными только указанные поля файла, а не целую запись. Доступны только те поля, которые явно объявлены в операторах PROJECT в структуре VIEW.

Оператор PROJECT может указываться в структуре VIEW или внутри одного из ее компонентов - в структуре JOIN. Если в структуре VIEW и в ее структурах JOIN нет операторов PROJECT, то доступны все поля из файлов, составляющих виртуальный файл.

#### Пример:

```
Detail      FILE,DRIVER('Clarion'),PRE(Dtl)  !Объявить файл накладных
OrderKey    KEY(Dtl:OrderNumber)
Record      RECORD
OrderNumber                                LONG
Item        LONG
Quantity    SHORT
Description  STRING(20)                    !Строка описания элемента
..
```

```

Product      FILE,DRIVER('Clarion'),PRE(Pro)  !Объявить файл товаров
ItemKey      KEY(Pro:Item)
Record       RECORD
Item         LONG
Description   STRING(20)                      !Описание товара
Price        DECIMAL(9,2)
..

ViewOrder    VIEW(Detail)
              PROJECT(Det:OrderNumber,Det:Item,Det:Description)
              JOIN(Pro:ItemKey,Det:Item)
              PROJECT(Pro:Description,Pro:Price)
              END
            END

```

### JOIN (объявить операцию “соединения”)

```

JOIN(  |вторичный ключ, связывающие поля) |
      |вторичный файл, выражение          | )
      [PROJECT( )]
      [JOIN( )]
      [PROJECT( )]
      END]
END

```

**JOIN**                      Объявляет вторичный файл для реляционной операции “соединения”.

*вторичный ключ*            Метка оператора KEY, которая определяет вторичный файл и ключ доступа к нему.

*связывающие поля*        Разделяемый запятыми список полей из связанного файла, которые содержат значения вторичного ключа, используемого для доступа к записям.

*вторичный файл,*        Метка вторичного файла.

*выражение*                Строковая константа, содержащая единое логическое выражение для объединения файлов. Это выражение может содержать любые логические и булевы операторы.

**PROJECT**                Задаёт поля из вторичного файла, указанного структурой JOIN, которые будут доступны в структуре VIEW. Если этот оператор опущен, то доступны все поля файла.

Структура **JOIN** объявляет вторичный файл для реляционной операции “соединения”. Реляционная операция “соединения”, основываясь на определенных между файлами связях, выбирает данные из нескольких файлов. В структуре VIEW может быть несколько структур JOIN, и они могут быть вложенными одна в другую, выполняя многоуровневую операцию соединения.

Вторичный ключ определяет ключ доступа к вторичному файлу. Связывающие поля обозначают поля в файле, с которым связан вторичный файл, поля, которые содержат данные используемые для доступа к связанным записям. Для структуры JOIN, находящейся непосредственно в структуре VIEW это поля первичного файла. Для структуры JOIN, вложенной в другую структуру JOIN, эти поля выбираются из вторичного файла, описанного структурой JOIN, в которую она вложена. Во вторичном ключе допустимы поля не относящиеся к связи между файлами, поскольку эти поля имеются в списке компонент ключа после всех связывающих полей.

Если при выборке данных для записи первичного файла во вторичном, связанном файле нет соответствующей записи, то полям, указанным в операторе PROJECT для этого файла присваиваются нулевые (пробельные) значения. Этот тип реляционной операции “соединения” известен как внешнее соединение. Для реализации других форм реляционной операции “соединения” можно использовать атрибут FILTER.

### Пример:

```
Customer  FILE,DRIVER('Clarion'),PRE(Cus) !Объявить структуру файла покупателей
AcctKey   KEY(Cus:AcctNumber)
Record    RECORD
AcctNumber      LONG
OrderNumber     LONG
Name            STRING(20)
Addr            STRING(20)
City            STRING(20)
State           STRING(20)
Zip             STRING(20)
..
Header      FILE,DRIVER('Clarion'),PRE(Hea) !Объявить структуру файла заголовков
AcctKey     KEY(Hea:AcctNumber)
OrderKey    KEY(Hea:AcctNumber,Hea:OrderNumber)
Record      RECORD
AcctNumber  LONG
OrderNumber LONG
Total       DECIMAL(11,2) !Всего оплачено наличными
Discount    DECIMAL(11,2) !Величина данной ссудки
OrderDate   LONG
..
Detail      FILE,DRIVER('Clarion'),PRE(Dtl) !Объявить структуру файла накладных
OrderKey    KEY(Dtl:AcctNumber,Dtl:OrderNumber)
Record      RECORD
AcctNumber  LONG
OrderNumber LONG
Item        LONG
Quantity    SHORT
```

..

```

Product      FILE, DRIVER('Clarion'), PRE(Pro)  !Объявить структуру файла товаров
ItemKey      KEY(Pro:Item)
Record       RECORD
Item         LONG
Description   STRING(20)
Price        DECIMAL(9,2)

```

..

```

ViewOrder1   VIEW(Header)                      !Объявить структуру VIEW
              PROJECT(Hea:AcctNumber,Hea:OrderNumber)
              JOIN(Dtl:OrderKey,Hea:AcctNumber,Hea:OrderNumber) !Присоединить файл

```

```

Detail
              PROJECT(Dtl:ItemDtl:Quantity)
              JOIN(Pro:ItemKey,Dtl:Item) !Присоединить файл Product
              PROJECT(Pro:Description,Pro:Price)
              END
            END
          END

```

```

ViewOrder2   VIEW(Customer)                    !Объявить структуры VIEW
              JOIN(Header,'Cus:AcctNumber = Hea:AcctNumber AND ' & |
                  ' (Hea:Discount + Hea:Total) * .1 > Hea:Discount')
              PROJECT(Hea:AcctNumber,Hea:OrderNumber)
              JOIN(Dtl:OrderKey,Hea:AcctNumber,Hea:OrderNumber) ! Присоединить файл

```

```

Detail
              PROJECT(Dtl:ItemDtl:Quantity)
              JOIN(Pro:ItemKey,Dtl:Item) ! Присоединить файл Product
              PROJECT(Pro:Description,Pro:Price)
              END
            END
          END

```

```

END

```

## Операторы для работы с виртуальным файлом

### CLOSE (закрыть виртуальный файл)

**CLOSE**(*view*)

**CLOSE**           Закрыть виртуальный файл.  
*view*           Метка виртуального файла.

Оператор **CLOSE** закрывает виртуальный файл. Виртуальный файл, объявленный в процедуре, неявно закрывается при завершении процедуры, если он не был закрыт явным образом.

Если непосредственно перед оператором **CLOSE** не стоит оператор **REGET**, то для первичного и связанных файлов устанавливается состояние “нет текущей записи”. Это означает, что значение буферов записей файлов не определено, а перед последовательной обработкой файла нужно выполнить оператор **SET** или **RESET**.

#### Пример:

```
Customer  FILE,DRIVER('Clarion'),PRE(Cus) !Объявить структуру файла покупателей
AcctKey   KEY(Cus:AcctNumber)
Record    RECORD
AcctNumber                LONG
OrderNumber               LONG
Name                     STRING(20)
Addr                     STRING(20)
City                     STRING(20)
State                    STRING(20)
Zip                      STRING(20)

ViewCust  ..
          VIEW(Customer)                !Объявить структуру VIEW
          PROJECT(Cus:AcctNumber,Cus:Name)
          END
          CODE
          OPEN(Customer,22h)
          SET(Cus:AcctKey)
          OPEN(ViewCust)                !Открыть виртуальный файл
          !исполняемые операторы
          CLOSE(ViewCust)               ! и снова закрыть его
```

## OPEN (открыть виртуальный файл)

### OPEN(*view*)

**OPEN**            Открыть виртуальный файл.  
*view*            Метка виртуального файла.

Оператор **OPEN** открывает для обработки структуру VIEW. Перед тем, как можно будет работать с виртуальным файлом, его нужно явно открыть. Файлы, используемые в структуре VIEW должны быть уже открыты.

Для того, чтобы установить последовательную обработку виртуального файла (последовательность и начальную точку) непосредственно перед оператором OPEN нужно выполнить оператор SET для первичного файла. В то время, когда виртуальный файл открыт, нельзя выполнить оператор SET; сначала нужно закрыть виртуальный файл, затем выполнить оператор SET перед последующим OPEN(*view*).

### Пример:

```
Header      FILE,DRIVER('Clarion'),PRE(Hea) !Объявить структуру файла заголовков
AcctKey      KEY(Hea:AcctNumber)
OrderKey     KEY(Hea:OrderNumber)
Record       RECORD
AcctNumber   LONG
OrderNumber  LONG
ShipToName   STRING(20)
ShipToAddr   STRING(20)
..
Detail       FILE,DRIVER('Clarion'),PRE(Dtl) !Объявить структуру файла накладных
OrderKey     KEY(Dtl:OrderNumber)
Record       RECORD
OrderNumber  LONG
Item         LONG
Quantity     SHORT
..
Product      FILE,DRIVER('Clarion'),PRE(Pro) !Объявить структуру файла товаров
ItemKey      KEY(Pro:Item)
Record       RECORD
Item         LONG
Description  STRING(20)
Price        DECIMAL(9,2)
..
ViewOrder    VIEW(Header) !Объявить структуру VIEW
              PROJECT(Hea:OrderNumber)
              JOIN(Dtl:OrderKey,Hea:OrderNumber) !Соединить с файлом накладных
              PROJECT(Det:Item,Det:Quantity)
```

```

JOIN(Pro:ItemKey,Dtl:Item) !Соединить с файлом товаров
PROJECT(Pro:Description,Pro:Price)
...
CODE
OPEN((Header,22h)
OPEN(Detail,22h)
OPEN(Product,22h)
SET(Hea:AcctKey)
OPEN(ViewOrder)           !Открыть виртуальный файл

```

## DELETE (удалить запись первичного в структуре VIEW файла)

### DELETE(*view*)

**DELETE** Удаляет запись первичного файла.  
*view* Метка виртуального файла.

Оператор **DELETE** запись первичного файла для структуры VIEW, к которой было последнее обращение оператором NEXT или PREVIOUS. Значение ключа для этой записи тоже удаляется из файла ключей. Из какого-либо вторичного, связанного в структуре JOIN файла оператор DELETE записи не удаляет.

Оператор DELETE удаляет запись только первичного файла потому, что структура VIEW выполняет одновременно и проекцию и соединение. И в этом случае возможно создать такую структуру VIEW, что обновление всех ее компонент нарушило бы правила внутренней непротиворечивости базы данных. Общее решение этой проблемы в SQL-ориентированных продуктах - это обновление только первичного файла. Поэтому в Clarion реализуется такое же стандартное решение.

Если предварительной выборки записи не было, или она заблокирована другой рабочей станцией, то выдается сообщение "Record Not Available", а запись не удаляется. Конкретные действия, выполняемые на диске оператором DELETE, зависят от файлового драйвера.

#### Выдаваемые сообщения об ошибках:

05 Access Denied (доступ к файлу запрещен)  
 37 Record Not Available (запись недоступна))

#### Пример:

```

Customer  FILE,DRIVER('Clarion'),PRE(Cus) !Объявить структуру файла покупателей
AcctKey    KEY(Cus:AcctNumber)
Record     RECORD
AcctNumber LONG
Name       STRING(20)

```



```

Addr                                STRING(20)
City                                STRING(20)
State                               STRING(20)
Zip                                 STRING(20)

CustView  ..
VIEW(Customer)                      !Объявить структуру VIEW
PROJECT(Cus:AcctNumber,Cus:Name)
END
CODE
OPEN(Customer)
Cus:AcctNumber = 12345              !Присвоить значение полю ключа
SET(Cus:AcctKey,Cus:AcctKey)
OPEN(CustView)
NEXT(CustView)                      !Прочитать запись
IF ERRORCODE() THEN STOP(ERROR()).
DELETE(CustView)                    !Удалить запись о покупателе
    
```

Смотри также: HOLD, NEXT, PREVIOUS, PUT

## HOLD (исключительный доступ к записи виртуального файла)

**HOLD**(*вирт. файл*[,*секунд*])

**HOLD** Включает блокировку записей.  
*вирт. файл* Метка виртуального файла.  
*секунд* Числовая константа или переменная, которая задает максимальное время ожидания в секундах.

Оператор **HOLD** включает блокировку записей первичного файла структуры VIEW в многопользовательской среде. Последующее выполнение операторов NEXT и PREVIOUS помечает запись первичного файла как “заблокированную” при условии, что запись прочитана успешно. Обычно блокировка записи исключает изменение данных в записи другими пользователями, хотя и не запрещает чтение записи. Конкретные действия, выполняемые оператором HOLD, зависят от файлового драйвера.

**HOLD**(вирт. файл) Включает процесс безусловного захвата записей; последующие операторы GET, NEXT и PREVIOUS до тех пор предпринимают попытки захватить запись, пока попытка не завершится успешно. Если запись захвачена другой рабочей станцией, то попытки захватить запись будут продолжаться до тех пор, пока другой пользователь не освободит ее.

**HOLD**(вирт. файл,секунд) Включает условный процесс удержания т.е. после безуспешных попыток захватить запись в течение заданного числа секунд операторы GET, NEXT и PREVIOUS выдают сообщение об ошибке “Record Is Already Held” (запись уже кем-то захвачена).

За один раз пользователь может заблокировать только одну запись в структуре VIEW. Если в этом же файле нужно обратиться ко второй записи, предыдущая запись должна быть освобождена (см RELEASE).

Как и при блокировке файлов, при захвате записей существует ситуация, которую следует избегать, - взаимная блокировка. Эта ситуация возникает, когда две рабочие станции пытаются захватить один и тот же набор записей в разной последовательности и обе используют форму HOLD(файл) этого оператора. Одна станция уже захватила запись одного файла и пытается захватить запись другого, а другая рабочая станция, наоборот. Этой проблемы можно избежать, используя форму HOLD(файл, секунд) и отслеживая ситуацию "Record Is Already Held".

### Пример:

ViewOrder	VIEW(Customer)	!Объявить структуру VIEW
	PROJECT(Cus:AcctNumber,Cus:Name)	
	JOIN(Hea:AcctKey,Cus:AcctNumber)	!Соединить с файлом заголовков
	PROJECT(Hea:OrderNumber)	
	JOIN(Dtl:OrderKey,Hea:OrderNumber)	!Соединить с файлом накладных
	PROJECT(Det:Item,Det:Quantity)	
	JOIN(Pro:ItemKey,Dtl:Item)	!Соединить с файлом товаров
	PROJECT(Pro:Description,Pro:Price)	
	END	
	END	
	END	
	END	
	CODE	
	OPEN(Customer,22h)	
	OPEN((Header,22h)	
	OPEN(Detail,22h)	
	OPEN(Product,22h)	
	SET(Cus:AcctKey)	
	OPEN(ViewOrder)	
	LOOP	!Цикл обработки записей
	LOOP	!Цикл для предотвращения взаимной блокировки
	HOLD(ViewOrder,1)	!Попытаться заблокировать запись в течение 1
секунды		
	NEXT(ViewOrder)	!Прочитать и заблокировать запись
	IF ERRORCODE() = 43	!Если это сделал кто-то другой
	CYCLE	! попытаться еще раз
	ELSE	
	BREAK	!Если запись не блокирована другим, выйти из
цикла		
	END	
	END	
	IF ERRORCODE() THEN BREAK.	!Проверять на конец файла

```

!Обработка записей
RELEASE(ViewOrder)      !Освободить заблокированные записи
END
CLOSE(ViewOrder)

```

Смотри также: RELEASE, NEXT, PREVIOUS

## NEXT (прочитать следующую запись виртуального файла)

### NEXT (*вирт. файл*)

**NEXT** Читает следующую запись виртуального файла в заданной последовательности.  
*вирт. файл* Метка виртуального файла.

Оператор **NEXT** считывает следующую в заданной ранее последовательности запись виртуального файла и помещает соответствующие поля в буферы данных файлов - компонент структуры VIEW. Если VIEW содержит структуры JOIN, то оператор NEXT считывает соответствующий набор следующих записей.

Оператор SET, выполняемый для первичного файла структуры VIEW перед открытием виртуального файла, определяет последовательность чтения записей. Первое выполнение оператора NEXT(вирт. файл) считывает запись в позиции, установленной оператором SET. Последующие выполнения оператора NEXT считывают последующие записи в этой последовательности. Выполнение операторов PUT и DELETE не влияет на установленную последовательность обработки записей.

Выполнение оператора NEXT без предварительной установки оператором SET положения в файле и последовательности обработки или попытка считывания записи после конца файла приводит к возникновению ошибочной ситуации "Record Not Available" (запись недоступна).

#### Выдаваемые сообщения об ошибках:

- 33 Record Not Available (запись недоступна)
- 37 File Not Open (файл не открыт)
- 43 Record Is Already Held (запись уже заблокирована)

#### Пример:

```

ViewOrder VIEW(Customer)      !Объявить структуру VIEW
      PROJECT(Cus:AcctNumber,Cus:Name)
      JOIN(Hea:AcctKey,Cus:AcctNumber)      !Соединить с файлом заголовков
      PROJECT(Hea:OrderNumber)
      JOIN(Dtl:OrderKey,Hea:OrderNumber)      !Соединить с файлом накладных

```

```

PROJECT(Det:Item,Det:Quantity)
JOIN(Pro:ItemKey,Dtl:Item)      !Соединить с файлом товаров
PROJECT(Pro:Description,Pro:Price)
END
END
END
END

```

## CODE

```

OPEN(Customer,22h)
OPEN((Header,22h)
OPEN(Detail,22h)
OPEN(Product,22h)
SET(Cus:AcctKey)
OPEN(ViewOrder)
LOOP                                !Прочитать все записи до конца файла
NEXT(ViewOrder)                    !      считать следующую запись
IF ERRORCODE() THEN BREAK. !      выйти по концу файла
DO PostTrans                       !      обратиться к процедуре
инициирования транзакции
END                                !Конец цикла

```

Смотри также: PREVIOUS, HOLD

## NOMEMO (читать записи, не считывая MEMO-поля)

**NOMEMO**(*вирт. файл*)

**NOMEMO**                      Включает режим выборки без MEMO-полей.  
*вирт. файл*                    Метка виртуального файла.

Оператор **NOMEMO** включает режим выборки без MEMO-полей последующими операторами NEXT и PREVIOUS. Следующие операторы NEXT и PREVIOUS считывают запись данных, но не считывают связанное с записью MEMO-поле. Обычно тем самым ускоряется выборка записей и этот режим следует использовать, когда содержимое MEMO-поля не нужно в данной процедуре.

### Пример:

```

Customer  FILE,DRIVER('Clarion'),PRE(Cus) !Объявить структуру файла покупателей
AcctKey   KEY(Cus:AcctNumber)
Notes     MEMO(1024)
Record    RECORD
AcctNumber                                LONG
Name      STRING(20)
Addr      STRING(20)

```

```

City      STRING(20)
State     STRING(20)
Zip       STRING(20)

CustView  ..
VIEW(Customer)      !Объявить структуру VIEW
END
CODE
OPEN(Customer)
Cus:AcctNumber = 12345      !Присвоить начальное значение полю ключа
SET(Cus:AcctKey,Cus:AcctKey)
OPEN(CustView)
LOOP
  NOMEMO(CustView)
  NEXT(CustView)      !Считать запись
  IF ERRORCODE() THEN BREAK.
  !Обработать запись
END
CLOSE(CustView)
    
```

**Смотри также:** GET, NEXT, PREVIOUS

## PREVIOUS (прочитать предыдущую запись виртуального файла)

**PREVIOUS**(*вирт. файл*)

**PREVIOUS** Читает предыдущую запись виртуального файла в заданной последовательности.

вирт. файл Метка виртуального файла.

Оператор **PREVIOUS** считывает предыдущую в заданной ранее последовательности запись виртуального файла и помещает соответствующие поля в буферы данных файлов - компонент структуры VIEW. Если VIEW содержит структуры JOIN, то оператор PREVIOUS считывает соответствующий набор предыдущих записей.

Оператор SET, выполняемый для первичного файла структуры VIEW перед открытием виртуального файла, определяет последовательность чтения записей. Первое выполнение оператора PREVIOUS(вирт. файл) считывает запись в позиции, установленной оператором SET. Последующие выполнения оператора PREVIOUS считывают предыдущие записи в этой последовательности. Выполнение операторов PUT и DELETE не влияет на установленную последовательность обработки записей.

Выполнение оператора PREVIOUS без предварительной установки оператором SET положения в файле и последовательности обработки или попытка считывания записи после прочтения первой записи файла приводит к возникновению ошибочной ситуации "Record Not Available" (запись недоступна).

Выдаваемые сообщения об ошибках:

- 33 Record Not Available (запись недоступна)
- 37 File Not Open (файл не открыт)
- 43 Record Is Already Held (запись уже заблокирована)

**Пример:**

```

ViewOrder VIEW(Customer)           !Объявить структуру VIEW
        PROJECT(Hea:OrderNumber)
        JOIN(Dtl:OrderKey,Hea:OrderNumber) !Соединить с файлом накладных
        PROJECT(Det:Item,Det:Quantity)
        JOIN(Pro:ItemKey,Dtl:Item) !Соединить с файлом товаров
        PROJECT(Pro:Description,Pro:Price)
        END
    END
END
CODE
OPEN((Header,22h)
OPEN(Detail,22h)
OPEN(Product,22h)
SET(Cus:AcctKey)
OPEN(ViewOrder)
LOOP                               !Прочитать все записи до начала файла
    NEXT(ViewOrder)                ! считать предыдущую запись
    IF ERRORCODE() THEN BREAK. ! выйти по началу файла
    DO PostTrans                    ! обратиться к процедуре инициирования
транзакции
END                                !Конец цикла

```

**Смотри также:** NEXT, HOLD

**PUT (занести запись обратно в первичный файл)**

**PUT** (*вирт. файл*)

**PUT**                    Заносит запись первичного файла структуры VIEW обратно на диск.  
*вирт. файл*            Метка виртуального файла.

Оператор **PUT** записывает текущие значения полей в буфере первичного файла структуры VIEW на место ранее прочитанной в файле записи. Если запись была заблокирована, то она автоматически освобождается. Оператор PUT заносит запись на место последней, полученной оператором NEXT, PREVIOUS. Если в буфере изменились значения полей, входящих в ключи, то обновляются и файлы ключей.

Оператор PUT обновляет запись только первичного файла потому, что структура VIEW выполняет одновременно и проекцию и соединение. И в этом случае возможно создать такую структуру VIEW, что обновление всех ее компонент нарушило бы правила внутренней непротиворечивости базы данных. Общее решение этой проблемы в SQL-ориентированных продуктах - это обновление только первичного файла. Поэтому в Clarion реализуется такое же стандартное решение.

Если предварительной выборки записи не было, или она удалена, то выдается сообщение “Record Not Available”, а запись не удаляется. Кроме того, для оператора PUT может выдаваться сообщение об ошибке “Create Duplicate Error”. Если выдано какое-либо сообщение об ошибке, то запись не заносится в файл.

Выдаваемые сообщения об ошибках:

- 05 Access Denied (доступ к файлу запрещен)
- 33 Record Not Available (запись недоступна)
- 40 Create Duplicate Error (создается повторяющееся значение ключа)

**Пример:**

```
ViewOrder VIEW(Header)
    JOIN(Dtl:OrderKey,Hea:OrderNumber) !Join Detail file
    PROJECT(Det:Item,Det:Quantity)
    END
    END
    CODE
    OPEN((Header,22h)
    OPEN(Detail,22h)
    SET(Cus:AcctKey)
    OPEN(ViewOrder)
    LOOP                                !Прочитать все записи в обратном порядке
        PREVIOUS(ViewOrder)           !считать запись
        IF ERRORCODE() THEN BREAK. !на начале файла выйти
        DO LastInFirstOut              !Вызвать процедуру обработки стека
        PUT(ViewOrder)                 !Записать обратно в файл запись транзакции
        IF ERRORCODE() THEN STOP(ERROR()).
    END                                !Конец цикла
```

**Смотри также:**    NEXT, PREVIOUS

**REGET (повторно прочитать запись)****REGET**(*вирт. файл, строка*)**REGET***вирт. файл**строка*

Повторно читает заданную запись виртуального файла.

Метка виртуального файла.

Строчковая константа или переменная, содержащая строку, полученную с помощью функции POSITION.

Оператор **REGET** читает запись виртуального файла, указанную строкой, возвращенной функцией POSITION(*вирт. файл*). Значение, содержащееся в строке, и его длина зависят от файлового драйвера. Если структура VIEW содержит структуры JOIN, то оператор REGET считывает соответствующий набор связанных записей.

Оператор REGET повторно заносит полные записи в буферы файлов - компонент структуры VIEW. Он не выполняет реляционной операции “проекция”. Оператор REGET(*вирт. файл*) специально предназначен для того, чтобы перед закрытием виртуального файла восстановить соответствующие записи в буферах. Однако последовательность обработки файлов должна быть повторно установлена операторами SET или RESET.

Выдаваемые сообщения об ошибках:

33 Record Not Available (запись недоступна)

**Пример:**

```
ViewOrder VIEW(Customer)           !Объявить структуры VIEW
        PROJECT(Cus:AcctNumber,Cus:Name)
        JOIN(Hea:AcctKey,Cus:AcctNumber) !Join Header file
        PROJECT(Hea:OrderNumber)
        JOIN(Dtl:OrderKey,Hea:OrderNumber) !Join Detail file
        PROJECT(Det:Item,Det:Quantity)
        JOIN(Pro:ItemKey,Dtl:Item) !Join Product file
        PROJECT(Pro:Description,Pro:Price)
        END
        END
        END
        END
RecordQue QUEUE,PREF(Que)
AcctNumber LIKE(Cus:AcctNumber)
Name LIKE(Cus:Name)
OrderNumber LIKE(Hea:OrderNumber)
Item LIKE(Det:Item)
Quantity LIKE(Det:Quantity)
Description LIKE(Pro:Description)
```



```

Price      LIKE(Pro:Price)
SavPosition STRING(260)
END
CODE
OPEN(Customer,22h)
OPEN((Header,22h)
OPEN(Detail,22h)
OPEN(Product,22h)
SET(Cus:AcctKey)
OPEN(ViewOrder)
LOOP
NEXT(ViewOrder)
IF ERRORCODE()
DO DisplayQue
BREAK
END
RecordQue := Cus:Record
RecordQue := Hea:Record
RecordQue := Dtl:Record
RecordQue := Pro:Record
SavPosition = POSITION(ViewOrder)
ADD(RecordQue)
IF ERRORCODE() THEN STOP(ERROR()).
END
ACCEPT
CASE ACCEPTED()
OF ?ListBox
GET(RecordQue,CHOICE())
REGET(ViewOrder,Que:SavPosition)
CLOSE(ViewOrder)
FREE(RecordQue)
UpdateProc
BREAK
END
END

```

!Прочитать все записи файла  
!читать последовательно записи

!Занести запись в очередь  
!Занести запись в очередь  
!Занести запись в очередь  
!Занести запись в очередь  
!Запомнить положение записей  
!и добавить элемент в очередь

!Восстановить буферы записей  
! и снова получить записи

!Вызвать процедуру обновления

**Смотри также:**    POSITION, RESET

**RELEASE (освободить запись виртуального файла)****RELEASE** (*вирт. файл*)**RELEASE***вирт. файл*

Освобождает заблокированную запись.

Метка виртуального файла.

Оператор **RELEASE** освобождает ранее заблокированную запись. Этот оператор не освобождает запись, заблокированную в многопользовательской среде другим пользователем. Если запись не заблокирована, или заблокирована другим пользователем, то оператор RELEASE игнорируется.

**Пример:**

```
ViewOrder VIEW(Customer)                !Объявить структуру VIEW
        PROJECT(Cus:AcctNumber,Cus:Name)
        JOIN(Hea:AcctKey,Cus:AcctNumber)    !Join Header file
        PROJECT(Hea:OrderNumber)
        JOIN(Dtl:OrderKey,Hea:OrderNumber)  !Join Detail file
        PROJECT(Det:Item,Det:Quantity)
        JOIN(Pro:ItemKey,Dtl:Item)          !Join Product file
        PROJECT(Pro:Description,Pro:Price)
        END
        END
        END
        END
        CODE
        OPEN(Customer,22h)
        OPEN((Header,22h)
        OPEN(Detail,22h)
        OPEN(Product,22h)
        SET(Cus:AcctKey)
        OPEN(ViewOrder)
        LOOP                                !Цикл обработки записей
        LOOP                                !Цикл во избежание взаимной блокировки
        HOLD(ViewOrder,1)                   !В течение 1 сек. пытаться заблокировать запись
        NEXT(ViewOrder)                     !Прочитать и заблокировать запись
        IF ERRORCODE() = 43                  !Если кто-то уже заблокировал
        CYCLE                                ! попытаться снова
        ELSE
        BREAK                                !Выйти если запись никем не заблокирована
        END
        END
        IF ERRORCODE() THEN BREAK.          !Проверить на конец файла
        !Обработка записей
        RELEASE(ViewOrder)                  !Освободить заблокированную запись
```

END

Смотри также: HOLD

**RESET (восстановить положение в последовательности записей)****RESET**(*вирт. файл, строка*)

**RESET** Восстановить указатель последовательной обработки на заданную запись виртуального файла

*вирт. файл* Метка виртуального файла.

*строка* Строковая константа или переменная, содержащая строку, полученную с помощью функции POSITION.

Оператор **RESET** восстанавливает указатель последовательной обработки на запись виртуального файла, которая задана строкой, возвращенной функцией POSITION. Как только указатель восстановлен оператором RESET, оператор NEXT или PREVIOUS прочтет эту запись.

Значение, содержащееся в строке (возвращаемой функцией POSITION), и его длина зависят от файлового драйвера. Оператор RESET используется совместно с функцией POSITION для временной приостановки и продолжения последовательной обработки виртуального файла.

**Пример:**

```
ViewOrder VIEW(Customer)           !Объявить структуру VIEW
      PROJECT(Cus:AcctNumber,Cus:Name)
      JOIN(Hea:AcctKey,Cus:AcctNumber) !Соединить с файлом заголовков
      PROJECT(Hea:OrderNumber)
      JOIN(Dtl:OrderKey,Hea:OrderNumber) !Соединить с файлом накладных
      PROJECT(Det:Item,Det:Quantity)
      JOIN(Pro:ItemKey,Dtl:Item) !Соединить с файлом товаров
      PROJECT(Pro:Description,Pro:Price)
      END
      END
      END
      END
RecordQue QUEUE,PRE(Que)
AcctNumber      LIKE(Cus:AcctNumber)
Name            LIKE(Cus:Name)
OrderNumber     LIKE(Hea:OrderNumber)
Item            LIKE(Det:Item)
Quantity        LIKE(Det:Quantity)
Description     LIKE(Pro:Description)
```

```

Price      LIKE(Pro:Price)
           END
SavPosition STRING(260)
           CODE
           OPEN(Customer,22h)
           OPEN((Header,22h)
           OPEN(Detail,22h)
           OPEN(Product,22h)
           SET(Cus:AcctKey)
           OPEN(ViewOrder)      !Начало файла в последовательности ключа
           LOOP                  !Прочитать все записи в файле
           NEXT(ViewOrder)      !   читать следующую запись
           IF ERRORCODE()
           DO DisplayQue
           BREAK
           END
           RecordQue := Cus:Record !Занести запись в очередь
           RecordQue := Hea:Record !Занести запись в очередь
           RecordQue := Dtl:Record !Занести запись в очередь
           RecordQue := Pro:Record !Занести запись в очередь
           ADD(RecordQue)        !   и добавить элемент в очередь
           IF ERRORCODE() THEN STOP(ERROR()).
           IF RECORDS(RecordQue) = 20 !В очереди 20 записей?
           DO DisplayQue        !Вывести очередь на экран
           ..                    !Конец цикла

```

```

DisplayQue ROUTINE
           SavPosition = POSITION(ViewOrder)      !Запомнить положение в
последовательности
           DO ProcessQue      !Вывести очередь на экран
           FREE(RecordQue)    !   и очистить ее
           RESET(ViewOrder,SavPosition) !Установить заново указатель
           NEXT(ViewOrder)    ! и прочесть заново запись

```

**Смотри также:**    POSITION, NEXT, PREVIOUS

## SET (последовательная обработка виртуального файл)

**SET**(*виртуальный\_файл* [, *число* ] )

**SET**                    Устанавливает последовательную обработку определенного фильтром набора записей виртуального файла, упорядоченного в соответствии с атрибутом ORDER.

*виртуальный\_файл*    Метка структуры VIEW.

*число*                    Целочисленная константа, переменная или выражение, задающее начальное положение, на основе числа первых компонент атрибута OR-

DER. Если этот параметр опущен, то используются все компоненты атрибута ORDER.

Оператор **SET** устанавливает последовательную обработку определенного фильтром набора записей виртуального файла, упорядоченного в соответствии с атрибутом ORDER. Параметр номер ограничивает оператор SET использованием только заданное число первых компонент атрибута ORDER. До выполнения оператора SET виртуальный файл должен быть открыт.

### Пример:

```
ViewOrder VIEW(Customer),FILTER('Hea:OrderTotal >= 500') |
,ORDER('-Hea:OrderDate,Cus:Name')
PROJECT(Cus:AcctNumber,Cus:Name)
JOIN(Hea:AcctKey,Cus:AcctNumber)      !Присоединить файл Header
PROJECT(Hea:OrderNumber,Hea:OrderTotal,Hea:OrderDate)
JOIN(Dtl:OrderKey,Hea:OrderNumber)  ! Присоединить файл Detail
PROJECT(Det:Item,Det:Quantity)
JOIN(Pro:ItemKey,Dtl:Item) ! Присоединить файл Product
PROJECT(Pro:Description,Pro:Price)
END
END
END
END

CODE
OPEN(Customer,22h)
OPEN((Header,22h)
OPEN(Detail,22h)
OPEN(Product,22h)
OPEN(ViewOrder)
SET(ViewOrder)                !Начало набора записей определенного атр.

ORDER
LOOP                          !Читать все записи файла
NEXT(ViewOrder)               !последовательно
IF ERRORCODE() THEN BREAK.
    !Обработка
..                             !Конец цикла
```

**Смотри также:** NEXT, PREVIOUS, FILTER, ORDER

## SKIP (пропустить записи в виртуальном файле)

**SKIP** (*вирт. файл, число*)

**SKIP** Пропускает записи при последовательной обработке виртуального файла

*вирт. файл* Метка виртуального файла.

*число* Числовая константа или переменная. Параметр число задает количество пропускаемых записей. Если это число положительное, то записи пропускаются в прямом направлении последовательности обработки (как NEXT); если параметр число отрицательное, то записи пропускаются в обратной последовательности (как PREVIOUS).

Оператор **SKIP** используется для того, чтобы пропускать записи при последовательной обработки виртуального файла. Он пропускает записи (в последовательности, заданной оператором SET) изменяя значение указателя на заданное число записей. Для пропуска записей оператор SKIP более эффективен, чем NEXT или PREVIOUS, потому что он не считывает записи в буфер(ы).

Если при выполнении оператора SKIP происходит выход за конец или начало файла, то функция EOF или BOF возвращает значение “истина” (если это поддерживается используемой файловой системой). Если предварительно не был выполнен оператор SET, то оператор SKIP игнорируется.

### Пример:

```
ViewOrder VIEW(Customer)           !Объявить структуру VIEW
      PROJECT(Cus:AcctNumber,Cus:Name)
      JOIN(Hea:AcctKey,Cus:AcctNumber)   !Соединить с файлом заголовков
      PROJECT(Hea:OrderNumber)
      JOIN(Dtl:OrderKey,Hea:OrderNumber) !Соединить с файлом накладных
      PROJECT(Det:Item,Det:Quantity)
      JOIN(Pro:ItemKey,Dtl:Item)   !Соединить с файлом товаров
      PROJECT(Pro:Description,Pro:Price)
      END
      END
      END
      END
SavOrderNo LONG
      CODE
      OPEN(Customer,22h)
      OPEN((Header,22h)
      OPEN(Detail,22h)
      OPEN(Product,22h)
      SET(Cus:AcctKey)
```

```

OPEN(ViewOrder)           !Начало файла в последовательности ключа
LOOP                       !Обработать все записи
NEXT(ViewOrder)           ! прочитать запись
IF ERRORCODE() THEN BREAK.
IF Hea:OrderNumber <> SavOrderNo
! проверить на первый элемент в последовательности
IF Hea:OrderStatus = 'Cancel' ! Это не отмененный заказ?
SKIP(Items,View:ItemCount-1) ! пропустить остаток элементов
CYCLE                       ! и обработать следующий заказ
..                           !Конец структур IF
DO ItemProcess             ! Обработать элемент
SavInvNo = Hea:OrderNUmber ! Запомнить номер заказа
END                         !Конец цикла

```

## WATCH (автоматическая проверка совместного использования)

**WATCH**(*виртуальный файл*)

**WATCH** Включает автоматическую проверку совместного использования базы данных по оптимистической стратегии.

*виртуальный файл* Метка объявления виртуального файла.

Оператор **WATCH** включает автоматическую проверку файловым драйвером совместного использования базы данных для последующих операторов NEXT и PREVIOUS в многопользовательской среде по оптимистической стратегии. Обычно файловый драйвер сохраняет копию значений полей, прочитанных из каждого файла при успешном выполнении операторов NEXT и PREVIOUS. При записи оператором PUT обратно в виртуальный файл поля на диске сравниваются с первоначально выбранными данными. В случае изменения их другим пользователем при выполнении оператора PUT выдается код ошибки. Конкретные действия, выполняемые оператором WATCH, зависят от файлового драйвера.

### Пример:

```

Customer  FILE,DRIVER('Clarion'),PRE(Cus) !Объявить структуру файла покупателей
AcctKey   KEY(Cus:AcctNumber)
Record    RECORD
AcctNumber                LONG
OrderNumber               LONG
Name                     STRING(20)
Addr                     STRING(20)
City                     STRING(20)
State                    STRING(20)
Zip                      STRING(20)
..

```

```
CustView  VIEW(Customer)          !Объявить структуру VIEW
          END
          CODE
          OPEN(Customer,22h)
          SET(Cus:AcctKey)
          OPEN(ViewOrder)          !Начало файла в последовательности ключа
          LOOP                     !Обработать все записи
            WATCH(ViewOrder)       !Включить проверку совместного использования
            NEXT(ViewOrder)        ! Прочитать запись
            IF ERRORCODE() THEN BREAK.
            DO ItemProcess          ! Обработать
            PUT(ViewOrder)         ! и записать обратно
            IF ERRORCODE() THEN STOP(ERROR()).
!Остановиться по любой ошибке, включая изменение записи другим пользователем
          END
```



## Функции для работы с виртуальными файлами

### POSITION (получить идентификатор положения в виртуальном файле)

**POSITION**(последовательность)

**POSITION** Однозначно идентифицировать положение в виртуальном файле.  
*последовательность* Метка объявления виртуального файла.

Функция **POSITION** возвращает строку, которая однозначно идентифицирует положение записи в заданном виртуальном файле. Эта функция возвращает положение записи виртуального файла, к которой было последнее обращение. Функция **POSITION** используется в сочетании с оператором **RESET** для того, чтобы временно приостанавливать и возобновлять последовательную обработку файла.

Возвращаемая функцией **POSITION** строка содержит идентификацию последовательности, установленной в основном (первичном) файле оператором **SET**, выполненном непосредственно перед открытием виртуального файла. Кроме того, эта строка содержит специфичное для файловой системы возвращаемое функцией **POSITION** значение ключа первичного файла и всех связывающих ключей вторичных файлов. Это позволяет функции **POSITION**(последовательность) однозначно определить всех связанных записей виртуального файла.

Как правило для файловых систем, в которых записи имеют номера, размер строки, возвращаемой функцией **POSITION**(файл) равна 4 байта. Размер строки, возвращаемой функцией **POSITION**(ключ) равна 4 плюс сумма длин полей, составляющих ключ. Для файловых систем, в которых записи не имеют номеров, размер строки, возвращаемой функцией **POSITION**(файл) обычно равна сумме длин полей первичного ключа (первый ключ файла, не имеющий атрибута **DUP** и **OPT**). Размер строки, возвращаемой в этом случае функцией **POSITION**(ключ) равна сумме длин полей первичного ключа плюс сумме длин полей заданного ключа.

Тип возвращаемого значения:      **STRING**

**Пример:**

```
ViewOrder VIEW(Customer)                !Объявить структуру VIEW
        PROJECT(Cus:AcctNumber,Cus:Name)
        JOIN(Hea:AcctKey,Cus:AcctNumber)  !Join Header file
        PROJECT(Hea:OrderNumber)
        JOIN(Dtl:OrderKey,Hea:OrderNumber) !Join Detail file
        PROJECT(Det:Item,Det:Quantity)
        JOIN(Pro:ItemKey,Dtl:Item)        !Join Product file
        PROJECT(Pro:Description,Pro:Price)
```

```

        END
    END
END
END
RecordQue QUEUE,PRE(Que)
AcctNumber      LIKE(Cus:AcctNumber)
Name           LIKE(Cus:Name)
OrderNumber    LIKE(Hea:OrderNumber)
Item           LIKE(Det:Item)
Quantity       LIKE(Det:Quantity)
Description    LIKE(Pro:Description)
Price          LIKE(Pro:Price)
END
SavPosition STRING(260)
CODE
OPEN(Customer,22h)
OPEN((Header,22h)
OPEN(Detail,22h)
OPEN(Product,22h)
SET(Cus:AcctKey)
OPEN(ViewOrder)      !Начало файла в последовательности ключа
LOOP                 !Прочитать все записи файла
    NEXT(ViewOrder)   ! читать записи последовательно
    IF ERRORCODE()
        DO DisplayQue !Вывести очередь
        BREAK
    END
    RecordQue := Cus:Record !Переслать данные из записи в очередь
    RecordQue := Hea:Record !Переслать данные из записи в очередь
    RecordQue := Dtl:Record !Переслать данные из записи в очередь
    RecordQue := Pro:Record !Переслать данные из записи в очередь
    ADD(RecordQue)         ! и добавить элемент в очередь
    IF ERRORCODE() THEN STOP(ERROR()).
    IF RECORDS(RecordQue) = 20 !20 записей в очереди?
        DO DisplayQue      !Вывести очередь
    ..
DisplayQue ROUTINE
    SavPosition = POSITION(ViewOrder)      !Запомнить положение записи
    DO ProcessQue      !Вывести очередь
    FREE(RecordQue)    ! и очистить ее
    RESET(ViewOrder,SavPosition) !Восстановить указательна запись
    NEXT(ViewOrder)    ! и получить запись снова

```

**Смотри также:**    RESET

## Глава 13 Очереди в памяти.

### Структура QUEUE

#### QUEUE (объявить структуру QUEUE)

<i>метка</i>	<b>QUEUE( [ группа ] ) [,PRE][,STATIC][,TREAD][,TYPE][,BINDABLE]</b>
	<b>[,EXTERNAL] [,DLL]</b>
<i>метка_поля</i>	<i>переменная</i> [,NAME())]
.	

#### QUEUE

Объявляет структуру очереди в памяти.

#### *метка*

Имя структуры QUEUE.

#### *группа*

Метка ранее объявленной структуры GROUP или QUEUE от которой наследуется структура данных. В этом качестве могут выступать структуры GROUP и QUEUE с атрибутом TYPE

#### PRE

Объявляет префикс для полей в данной структуре.

#### STATIC

Объявляет локальную по отношению к процедуре или функции очередь, буфер которой выделяется в статической памяти.

#### TREAD

Указывает, что память для очереди выделяется один раз для каждого исполняемого процесса. Должен использоваться с атрибутом STATIC для локальных данных процедуры.

#### TYPE

Указывает, что данное объявление является объявлением типа для очереди, передаваемой в качестве параметра.

#### BINDABLE

Задаёт, что переменные из этой структуры можно использовать в динамических выражениях.

#### EXTERNAL

Указывает, что структура QUEUE описывается во внешней библиотеке (там же выделяется память для нее).

#### DLL

Объявляет структуру QUEUE, определенную внешне, в библиотеке DLL. Дополнительно к этому атрибуту требуется атрибут EXTERNAL.

#### *переменная*

Объявление данных. Общая длина объявляемых полей может быть до 65000 байт в 16-ти разрядных приложениях и до 4 Мбт в 32-х разрядных.

Оператор **QUEUE** объявляет структуру записи очереди в памяти. Метка структуры QUEUE используется в операторах и функциях, манипулирующих с элементами очереди и с очередью в целом. При использовании в операторах присвоения, выражениях или списках параметров элемент очереди рассматривается как группа переменных.

Начало структуры QUEUE, объявленной с параметром группа совпадает со структурой, указываемой этим параметром, данная очередь наследует поля группы, указываемой параметром группа. В структуре QUEUE могут содержаться дополнительные, собственные поля, которые следуют за наследуемыми полями.

Очередь может рассматриваться как файл в памяти, реализованный как динамический массив из элементов очереди. При объявлении структуры `QUEUE` ей выделяется буфер (совсем как для файла). Каждый элемент очереди занимает точно такой объем памяти как буфер данных, без дополнительных издержек (а также без сжатия данных или отбрасывания пробелов).

Буфер данных для локальной по отношению к процедуре (объявленной в разделе данных процедуры или функции) очереди выделяется в стеке (если не указан атрибут `STATIC` и элемент очереди не слишком велик). Память, выделяемая для элементов локальной по отношению к процедуре очереди без атрибута `STATIC`, распределяется для нее только до тех пор, пока не будет выполнен оператор `FREE` или не завершится выполнение процедуры или функции - в этом случае память, занимаемая очередью освобождается автоматически.

Для очереди, объявленной в области глобальных данных, данных модуля или локальной очереди с атрибутом `STATIC` буфер выделяется в статической памяти и данные в нем сохраняются и при переходе от одной процедуры к другой. Память, выделенная для элементов очереди, принадлежит очереди до тех пор, пока очередь не будет очищена оператором `FREE`.

Переменной в буфере данных очереди не присваивается никакого начального значения автоматически, значения им нужно присваивать явно. До того, как в вашей программе им впервые будет присвоено какое-либо значение, нельзя предполагать, что они содержат пробельное или нулевое значение.

Как только элемент добавляется в очередь, для него динамически выделяется память и данные копируются из буфера в элемент очереди. При удалении элемента из очереди занимаемая им память освобождается. Максимальное количество элементов в очереди равно 1000000. Объем памяти, занимаемой каждым элементом, равен сумме длин составляющих его полей.

Структура `QUEUE` с атрибутом `BINDABLE` предполагает, что все переменные из этой структуры доступны для использования в динамических выражениях, без обязательного выполнения оператора `BIND` для каждого поля (позволяя выполнить один оператор `BIND(очередь)`, чтобы сделать все доступными все поля из нее). Содержимое параметра `NAME` для каждой переменной является логическим именем, используемым в динамическом выражении. Если атрибут `NAME` не указан, то используется имя переменной (включая префикс). Для имен всех переменных структуры в EXE-модуле резервируется память. Это увеличивает размер программы и затраты оперативной памяти. Поэтому атрибут `BINDABLE` следует применять только когда большая часть составляющих структуру полей планируется использовать в динамических выражениях.

Для структуры `QUEUE` с атрибутом `TYPE` память не выделяется. Это только

определение типа для очередей, передаваемых в качестве параметров в процедуры или функции. Определение типа позволяет процедуре или функции непосредственно адресоваться к отдельным полям переданной очереди. Объявление параметра в операторе PROCEDURE или FUNCTION устанавливает локальный префикс для передаваемой очереди. Например, PROCEDURE(LOC:PassedGroup) объявляет, что процедура использует для непосредственного обращения к полям-компонентам передаваемой в качестве параметра структуры QUEUE префикс LOC: (вместе с именами отдельных полей, использованными в объявлении типа).

### Пример:

```
NameQue  QUEUE,PRE(Nam)           !Объявить очередь
Name     STRING(20)
Zip      DECIMAL(5,0),NAME('SortField')
.        !Конец структуры очереди
NameQue2 QUEUE(NameQue),PRE(Nam2)  !Очередь, наследующая поля Name и Zip
Phone    STRING(10)                ! и имеющая дополнительно поле Phone
END                                         !Конец объявления очереди
```

**Смотри также:** PRE, STATIC, NAME, FREE, THREAD

## PRE (задать префикс для переменных структуры)

**PRE( [префикс ])**

**PRE**                      Обеспечивает префикс для переменных, объявленных в структуре QUEUE.

*префикс*                  Допустимы символы букв, цифр от 0 до 9 и символ подчеркивания. Префикс должен начинаться с буквы или символа подчеркивания.

Атрибут PRE обеспечивает префикс для составных структур данных. Он используется для того, чтобы различать одноименные переменные в различных структурах. При использовании в исполняемых операторах, операторах присваивания и в списках параметров префикс присоединяется к имени переменной с помощью двоеточия (префикс:имя).

Другой способ различать переменные с одинаковыми именами, которые находятся в разных структурах, состоит в том, чтобы вместо атрибута PRE использовать синтаксис уточнения имен. При упоминании в исполняемом операторе, присвоении или списке параметров имя структуры, содержащей переменную присоединяется спереди к имени переменной через двоеточие (QueueName:Label).

**Пример:**

```
SaveQueue QUEUE,PRE(Sav)
Field1      LONG           !Обращаться как к Sav:Field1 или SaveQueue:Field1
Field2      STRING        !Обращаться как к Sav:Field2 или SaveQueue:Field2
END
```

**Смотри также:** Зарезервированные слова, Синтаксис уточнения имен.

**STATIC (статическая локальная очередь)****STATIC**

Атрибут **STATIC** говорит о том, что буферу данных очереди, объявленной в процедуре или функции, должна распределяться статическая память, а не память в стеке. Этот атрибут позволяет любому значению, содержащемуся в такой переменной “возобновляться” при переходе от одной копии процедуры к другой.

**Пример:**

```
SomeProc PROCEDURE
SaveQueue QUEUE,STATIC      !Статический буфер данных очереди
Field1      LONG           !Значение, сохраняемое между
Field2      STRING        ! обращениями к процедурам
END
```

**Смотри также:** Объявление данных и распределение памяти

**THREAD (отдельный буфер очереди для каждого процесса)****THREAD**

Атрибут **THREAD** указывает, что память для буфера статической очереди распределяется отдельно для каждого исполняемого процесса в программе. Таким образом значения, содержащиеся в буфере зависят от того, какой процесс выполняется. Как только начат новый исполняемый процесс, для него создается новая, собственная копия очереди.

Атрибут **THREAD** подразумевает статическую очередь, так что атрибут **STATIC** не требуется для очереди локальной по отношению к процедуре. Этот атрибут влечет за собой массу накладных расходов во время выполнения программы, поэтому его следует использовать только когда это абсолютно необходимо.

**Пример:**

```
SomeProc PROCEDURE
SaveQueue QUEUE,THREAD      !Буфер данных статической очереди
Field1      LONG      !отдельная очередь для каждого исполняемого процесса
Field2      STRING
END
```

**Смотри также:** Объявление данных и распределение памяти

**NAME (задать внешнее имя для переменной в очереди)**

NAME([ *имя* ])

**NAME**            Задает “внешнее” имя.  
*имя*              Строковая константа, содержащая внешнее имя.

Атрибут **NAME** указывает в объявлении переменной в очереди “внешнее” имя для обработки очереди. Это имя представляет собой альтернативный способ обращения к переменным очереди, используемый операторами SORT, GET, PUT и ADD.

**Пример:**

```
SortQue     QUEUE,PRES(Que)
Field1      STRING(10),NAME('FirstField')  !Имя для сортировки
Field2      LONG,NAME('SecondField') !Имя для сортировки
END
```

**Смотри также:** QUEUE, SORT, GET, PUT, ADD

**TYPE (определение типа для очереди)**

TYPE

Атрибут **TYPE** создает очередь, которой не распределяется никакой памяти; она представляет собой только определение типа для очередей передаваемых в качестве параметра в процедуру или функцию. Определение типа позволяет процедуре или функции непосредственно адресоваться к отдельным полям переданной очереди. Объявление параметра в операторе PROCEDURE или FUNCTION устанавливает локальный префикс для передаваемой очереди. Например, PROCEDURE(LOC:PassedGroup) объявляет, что процедура использует для непосредственного обращения к полям-компонентам передаваемой в качестве параметра структуры QUEUE префикс LOC: (с именами отдельных полей, объявленными в определении типа).

Пример:

```
PassQue  QUEUE,TYPE      !Определение типа для передаваемых параметров типа QUEUE
F1       STRING(20)      ! первое поле
F2       STRING(1)       ! среднее поле
F3       STRING(20)      ! последнее
        END
        MAP
        MyProc1(PassQue)  !Передаваемая очередь определенная также как
PassQue
        END

NameQue  QUEUE,PRE(Nme)  !Очередь имен
First    STRING(20)      !Фамилия
Middle   STRING(1)       ! инициал
Last     STRING(20)
        END              !Конец объявления очереди

        CODE
        MyProc1(NameQue)  !Вызов процедуры с параметром NameQue
MyProc1  PROCEDURE(LOC:PassedGroup) !Процедура, получающая параметр QUEUE
LocalVar STRING(20)
        CODE
        LocalVar = LOC:F1  !Присвоение значения Nme:First переменной
LocalVar
                        !Из переданного параметра
```

**BINDABLE (использование в динамических выражениях)**

**BINDABLE**

Атрибут **BINDABLE** в операторе **QUEUE** объявляет, что составляющие структуру переменные можно использовать в динамических выражениях. Значение атрибута **NAME** каждой переменной, является ее логическим именем, используемым в динамическом выражении. Если атрибут **NAME** отсутствует, то используется имя переменной (включая префикс). Для имен всех переменных структуры в исполняемом файле резервируется пространство. Тем самым программа увеличивается, и требует больше памяти, чем было бы в обычном случае. Следовательно атрибут **BINDABLE** следует использовать, только когда собираетесь большую часть составляющих структуру полей использовать в динамических выражениях.

В исполняемой части программы перед тем как, отдельные поля структуры можно будет использовать в динамическом выражении, нужно выполнить оператор **BIND** в форме **BIND(группа)**.



**Пример:**

```

Names      QUEUE,BINDABLE
           !Структура, поля которой можно использовать в динамических выражениях
Name       STRING(20)
FileName   STRING(8),NAME('FName')  !Динамическое имя: FName
Dot        STRING(1)                !Динамическое имя: Dot
Extension  STRING(3),NAME('EXT')     ! Динамическое имя: EXT
           END
           CODE
           BIND(Names)

```

**Смотри также:**     BIND, UNBIND, EVALUATE

**EXTERNAL (очередь объявлена вне данной программы)****EXTERNAL**

Атрибут **EXTERNAL** указывает что очередь, к которой он относится, определена во внешней библиотеке. Таким образом структура **QUEUE** с атрибутом **EXTERNAL** объявляется и может использоваться в Clarion-программе, но память для буфера не выделяется. Память для буфера элемента такой очереди выделяется во внешней библиотеке. Этот атрибут позволяет Clarion-программе получить доступ к очередям, объявленным во внешней библиотеке как “public” - общие.

При использовании атрибута **EXTERNAL** для объявления очереди, совместно используемой несколькими библиотеками (.LIB , .DLL и .EXE) только в одной из них эта очередь должна объявляться без атрибута **EXTERNAL**. Во всех других библиотеках и программах следует объявлять эту очередь с атрибутом **EXTERNAL**. Это обеспечит уверенность в том, что для нее распределен только один буфера и во всех библиотеках и программах при обращении к нему будут ссылки на одну и ту же область памяти.

Объявления очередей во всех библиотеках (или .EXE - модулях), которые ссылаются на эти общие очереди, должны быть в точности одинаковыми (с соответствующим добавлением атрибута **EXTERNAL**). Если объявления отличаются, то может произойти разрушение данных. Ответственность за соблюдение идентичности объявлений лежит на программисте, поскольку ни компилятор не компоновщик не могут определить несоответствия объявлений в различных программах и библиотеках.

Можно посоветовать при разработке больших систем, использующих много библиотек DLL и/или EXE модулей, которые совместно используют одни и те же файлы, собирать реальные описания разделяемых глобальных переменных и очередей в одну библиотеку DLL. Таким образом создается одна “главная” библиотека DLL, в которой происходит отслеживание всех действительных объявлений очередей. Эта главная библиотека

связывается со всеми программами, которые используют общие очереди и переменные. Во всех других библиотеках и программах в этой системе общие файлы очереди и переменные должны объявляться с атрибутами EXTERNAL и DLL.

### Пример:

Names	QUEUE,EXTERNAL	!Очередь, объявленная во внешней библиотеке
Name	STRING(20)	
FileName	STRING(8),NAME('FName')	! Динамическое имя: FName
Dot	STRING(1)	! Динамическое имя: Dot
Extension	STRING(3),NAME('EXT')	! Динамическое имя: EXT
	END	

## DLL (очередь объявлена внешне, в библиотеке DLL)

**DLL( [ флаг ] )**

**DLL**                      Объявляет очередь, определенную внешне, в библиотеке DLL.  
*флаг*                      Числовая константа, метка соответствия, или определение системы поддержки проекта, которое задает активен ли данный атрибут. Если флаг установлен в 0, то этот атрибут неактивен, как если бы его вообще не было. Если флаг имеет отличное от нуля значение, то атрибут активен.

Атрибут **DLL** указывает, что структура QUEUE, в которой имеется этот атрибут, определена в библиотеке с динамическими связями (DLL). Структура QUEUE, имеющая атрибут DLL, должна иметь и атрибут EXTERNAL. В 32-х разрядных приложениях атрибут DLL обязателен, так как такие библиотеки являются настраиваемыми (перемещаемыми) в 32-битовом адресном пространстве, которое требует от компилятора еще одного дополнительного разыменовывания (преобразования адреса) при обращении к файлу.

Объявления очередей во всех библиотеках (или .EXE - модулях), которые ссылаются на эти общие очереди, должны быть в точности одинаковыми (с соответствующим добавлением атрибутов EXTERNAL и DLL). Если объявления отличаются, то может произойти разрушение данных. Ответственность за соблюдение идентичности объявлений лежит на программисте, поскольку ни компилятор не компоновщик не могут определить несоответствия объявлений в различных программах и библиотеках.

При использовании атрибутов EXTERNAL и DLL для объявления очереди, совместно используемой несколькими библиотеками (.LIB, .DLL и .EXE) только в одной из них эта очередь должна объявляться без атрибутов EXTERNAL и DLL. Во всех других библиотеках и программах следует объявлять очередь с этими атрибутами. Это обеспечит уверенность в том, что для нее распределен только один буфер и во всех библиотеках и программах при обращении к нему будут ссылки на одну и ту же область памяти.

Можно посоветовать при разработке больших систем, использующих много библиотек DLL и/или EXE модулей, которые совместно используют одни и те же очереди, собирать реальные описания разделяемых глобальных переменных, очередей и файлов в одну библиотеку DLL. Таким образом создается одна “главная” библиотека DLL, в которой происходит отслеживание всех действительных объявлений. Эта главная библиотека связывается со всеми программами, которые используют общие очереди, файлы и переменные. Во всех других библиотеках и программах в этой системе общие переменные должны объявляться с атрибутами EXTERNAL и DLL.

**Пример:**

```
DLLQueue  QUEUE,PRE(Que),EXTERNAL,DLL(1)
          ! Очередь, объявленная во внешней библиотеке .DLL
TotalCount  LONG
          END
```

**Смотри также:**     EXTERNAL

## Операторы работы с очередью

### ADD (добавить элемент в очередь)

<i>указатель</i>	
<b>ADD</b> (очередь[, [+ ]ключ,..., [- ]ключ]	
<i>имя</i>	
<b>ADD</b>	Добавляет новый элемент в очередь.
<i>очередь</i>	Метка структуры QUEUE или метка переданного параметра, представляющего собой структуру QUEUE.
<i>указатель</i>	Числовая константа, переменная или выражение. Значение указателя должно находиться в диапазоне от 1 до числа элементов в очереди.
<b>+-</b>	Стоящий спереди плюс или минус указывает, что данный ключ сортируется в возрастающей или убывающей последовательности.
<i>ключ</i>	Метка поля, объявленного внутри структуры QUEUE. Если у очереди есть атрибут PRE, то параметр ключ должен включать в себя и префикс.
<i>имя</i>	Строковая константа, переменная или выражение, содержащее разделенные запятыми, с необязательным знаком + или - спереди значения атрибутов NAME, относящихся к полям в структуре QUEUE. Регистр букв в значениях этого параметра является значимым.

Оператор **ADD** записывает новый элемент из буфера в очередь. Если для добавления не хватает памяти, то выдается сообщение “Insufficient Memory”.

- ADD(очередь )** Добавляет новый элемент в конец очереди.
- ADD(очередь, указатель )** Помещает новый элемент в позицию с относительным номером, заданным параметром указатель. Если уже существует элемент с таким номером, то он “проталкивается” вниз, для того, чтобы освободить номер для нового элемента. Указатели всех следующих элементов подстраиваются с учетом появления нового элемента. Например, элемент, добавленный в позицию с относительным номером 10 “проталкивает” 10-й элемент в 11-ю позицию, 11-й - в 12-ю и т.д. Если указатель равен 0 или больше, чем число элементов в очереди, то элемент добавляется в конец очереди.
- ADD(очередь, ключ)** Включает новый элемент в упорядоченную очередь. Можно использовать несколько параметров ключ (до 16), разделенных запятыми, с необязательным знаком плюс или минус для обозначения возрастающей или убывающей последовательности. Данный элемент включается сразу после всех элементов, имеющих совпадающие с ним значения ключей. Если очередь пустая, то оператор **ADD(очередь, ключ)** можно использовать для построения упорядоченной очереди.
- ADD(очередь, имя)** Включает новый элемент в упорядоченную очередь в памяти. Строка

имя должна содержать значения атрибутов NAME, относящихся к полям структуры QUEUE, разделенные запятыми, с необязательным знаком + или - спереди. Элемент очереди вставляется сразу после всех элементов, имеющих совпадающие с ним значения полей. Если очередь пуста, то эта форма оператора ADD может использоваться для построения упорядоченной очереди.

Выдаваемые сообщения об ошибках:

08 Insufficient Memory

75 Invalid Field Type Descriptor

**Пример:**

NameQue QUEUE,PRE(Que)

Name STRING(20),NAME('FirstField')

Zip DECIMAL(5,0),NAME('SecondField')

CODE

ADD(NameQue)

!Добавить элемент в конец очереди

ADD(NameQue,1)

!Добавить элемент в позицию 1

Que:Name = 'Jones'

!Присвоить значения полям

Que:Zip = 12345

ADD(NameQue,+Que:Name,-Que:Zip) !По возрастанию имен

! по убыванию ZIP кодов

Que:Hame = 'Smith'

!Присвоить значения полям

Que:Zip = 12345

ADD(NameQue.'+FirstField,-SecondFiel4') !По возрастанию имен

! по убыванию ZIP кодов

## DELETE (удалить элемент очереди)

**DELETE**(очередь)

### DELETE

Удаляет элемент из очереди

*очередь*

Метка оператора QUEUE или метка переданного параметра, представляющего собой структуру QUEUE.

Оператор **DELETE** удаляет элемент очереди, к которому было последнее успешное обращение оператором GET или ADD, и высвобождает занимаемую им память. Если предварительно не выполнялся оператор GET или ADD, то выдается сообщение об ошибке "Entry Not Found" (элемент не найден). Оператор DELETE не влияет на значение, возвращаемое в данный момент функцией POINTER.

Выдаваемые сообщения об ошибках:

08 Insufficient Memory (недостаточно памяти)

30    Entry Not Found (элемент очереди не найден)

**Пример:**

Que:Name = 'Jones'	!Присвоить значение ключевому полю
GET(Name4ue,4ue:Name)	!Получить соответствующий элемент
DELETE(Name4ue)	!Удалить запись

**FREE (удалить все элементы очереди)**

FREE(*очередь*)

<b>FREE</b>	Удаляет все элементы очереди.
<i>очередь</i>	Метка оператора <b>QUEUE</b> или метка переданного параметра, представляющего собой структуру <b>QUEUE</b> .

Оператор **FREE** удаляет все элементы из очереди и высвобождает занимаемую ими память. Кроме того, при этом освобождается память, использованная для внутренних нужд при организации очереди. Буфер данных очереди оператор **FREE** не очищает.

Выдаваемые сообщения об ошибках:  
08    Insufficient Memory (недостаточно памяти)

**Пример:**

FREE(Location)	!Освободить очередь Location
FREE(NameQue)	!Освободить очередь NameQue

**GET (прочитать элемент очереди)**

*указатель*  
GET(*очередь* [, [+ ]ключ, ..., [- ]ключ ])  
*имя*

<b>GET</b>	Читает элемент очереди.
<i>очередь</i>	Метка структуры <b>QUEUE</b> или метка переданного параметра, представляющего собой структуру <b>QUEUE</b> .
<i>указатель</i>	Числовая константа, переменная или выражение. Значение указателя должно находиться в диапазоне от 1 до числа элементов в очереди.
<i>+-</i>	Стоящий спереди плюс или минус указывает, что данный ключ сортируется в возрастающей или убывающей последовательности.
<i>ключ</i>	Метка поля, объявленного внутри структуры <b>QUEUE</b> . Если у очереди есть атрибут <b>PRE</b> , то параметр ключ должен включать в себя и префикс.
<i>имя</i>	Строковая константа, переменная или выражение, содержащее разделенные

запятыми, с необязательным знаком + или - спереди значения атрибутов NAME, относящихся к полям в структуре QUEUE. Регистр букв в значениях этого параметра является значимым.

Оператор **GET** считывает элемент в буфер структуры QUEUE для работы с ним. Если оператор GET не находит соответствующего запросу элемента, то выдается сообщение об ошибке “Entry Not Found” (элемент не найден).

GET(очередь, указатель ) Выбирает элемент с относительным номером, заданным параметром указатель. Если указатель равен нулю, то функция POINTER тоже возвращает значение 0.

GET(очередь, ключ) Ищет элемент очереди, который соответствует значениям ключевых полей в буфере. Может использоваться несколько (до 16-ти) параметров ключ. Очередь должна быть уже упорядочена по полям, использующимся в качестве ключевых параметров.

GET(очередь, имя) Ищет элемент очереди, который соответствует значениям полей, имеющих атрибуты NAME, указанные параметром имя. Строка имя должна содержать значения атрибутов NAME, относящихся к полям структуры QUEUE, разделенные запятыми, с необязательным знаком + или - спереди. Очередь предварительно должна быть упорядочена по полям, указанным в параметре имя.

#### Выдаваемые сообщения об ошибках:

- 08 Insufficient Memory (недостаточно памяти)
- 30 Entry Not Found (элемент не найден)
- 75 Invalid Field Type Descriptor (неправильный описатель типа поля)

#### **Пример:**

```
NameQue  QUEUE,PRE(Que)
Name     STRING(20),NAME('FirstField')
Zip      DECIMAL(5,0),NAME('SecondField')

CODE
DO BuildQue                !Вызов подпрограммы построения очереди
GET(NameQue,1)             !Получить первую запись
IF ERRORCODE() THEN STOP(ERROR()).
Que:Name = 'Jones'         !Присвоить значение полю
GET(NameQue,Que:Name)      !Получить соответствующую запись
IF ERRORCODE() THEN STOP(ERROR()).
Que:Name=Fil:Name          !Присвоить полю значение Fil:Name
GET(NameQue,Que:Name)      !Получить соответствующую запись
IF ERRORCODE() THEN STOP(ERROR()).
Que:Name = 'Smith'         !Присвоить значение ключевым полям
```

```
Que:Zip = 12345
GET(NameQue,'FirstField,SecondField')      !Получить соответствующую запись
IF ERRORCODE() THEN STOP(ERROR()).
```

Смотри также: SORT

**PUT (записать элемент в очередь)**

```
PUT(очередь[, [+ /ключ, ..., [- /ключ]
      имя
```

<b>PUT</b>	Записывает элемент обратно в очередь.
<i>очередь</i>	Метка структуры QUEUE или метка переданного параметра, представляющего собой структуру QUEUE.
<i>+ - ‘</i>	Стоящий спереди плюс или минус указывает, что данный ключ сортируется в возрастающей или убывающей последовательности.
<i>ключ</i>	Метка поля, объявленного внутри структуры QUEUE. Если у очереди есть атрибут PRE, то параметр ключ должен включать в себя и префикс.
<i>имя</i>	Строковая константа, переменная или выражение, содержащее разделенные запятыми, с необязательным знаком + или - спереди значения атрибутов NAME, относящихся к полям в структуре QUEUE. Регистр букв в значениях этого параметра является значимым.

Оператор **PUT** после успешного выполнения оператора GET или ADD записывает содержимое буфера данных обратно в очередь. Если перед оператором PUT не выполнялся оператор Get или ADD, то выдается сообщение об ошибке “Entry Not Found” (элемент не найден).

- PUT(очередь)

Записывает данные обратно в ту же самую относительную позицию внутри очереди, к которой обращался последний успешно выполненный оператор GET или ADD.
- PUT(очередь, ключ)

После успешно выполненного оператора GET или ADD возвращает элемент в упорядоченную очередь, сохраняя упорядоченность, если любые ключевые поля изменили свое значение. Может использоваться несколько (до 16-ти) параметров ключ, разделенных запятыми, с необязательным знаком “+” или “-” спереди, означающим возрастающую или убывающую последовательность сортировки. Если в очереди имеется несколько элементов с данным значением ключевого поля, то записываемый элемент становится среди них последним.
- PUT(очередь, имя)

После успешно выполненного оператора GET или ADD возвращает элемент в упорядоченную очередь, сохраняя упорядоченность, если любые ключевые поля изменили свое значение. Строка имя должна содержать значения атрибутов NAME полей из структуры QUEUE, разделенных



запятыми с необязательным знаком “+” или “-” спереди, означающим возрастающую или убывающую последовательность сортировки. Если в очереди имеется несколько элементов с данным значением ключевого поля, то записываемый элемент становится среди них последним.

Выдаваемые сообщения об ошибках:

- 08 Insufficient Memory (недостаточно памяти)
- 30 Entry Not Found (элемент не найден)
- 75 Invalid Field Type Descriptor (неправильный типа поля)

**Пример:**

```
NameQue  QUEUE,PRE(Que)
Name     STRING(20),NAME('FirstField')
Zip      DECIMAL(5,0),NAME('SecondField')

CODE
DO BuildQue                                !Вызов подпрограммы построения очереди
Que:Name = 'Jones'                        !Присвоить значение ключевому полю
GET(NameQue,Que:Name)                    !Получить соответствующую запись
IF ERRORCODE() THEN STOP(ERROR()).
Que:Zip = 12345                            !Изменить Zip код
PUT(NameQue)                              !Записать изменения в очередь
IF ERRORCODE() THEN STOP(ERROR()).
Que:Name = 'Jones'                        !Присвоить значение ключевому полю
GET(NameQue,Que:Name)                    !Получить соответствующую запись
IF ERRORCODE() THEN STOP(ERROR()).
Que:Name = 'Smith'                        !Изменить значение ключевого поля
PUT(NameQue,Que:Name)                    !Записать изменения в очередь
IF ERRORCODE() THEN STOP(ERROR()).
Que:Name = 'Smith'                        !Присвоить значение ключевому полю
GET(NameQue,'FirstField')                !Получить соответствующую запись
IF ERRORCODE() THEN STOP(ERROR()).
Que:Name = 'Jones'                        !Изменить значение ключевого поля
PUT(NameQue,'FirstField')                !Записать изменения в очередь
IF ERRORCODE() THEN STOP(ERROR()).
```

## **SORT (упорядочить очередь)**

**SORT**(*очередь*[/+ /ключ,...,/- /ключ]  
*имя*)

**SORT** Упорядочивает очередь  
*очередь* Метка структуры QUEUE или метка переданного параметра, представляющего собой структуру QUEUE.  
 +- Стоящий спереди плюс или минус указывает, что данный ключ сортируется

	в возрастающей или убывающей последовательности.
<i>ключ</i>	Метка поля, объявленного внутри структуры QUEUE. Если у очереди есть атрибут PRE, то параметр ключ должен включать в себя и префикс.
<i>имя</i>	Строковая константа, переменная или выражение, содержащее разделенные запятыми, с необязательным знаком + или - спереди значения атрибутов NAME, относящихся к полям в структуре QUEUE. Регистр букв в значениях этого параметра является значимым.

Оператор **SORT** переупорядочивает элементы в очереди. Элементы с одинаковыми значениями ключевых полей сохраняют свое положение относительно друг друга.

**SORT(очередь, ключ)** Переупорядочивает очередь в последовательности, заданной параметром ключ. Может использоваться несколько параметров ключ (до 16-ти), разделенных запятыми, с необязательным знаком “+” или “-” спереди, означающим возрастающую или убывающую последовательность сортировки.

**SORT(очередь, имя)** Переупорядочивает очередь в последовательности, заданной строкой имя. Строка имя должна содержать значения атрибутов NAME полей из структуры QUEUE, разделенных запятыми с необязательным знаком “+” или “-” спереди, означающим возрастающую или убывающую последовательность сортировки.

#### Выдаваемые сообщения об ошибках:

08 Insufficient Memory (недостаточно памяти)

75 Invalid Field Type Descriptor (неправильный описатель типа поля)

#### **Пример:**

```
Location  QUEUE,PRE(Loc)
Name      STRING(20),NAME('FirstField')
City      STRING(10),NAME('SecondField')
State     STRING(2)
Zip       DECIMAL(5,0)
```

#### **CODE**

```
SORT(Location,Loc:State,Loc:City,Loc:Zip)
```

!Сортировать по Zip коду внутри города и внутри штата

```
SORT(Location,+Loc:State,-Loc:Zip)
```

!Сортировать по убыванию Zip кода внутри города

```
SORT(Location,'FirstField','-SecondField')
```

!Сортировать по убыванию города внутри имени

## Функции для работы с очередью

### POINTER (получить относительный номер)

#### POINTER(*очередь*)

**POINTER** Получает относительный номер записи, к которой было последнее обращение.

*очередь* Метка структуры QUEUE или метка переданного параметра, представляющего собой структуру QUEUE.

Функция **POINTER** возвращает целочисленное значение типа LONG, указывающее номер элемента очереди, к которому было обращение последним оператором ADD, GET, или PUT.

Тип возвращаемого значения: LONG

#### Пример:

```
Que:Name = 'Jones'           !Присвоить значение ключевому полю
GET(NameQue,Que:Name)       !Получить элемент
IF ERRORCODE() THEN STOP(ERROR()). !Проверить ошибку
SavPoint = POINTER(NameQue)   !Запомнить номер элемента
```

### RECORDS (получить число элементов в очереди)

#### RECORDS(*очередь*)

**RECORDS** Возвращает число элементов в очереди.

*очередь* Метка структуры QUEUE или метка переданного параметра, представляющего собой структуру QUEUE.

Функция **RECORDS** возвращает целочисленное значение типа LONG, содержащее число элементов в очереди.

Тип возвращаемого значения: LONG

#### Пример:

```
Entries# = RECORDS(Location) !Определить количество элементов
LOOP I# = 1 TO Entries#      !Цикл по всем элементам очереди
  GET(Location,I#)           ! получить запись
  IF ERRORCODE() THEN STOP(ERROR()).
  DO SomeProcess              ! и обработать ее
```



## Глава 14 Прочие операторы и функции

# Математические функции

### ABS (получить абсолютную величину)

**ABS**(*выражение*)

<b>ABS</b>	Возвращает абсолютную величину.
<i>выражение</i>	Константа, переменная или выражение.

Функция **ABS** возвращает абсолютную величину выражения. Абсолютная величина числа всегда положительная величина или ноль.

Тип возвращаемого значения: REAL

### Пример:

C = ABS(A - B) !C представляет абсолютное значение разности  
IF B < 0 THEN B = ABS(B). !Если B отрицательное число, сделать его положительным

## INRANGE (проверить попадание в диапазон)

**INRANGE**(*выражение, нижняя граница, верхняя граница*)

<b>INRANGE</b>	Проверяет попадание в диапазон.
<i>выражение</i>	Числовая константа, переменная или выражение
<i>нижняя граница</i>	Числовая константа, переменная или выражение, представляющая нижнюю границу диапазона
<i>верхняя граница</i>	Числовая константа, переменная или выражение, представляющая верхнюю границу диапазона

Функция **INRANGE** сравнивает числовое выражения с включающим числовым диапазоном. Функция возвращает единицу (“истина”), если значение выражения попадает в указанный диапазон. Если значение выражения больше параметра верхняя граница или меньше параметра нижняя граница, то функция возвращает ноль (“ложь”).

Тип возвращаемого значения: LONG

### Пример:

IF INRANGE(Date % 7,1,5)                      !Если это рабочий день

DO WeekdayRate	! использовать тариф рабочего дня
ELSE	! в противном случае
DO WeekendRate	! использовать тариф выходного дня

## INT (взять целую часть)

**INT**(*выражение*)

**INT** Возвращает целую часть числа.  
*выражение* Числовая константа, переменная или выражение

Функция **INT** возвращает целую часть значения числового выражения. Округления значения при этом не производится, а знак не изменяется.

Тип возвращаемого значения: REAL

### Пример:

INT(8.5) !возвращает 8  
 INT(-5.9) !возвращает -5  
 x=int(y) !возвращает целую часть от Y

## LOGE (вычислить натуральный логарифм)

**LOGE**(*выражение*)

**LOGE** Возвращает натуральный логарифм.  
*выражение* Числовая константа, переменная или выражение. Если значение выражения меньше нуля, то возвращаемое функцией значение равно нулю. Натуральный логарифм неопределен для значений меньших или равных нулю.

Функция **LOGE** возвращает натуральный логарифм значения числового выражения. Натуральный логарифм числа - это степень, в которую нужно возвести число e, чтобы получить данное число. Значение e равно 2.71828182846.

Тип возвращаемого значения: REAL

### Пример:

LOGE(2.71828182846) !возвращает 1  
 LOGE(1) !возвращает 0  
 LogVal = LOGE(Val) !Получить натуральный логарифм Val

**LOG10 (вычислить десятичный логарифм)****LOG10**(*выражение*)**LOG10***выражение*

Возвращает десятичный логарифм.

Числовая константа, переменная или выражение. Если значение выражения меньше нуля, то возвращаемое функцией значение равно нулю. Десятичный логарифм неопределен для значений меньших или равных нулю.

Функция **LOG10** возвращает десятичный логарифм значения числового выражения. Десятичный логарифм числа - это степень, в которую нужно возвести число 10, чтобы получить данное число.

Тип возвращаемого значения: REAL

**Пример:**

LOG10(10)	!возвращает 1
LOG10(1)	!возвращает 0
LogStore = LOG10(Var)	!Запомнить десятичный логарифм Val

**RANDOM (получить случайное число)****RANDOM**(*нижний предел, верхний предел*)**RANDOM***нижний предел*

Возвращает случайное число.

Числовая константа, переменная или выражение, задающее нижнюю границу диапазона.

*верхний предел*

Числовая константа, переменная или выражение, задающее верхнюю границу диапазона.

Функция **RANDOM** генерирует случайное целое в диапазоне между нижним и верхним пределами включительно. Параметры нижний предел и верхний предел могут быть любыми числовыми выражениями, но используется только их целая часть для формирования включающего диапазона.

Тип возвращаемого значения: LONG

**Пример:**

Num	BYTE,DIM(49)
LottoNbr	BYTE,DIM(6)
	CODE

```

CLEAR(Num)
CLEAR(LottoNbr)
LOOP X# = 1 TO 6
  LottoNbr[X#] = RANDOM(1,49)  !Взять значение для лотереи
  IF NOT Num[LottoNbr[X#]]
    Num[LottoNbr[X#]] = 1
  ELSE
    X# -= 1
  ..

```

## ROUND (округлить число)

**ROUND**(выражение, порядок)

### ROUND

*выражение*  
*порядок*

Округляет число.

Числовая константа, переменная или выражение

Числовое выражение, значение которого равно степени десяти, например: 1, 10, 100 или .1, .01, .001. Если значение параметра порядок не является кратным степени 10, то используется ближайшая более низкая кратная степень (например, вместо .55 используется 0.1, а вместо 155 используется 100).

Функция ROUND выдает значение выражения, округленного до степени десяти.

Тип возвращаемого значения: REAL

### Пример:

ROUND(5163,100)	возвращает 5200
ROUND(657.50,1)	возвращает 658
ROUND(51.63594,.01)	возвращает 51.64
Commission = ROUND(Price / Rate, .01) !Округлить комиссионные до цента	

## SQRT (квадратный корень)

**SQRT**(выражение)

### SQRT

*выражение*

Вычисляет квадратный корень.

Числовая константа, переменная или выражение. Если значение выражения меньше нуля, то функция возвращает 0.

Функция **SQRT** выдает значение квадратного корня выражения. Если X представляет любое положительное действительное число, то квадратный корень от X - это число, которое при умножении само на себя дает в результате X.



Тип возвращаемого значения: REAL

### Пример:

Length = SQRT(X^2 + Y^2)  
Пифагора)

!Найти расстояние от точки 0,0 до X,Y (теорема

## Тригонометрические функции

Тригонометрические функции возвращают значения, представляющие углы или отношения сторон прямоугольного треугольника - треугольника, у которого один угол прямой (равный 90 градусов). Сторона прямоугольного треугольника, противоположная прямому углу, называется гипотинузой. Для каждого из двух других углов прилегающая сторона образует угол с гипотинузой, а противоположная сторона находится против этого угла. Чтобы получить более подробное объяснение этих терминов используйте любой хороший учебник тригонометрии.

Углы выражаются в радианах. Число Пи является константой, которая представляет соотношение длины и радиуса окружности. Полный угол содержит 2\*Пи радиан (360 градусов).

Ниже приведены равенства, обеспечивающие высокоточное выражение констант для числа Пи и для преобразований между градусами и радианами.

PI	EQUATE(3.1415926535898)	! Число Пи
Rad2Deg	EQUATE(57.295779513082)	! Число градусов в радиане
Deg2Rad	EQUATE(.0174532925199)	! Число радиан в градусе

## SIN (синус)

**SIN(радианы)**

<b>SIN</b>	Возвращает синус.
<i>радианы</i>	Числовая константа, переменная или выражение представляющая угол в радианах.

Функция **SIN** возвращает значение тригонометрического синуса угла, измеренного в радианах. Синус угла представляет собой отношение длины противолежащего катета к длине гипотинузы.

Тип возвращаемого значения: REAL

**Пример:**

Angle = 45 \* Deg2Rad  
SineAngle = SIN(Angle)

!Преобразовать 45 градусов в радианы  
!Получить синус угла в 45 градусов

**COS (косинус)****COS**(*радианы*)**COS***радианы*

Возвращает косинус.

Числовая константа, переменная или выражение представляющая  
угол в радианах.

Функция **COS** возвращает значение тригонометрического косинуса угла, измеренного в радианах. Косинус угла представляет собой отношение длины прилежащего катета к длине гипотенузы.

Тип возвращаемого значения: REAL

**Пример:**

Angle = 45 \* Deg2Rad  
CosineAngle = COS(Angle)

!Преобразовать 45 градусов в радианы  
!Получить косинус угла в 45 градусов

**TAN (тангенс)****TAN**(*радианы*)**TAN***радианы*

Возвращает тангенс.

Числовая константа, переменная или выражение представляющая угол в  
радианах.

Функция **TAN** возвращает значение тригонометрического тангенса угла, измеренного в радианах. Тангенс угла представляет собой отношение длины противолежащего катета к длине прилежащего катета.

Тип возвращаемого значения: REAL

**Пример:**

Angle = 45 \* Deg2Rad  
TangentAngle = TAN(Angle)

!Преобразовать 45 градусов в радианы  
!Получить тангенс угла в 45 градусов

**ASIN (арксинус)****ASIN**(*выражение*)**ASIN** Возвращает арксинус.*выражение* Числовая константа, переменная или выражение, представляющая собой синус угла.

Функция **ASIN** возвращает арксинус. Арксинус значения синуса угла - это сам угол, который дает это значение. Возвращаемое значение представляет собой угол, выраженный в радианах.

Тип возвращаемого значения: REAL

**Пример:**

InvSine = ASIN(SineAngle) !Получить арксинус

**Смотри также:** SIN**ACOS (арккосинус)****ACOS**(*выражение*)**ACOS** Возвращает арккосинус.*выражение* Числовая константа, переменная или выражение, представляющая собой косинус угла.

Функция **ACOS** возвращает арккосинус. Арккосинус значения косинуса угла - это сам угол, который дает это значение. Возвращаемое значение представляет собой угол, выраженный в радианах.

Тип возвращаемого значения: REAL

**Пример:**

InvCosine = ACOS(CosineAngle) !Получить арккосинус

**Смотри также:** COS

## ATAN (арктангенс)

*ATAN(выражение)*

**ATAN** Возвращает арктангенс.

*выражение* Числовая константа, переменная или выражение, представляющая собой тангенс угла.

Функция **ATAN** возвращает арктангенс. Арктангенс значения тангенса угла - это сам угол, который дает это значение. Возвращаемое значение представляет собой угол, выраженный в радианах.

Тип возвращаемого значения: REAL

**Пример:**

InvTangent = ATAN(TangentAngle)      !Получить арктангенс

**Смотри также:** TAN

## Строковые функции

### ALL (повторение символов)

*ALL(строка[, длина])*

**ALL** Возвращает строку, повторенную несколько раз.

*строка* Строковое выражение, содержащее последовательность символов, которая должна быть повторена.

*длина* Длина возвращаемой строки. Если этот параметр опущен, то возвращается строка в 255 символов.

Функция **ALL** возвращает строку, содержащую повторения некоторой последовательности символов.

Тип возвращаемого значения: STRING

**Пример:**

Starline = ALL('\*',25)      !Взять 25 звездочек

Dotline = ALL('.',255)      !Взять 255 точек

**CENTER (центрировать строку)****CENTER**(*строка*[, *длина*])**CENTER**

Возвращает центрированную строку.

*строка*

Строковая константа, переменная или выражение.

*длина*

Длина возвращаемой строки. Если этот параметр опущен, то в этом качестве используется длина первого параметра.

Функция **CENTER** сначала удаляет пробелы в начале и в конце строки, затем, чтобы центрировать оставшуюся часть в рамках заданной длины дополняет слева и справа пробелами и возвращает отцентрированную строку.

**Пример:**

CENTER('ABC', 5)

возвращает ' ABC '

CENTER('ABC ')

возвращает ' ABC '

CENTER(' ABC')

возвращает ' ABC '

Message = CENTER(Message)

!Центрировать сообщение

Rpt:Title = CENTER(Name,60)

!Центрировать имя

**CHR (получить символ ASCII)****CHR**(*код*)**CHR**

Возвращает отображаемый символ.

*код*

Числовое выражение, содержащее код символа ASCII.

Функция **CHR** возвращает символ, соответствующий заданному параметром коду.

Тип возвращаемого значения: STRING

**Пример:**

Stringvar = CHR(122)

!Получить символ 'z'

Stringvar = CHR(65)

!Получить символ 'A'

## CLIP (отсечь пробелы в конце строки)

### CLIP(*строка*)

**CLIP** Удаляет пробелы в конце строки.  
*строка* Строковое выражение

Функция **CLIP** удаляет из строки оконечные пробелы. Возвращаемая строка представляет собой подстроку без пробелов в конце. Функция CLIP часто используется в строковых выражениях с операцией конкатенации.

Функция CLIP не нужна для строки типа CSTRING, поскольку она заканчивается нулем. Также не нужна функция CLIP для строки PSTRING, поскольку она имеет байт длины.

Тип возвращаемого значения: STRING

#### Пример:

Name = CLIP>Last) & ' ' & CLIP(First) & INIT & ' ' !Полное имя

## DEFORMAT (исключить форматирование из числовой строки)

### DEFORMAT(*строка*,*шаблон*)

**DEFORMAT** Исключает символы форматирования из числовой строки.  
*строка* Строковое выражение, содержащее числовую строку.  
*шаблон* Шаблон форматирования или метка переменной типа STRING, CSTRING или PSTRING, содержащей шаблон (CSTRING работает более эффективно чем STRING или PSTRING ). Если этот параметр опущен, то будет использоваться шаблон параметра строка. Если же эта строка объявлена без шаблона, то возвращаемое значение будет содержать только допустимые для числовой константы символы.

Функция **DEFORMAT** удаляет из числовой строки символы форматирования, возвращая только цифры, содержащиеся в строке. При использовании с шаблоном даты или времени (за исключением тех, которые подразумевают в себе буквы) эта функция возвращает строку, содержащую стандартную для Clarion дату или время.

Тип возвращаемого значения: STRING

#### Пример:

DialString = 'ATDT1' & DEFORMAT(Phone,@P(###)###-####P) & '<13,10>

!Получить строку для набора номера модемом

ClarionDate = DEFORMAT(dBaseDate,@D1)

!Получить стандартную в Clarion дату из мм/дд/гг

**FORMAT (форматировать число по шаблону)****FORMAT**(*значение, шаблон*)**FORMAT**

Возвращает отформатированную числовую строку.

*значение*

Числовое выражение, представляющее значение, которое должно быть отформатировано..

*шаблон*

Шаблон форматирования или метка переменной тип STRING, CSTRING или PSTRING, содержащая шаблон (CSTRING работает более эффективно чем STRING или PSTRING ).

Функция **FORMAT** возвращает строку цифр, отформатированную в соответствии с параметром шаблон.

Тип возвращаемого значения: STRING

**Пример:**

```
Rpt:SocSecNbr = FORMAT(Emp:SSN,@P###-##-####P)
Phone = FORMAT(DEFORMAT(Phone,@P###-###-####P),@P(###)###-####P)
!Изменить формат записи телефонного номера
DateString = FORMAT(DateLong,@D1) !Форматировать дату в строку
```

**INLIST (искать элемент в списке)****INLIST**(*строка, элемент списка[элемент списка]*)**INLIST**

Возвращает признак наличия или отсутствия элемента в списке.

*строка*

Константа, переменная или выражение, содержащее значение, которое следует искать в списке. Если это числовая величина, то перед сравнением она преобразуется в строку.

*элемент списка*

Метка переменной или константы, значение которой сравнивается с первым параметром функции. Если это числовая величина, то перед сравнением она преобразуется в строку. Может использоваться до 16-ти элементов списка, но должно быть по крайней мере два.

Функция **INLIST** сравнивает содержимое параметра строка со значением, содержащимся в каждом параметре элемента списка. Если совпадающее значение найдено, то функция возвращает номер параметра относительно первого элемента списка, содержащий этот совпадающий элемент. Если значение строки не найдено ни в одном элементе списка, то в этот случае функция возвращает 0.

Тип возвращаемого значения: LONG

**Пример:**

```
INLIST('D','A','B','C','D','E')  возвращает 4
INLIST('B','A','B','C','D','E')  возвращает 2
EXECUTE INLIST(Emp:Status,'Full','Part','Retired','Consult')
Scr:Message = 'All Benefits'
Scr:Message = 'Holidays Only'
Scr:Message = 'Medical/Dental Only'
Scr:Message = 'No Benefits'
END
```

**INSTRING (искать вхождение строки)**

**INSTRING**(*подстрока*, *строка*[,*шаг*][,*начало*])

**INSTRING**

Ищет подстроку в строке.

*подстрока*

Строковая константа, переменная или выражение, содержащая строку, которую следует искать. Следует использовать с переменной, содержащей подстроку, функцию CLIP, чтобы в функции INSTRING не производился поиск с учетом пробелов в конце переменной.

*строка*

Строковая константа или метка переменной типа STRING, PSTRING или CSTRING в которой должен осуществляться поиск.

*шаг*

Числовая константа, переменная или выражение, которая задает размер шага при поиске. Шаг равный 1 означает, что сравнение производится с подстрокой начало, которой каждый раз смещается на 1 символ, шаг равный 2 - что смещение происходит на 2 символа и т. д. Если параметр шаг опущен, то длина шага по умолчанию равна длине искомой подстроки.

*начало*

Числовая константа, переменная или выражение, которая задает с какого символа строки начинается поиск. Если этот параметр опущен, то поиск начинается с первого символа.

Функция **INSTRING** шаг за шагом продвигается по строке в поиске вхождения подстроки. Если подстрока найдена, то функция возвращает номер успешного шага (первый шаг имеет номер 1). Если подстрока не была найдена, то функция INSTRING возвращает ноль.

Тип возвращаемого значения: LONG

**Пример:**

```
INSTRING('DEF','ABCDEFGHIJ',1,1)  возвращает 4
INSTRING('DEF','ABCDEFGHIJ',2,1)  возвращает 0
INSTRING('DEF','ABCDEFGHIJ',2,2)  возвращает 2
INSTRING('DEF','ABCDEFGHIJ',3,1)  возвращает 2
```



```
Extension = SUB(FileSpec, INSTRING('.', FileSpec) + 1, 3)
!Выделить расширение из спецификации файла
IF INSTRING(CLIP(Search), Cus:Notes, 1, 1)
!Если значение переменной Search найдено
Scr:Message = 'FOUND' ! вывести сообщение
```

## LEFT (выровнять строку влево)

**LEFT**(*строка* [, *длина*])

<b>LEFT</b>	Выравнивает строку на налево.
<i>строка</i>	Строковая константа, переменная или выражение.
<i>длина</i>	Числовая константа, переменная или выражение, указывающая длину возвращаемой строки. Если этот параметр опущен, то берется длина параметра строка.

Функция **LEFT** возвращает выровненную влево строку. Пробелы в начале значения строки удаляются.

Тип возвращаемого значения: STRING

**Пример:**

```
!LEFT(' ABC') возвращает 'ABC '
CompanyName = LEFT(CompanyName) !Выровнять влево название компании
```

## LEN (получить длину строки)

**LEN**(*строка*)

<b>LEN</b>	Возвращает длину строки.
<i>строка</i>	Строковая константа, переменная или выражение

Функция **LEN** возвращает значение длины строки. Если параметр строка является меткой строковой переменной типа STRING, то функция возвратит ее объявленную длину. Если параметр строка является меткой строковой переменной типа CSTRING или PSTRING, то функции возвратит длину содержимого строки. Числовые переменные автоматически преобразуются в промежуточное, строковое значение.

Тип возвращаемого значения: LONG

**Пример:**

```
IF LEN(CLIP(Title) & ' ' & CLIP(First) & ' ' & CLIP(Last)) > 30 !Если полное имя не поместится
  Rpt:Name = CLIP(Title) & ' ' & SUB(First,1,1) & ' ' & CLIP(Last)
  ! использовать первую букву имени
ELSE
  Rpt:Name = CLIP(Title) & ' ' & CLIP(First) & ' ' & CLIP(Last)
  ! в противном случае использовать полное имя
.
Rpt:Name = CENTER(Cus:Name,LEN(Rpt:Title)) !Центрировать имя в заголовке
```

## LOWER (преобразовать в строчные буквы)

### LOWER(*строка*)

#### LOWER

Преобразует символы в строке в строчные.

*строка*

Строковая константа, переменная или выражение, представляющее строку, которая должна быть преобразована.

Функция LOWER возвращает строку, в которой все буквы строчные.

Тип возвращаемого значения: STRING

#### Пример:

```
!LOWER('ABC')           возвращает 'abc'
Name = SUB(Name,1,1) & LOWER(SUB(Name,2,19))
!Все буквы имени кроме первой записать маленькими буквами
```

## NUMERIC (проверить числовую строку)

### NUMERIC(*строка*)

#### NUMERIC

Проверяет числовую строку.

*строка*

Строковая константа, переменная или выражение.

Функция **NUMERIC** возвращает 1 (истина), если строка содержит допустимую числовую величину, и 0 (ложь) - если строка содержит недопустимые в числе символы. Допустимыми символами являются цифры от 0 до 9 знак минус спереди, десятичная точка.

Тип возвращаемого значения: LONG

#### Пример:

```
!NUMERIC('1234.56')      возвращает 1
!NUMERIC('1,234.56')     возвращает 0
```

!NUMERIC(' -1234.56')

возвращает 1

!NUMERIC('1234.56-')

возвращает 0

IF NOT NUMERIC(PartNumber)

!Если в поле не число

DO ChkValidPart

! выполнить подпрограмму проверки правильности значения

**RIGHT (выровнять строку вправо)****RIGHT**(строка, длина)**RIGHT**

Выравнивает строку вправо.

*строка*

Строковая константа, переменная или выражение.

*длина*

Числовая константа, переменная или выражение, указывающая длину возвращаемой строки. Если этот параметр опущен, то берется длина параметра строка.

Функция **RIGHT** возвращает выровненную вправо строку. Пробелы в конце значения строки удаляются, затем значение выравнивается вправо и строка возвращается дополненная в начале пробелами.

Тип возвращаемого значения: STRING

**Пример:**

!RIGHT('ABC ')

возвращает ' ABC'

Message = RIGHT(Message)

!Выровнять текст сообщения вправо

**SUB (получить подстроку)****SUB**(строка, позиция, длина)**SUB**

Возвращает часть строки.

*строка*

Строковая константа, переменная или выражение.

*позиция*

Целочисленная константа, переменная или выражение. Будучи положительной указывает положение символа относительно начала строки. Отрицательное же значение указывает положение символа относительно конца строки (т.е. значение -3 указывает на 3-й символ от конца строки).

*длина*

Числовая константа, переменная или выражение, задающее число возвращаемых символов.

Функция **SUB** возвращает подстроку заданной длины из строки начиная с указанной позиции.

Тип возвращаемого значения: STRING



## Функции манипуляций с битами

### BAND (поразрядное И)

**BAND**(значение, маска)

#### **BAND**

Выполняет поразрядную операцию логическое И.

*значение*

Числовая константа, переменная или выражение, битовое представление которого сопоставляется с маской. Если это необходимо, то перед выполнением операции значение преобразуется к типу LONG.

*маска*

Числовая константа, переменная или выражение, представляющее битовую маску. Если это необходимо, то перед выполнением операции маска преобразуется к типу LONG.

Функция **BAND** сопоставляет значение с маской, выполняя операцию поразрядного И между битом значения и битом маски. Возвращаемое значение представляет собой длинное целое, содержащее единицу в тех позициях битов, где и значение и маска содержат единицу, и содержащее ноль в других позициях битов.

Обычно эта функция используется для того, чтобы проверить установлен ли отдельный бит (или несколько битов) в 1 или нет.

Тип возвращаемого значения: LONG

#### Пример:

BAND(0110b,0010b)	возвращает 0010b !0110b = 6, 0010b=2
RateType BYTE	!Тип расценки
Female EQUATE(0001b)	!Маска для мужчин
Male EQUATE(0010b)	!Маска для женщин
Over25 EQUATE(0100b)	!Маска возраста старше 25
CODE	
IF BAND(RateType,Female)	!Если женщина
AND BAND(RateType,Over25)	! и старше 25-ти
DO BaseRate	! использовать основную премию
ELSIF BAND(RateType,Male)	!Если мужчина
DO AdjBase	! скорректировать основную премию
.	!Конец структуры IF

**BOR (поразрядное ИЛИ)****BOR**(значение, маска)**BOR**

Выполняет поразрядную операцию логическое ИЛИ.

*значение*

Числовая константа, переменная или выражение, битовое представление которого сопоставляется с маской. Если это необходимо, то перед выполнением операции значение преобразуется к типу LONG.

*маска*

Числовая константа, переменная или выражение, представляющее битовую маску. Если это необходимо, то перед выполнением операции маска преобразуется к типу LONG.

Функция **BOR** сопоставляет значение с маской, выполняя операцию поразрядного ИЛИ над битом значения и битом маски. Возвращаемое значение представляет собой длинное целое, содержащее единицу в тех позициях битов, где значение, или маска, или оба эти параметры содержат единицу, и содержащее ноль в других позициях битов.

Обычно функция BOR используется для безусловного включения (установки в 1) отдельного бита (или нескольких битов) в переменной.

Тип возвращаемого значения: LONG

**Пример:**

BOR(0110b,0010b)    возвращает 0110b

RateType BYTE

Female EQUATE(0001b)

Male EQUATE(0010b)

Over25 EQUATE(0100b)

CODE

RateType = BOR(RateType,Over25)

RateType = BOR(RateType,Male)

!0110b = 6, 0010b = 2

!Тип расценки

!Маска для мужчин

!Маска для женщин

!Маска возраста старше 25

!Включить бит возраста старше 25

!Установить расценку для мужчин

**BXOR (поразрядное исключающее ИЛИ)****BXOR**(значение, маска)**BXOR**

Выполняет поразрядную операцию исключающее ИЛИ.

*значение*

Числовая константа, переменная или выражение, битовое представление которого сопоставляется с маской. Если это необходимо, то перед выполнением операции значение преобразуется к типу LONG.

*маска*

Числовая константа, переменная или выражение, представляющее битовую маску. Если это необходимо, то перед выполнением операции маска преобразуется к типу LONG.

Функция **BXOR** сопоставляет значение с маской, выполняя операцию поразрядного исключающего ИЛИ над битом значения и битом маски. Возвращаемое значение представляет собой длинное целое, содержащее единицу в тех позициях битов, где значение, или маска, но не оба эти параметры содержат единицу. Нули возвращаются в тех позициях, где биты значения и маски одинаковы.

Обычно функция BOR используется для переключения (установки в 0 или 1) отдельного бита (или нескольких битов) в переменной.

Тип возвращаемого значения: LONG

**Пример:**

BXOR(0110b,0010b)    возвращает 0100b    !0110b = 6, 0100b = 4 0010b = 2

RateType    BYTE    !Тип расценки

Female    EQUATE(0001b)    !Маска для мужчин

Male    EQUATE(0010b)    !Маска для женщин

Over25    EQUATE(0100b)    !Маска возраста старше 25

Over65    EQUATE(1100b)    !Маска возраста старше 65

CODE

RateType = BXOR(RateType,Over65)    !Переключить биты возраста старше 65

**BSHIFT (поразрядный сдвиг)****BSHIFT**(значение, счетчик)**BSHIFT**

Выполняет операцию сдвига битов.

*значение*

Числовая константа, переменная или выражение. Если это необходимо, то перед выполнением операции значение преобразуется к типу LONG.

*счетчик*

Числовая константа, переменная или выражение представляющее число битов, на которое значение должно сдвигаться. Если счетчик положителен, то значение сдвигается влево. Если счетчик отрицательный, то значение сдвигается вправо.

Функция **BSHIFT** сдвигает битовое значение на величину, указанную счетчиком битов. Битовое значение может быть сдвинуто влево (в сторону старших разрядов) или вправо (в сторону младших разрядов). Возникающие свободные позиции заполняют нулевые биты.

Тип возвращаемого значения: LONG

Пример:

BSHIFT(0110b,1)

возвращает 1100b

BSHIFT(0110b,-1)

возвращает 0011b

Varswitch = BSHIFT(20,3)

!Умножить на восемь

Varswitch = BSHIFT(Varswitch,-2)

!Разделить на четыре



## **Операторы и функции даты и времени**

### **Стандартная дата**

---

Стандартная дата - это количество дней, прошедших с 28 декабря 1800 года. Диапазон возможных дат с 1 января 1801 г. (стандартная дата 4) по 31 декабря 2099 г. (стандартная дата 109211). Вне этого диапазона функции работающие с датами будут возвращать неправильное значение. Кроме того, календарь стандартных дат учитывает високосные года в данном диапазоне. Остаток от деления стандартной даты на 7 дает день недели (0 - воскресенье, 1 - понедельник и т.д.)

Для хранения стандартных дат обычно используются переменные типа LONG с шаблоном даты (@D). Дата, введенная по шаблону имеющему две цифры года по умолчанию попадает в столетний диапазон +20 лет - 80 лет от текущего года. Например, введя 01/01/01, получим 01/01/2001, если текущий год (по системным часам) больше чем 1980, а если меньше чем 1980 -й год то введенная дата преобразуется к 01/01/1901.

В Btrieve Record Manager для этого дат используется тип данных DATE. Перед выполнением любых математических функций или функций, работающих с датами, тип DATE неявно преобразуется в LONG, содержащий стандартную дату Clarion. Поэтому для совместимости с файлами Btrieve следует использовать тип DATE, а в других случаях для хранения дат следует использовать переменные типа LONG.

### **Стандартное время**

---

Стандартное время - это количество сотых долей секунды, истекших с полуночи + 1 . Допустимый диапазон значений - от 1 (означающей полночь) до 8640000 (означающее 11:59:59:99). Стандартное время 1 соответствует точно полуночи, что позволяет использовать 0 для определения того факта, что в поле с шаблоном времени ничего не введено. Хотя время выражается с точностью 0.01 сек. системные часы обновляются с частотой 18.2 раз в секунду (приблизительно каждые 5,5 сотых секунды)

Для хранения стандартного времени обычно используются переменные типа LONG с шаблоном времени (@T). В Btrieve Record Manager для этого используется тип данных TIME. Перед выполнением любых математических функций или функций, работающих с временем, тип TIME неявно преобразуется в LONG, содержащий стандартное время Clarion. Поэтому для совместимости с файлами Btrieve следует использовать тип TIME, а в других случаях для хранения времени следует использовать переменные типа LONG.



**SETCLOCK (установить системное время)****SETCLOCK(время)****SETCLOCK**

Устанавливает системное время.

*время*

Числовая константа, переменная или выражение представляющее стандартное время (выраженное в сотых долях секунды, истекших с полуночи).

Оператор **SETCLOCK** устанавливает в системе DOS текущее время.

**Пример:**

SETCLOCK(1)

**DATE (получить стандартное время)****DATE(месяц, день, год)****DATE**

Возвращает стандартное время.

*месяц*

Положительная числовая константа, переменная или выражение, задающее месяц.

*день*

Положительная числовая константа, переменная или выражение, задающее день

*год*

Числовая константа, переменная или выражение, задающее год. Для параметра год являются допустимыми диапазоны значений от 0 до 99 (что подразумевает с 1900 года по 1999-й) или 1801 - 2099.

Функция **DATE** возвращает стандартную дату для заданного года, месяца и дня. Допустимы параметры месяц и день, которые выходят вперед за обычные пределы. 13-ый месяц означает январь следующего года, а 32 января означает 1 февраля. Собственно, функции DATE(12,32,97), DATE(13,1,97) и DATE(1,1,98) все дадут один и тот же результат.

Тип возвращаемого значения: LONG

**Пример:**

HireDate = DATE(Hir:Month,Hir:day,Hir:Year) !Вычислить дату сдачи в прокат

FirstOfMonth = DATE(MONTH(TODAY()),1,YEAR(TODAY())) !Вычислить первый день месяца

**Смотри также:** стандартная дата

**DAY (получить день месяца)****DAY**(*дата*)**DAY**

Возвращает день месяца

*дата*

Числовая константа, переменная или выражение, или метка переменной типа STRING, CSTRING или PSTRING, объявленной с шаблоном даты. Параметр дата должен представлять стандартную дату. Переменные, объявленные с шаблоном даты, автоматически преобразуются в промежуточное значение в стандартном формате даты.

Функция **DAY** вычисляет для заданной стандартной даты день месяца (от 1 до 31).

Тип возвращаемого значения: LONG

**Пример:**

OutDay = DAY(TODAY())

!Получить сегодняшний день

DueDay = DAY(TODAY() +2)

!Вычислить день возвращения

**Смотри также:** стандартная дата

**MONTH (получить месяц)****MONTH**(*дата*)**MONTH**

Возвращает месяц.

*дата*

Числовая константа, переменная или выражение, или метка переменной типа STRING, CSTRING или PSTRING, объявленной с шаблоном даты. Параметр дата должен представлять стандартную дату. Переменные, объявленные с шаблоном даты, автоматически преобразуются в промежуточное значение в стандартном формате даты.

Функция **MONTH** вычисляет для заданной стандартной даты месяц года (от 1 до 12).

Тип возвращаемого значения: LONG

**Пример:**

PayMonth = MONTH(DueDate)

!Получить месяц из стандартной даты

**Смотри также:** стандартная дата

**YEAR (получить год)****YEAR**(*дата*)**YEAR** Возвращает год.

*дата* Числовая константа, переменная или выражение, или метка переменной типа STRING, CSTRING или PSTRING, объявленной с шаблоном даты. Параметр дата должен представлять стандартную дату. Переменные, объявленные с шаблоном даты, автоматически преобразуются в промежуточное значение в стандартном формате даты.

Функция **YEAR** вычисляет для заданной стандартной даты год (от 1801 до 2099).

Тип возвращаемого значения: LONG

**Пример:**

```
IF YEAR>LastOrd) < YEAR(TODAY())  
DO StartNewYear
```

**Смотри также:** стандартная дата

**AGE (получить возраст в заданный день)****AGE**(*дата рождения, базовая дата*)

**AGE** Возвращает длительность промежутка времени.

*дата рождения* Числовое выражение, представляющее собой стандартную дату.

*базовая дата* Числовое выражение, представляющее собой стандартную дату. Если этот параметр опущен, то используется системная дата из DOS.

Функция **AGE** возвращает строку, содержащую время, прошедшее между двумя датами. Возраст возвращается в следующем формате:

от 1 до 60 дней	-	'nn DAYS'
от 61 дня до 24 месяцев	-	'nn MOS'
от 2 до 999 лет	-	'nnn YRS'

Тип возвращаемого значения: STRING

**Пример:**

```
Message = Emp:Name & 'is ' AGE(Emp:DOB, TODAY()) & ' old today.'
```

!Поместить возраст служащего в текст сообщения

**Смотри также:** стандартная дата

## Операторы и функции обращения к DOS

### COMMAND (получить параметр командной строки)

**COMMAND**(*параметр*)

#### **COMMAND**

Возвращает параметры командной строки.

*параметр*

Строковая константа или переменная, содержащая параметр, значение которого должно быть найдено, или номер параметра командной строки значение которого должно быть получено.

Функция **COMMAND** возвращает значение заданного параметра из командной строки, конфигурационного файла или переменной среды DOS. Если заданный параметр не найден, то функция **COMMAND** возвращает пустую строку. Если же параметр указан в нескольких местах, то возвращается только первое найденное значение.

Функция **COMMAND** ищет контекст параметр=значение и возвращает значение. Между ключевым словом, задающим параметр, знаком “равно” и значением не должно быть пробелов. В случае обнаружения параметра в конфигурационном файле или переменной среды возвращается все находящееся справа от знака равно. Значение, возвращаемое для параметра в командной строке, заканчивается на первой запятой или первом пробеле. Если в параметре командной строки требуется пробел или запятая, то все значение справа от знака равно должно быть заключено в двойные кавычки (параметр=”значение”).

Эта функция ищет также параметры командной строки, начинающиеся со знака слэш (/). Если такой параметр найден, то функция возвращает значение этого параметра без слэша. Если параметр функции содержит только число, то она возвращает значение параметра командной строки имеющего заданный параметром порядковый номер. Если параметр функции содержит ‘0’, то функция возвращает минимальный путь в файловой системе, использованный для поиска команды. Этот минимальный путь всегда включает и саму команду (без параметров командной строки), но может и не содержать пути (если команда найдена в текущем каталоге). Если параметр равен ‘1’, то функция возвращает первый параметр командной строки. Если параметр функции представляет собой пустую строку (“”) и операция равна 3, то возвращается целиком вся командная строка, дополненная пробелом спереди.

Тип возвращаемого значения: STRING

#### **Пример:**

```
IF COMMAND('/N') !В командной строке был ключ /N ?  
DO SomeProcess
```

CommandString = COMMAND("")	!Получить все параметры командной строки
CommandString = COMMAND('0')	!Получить всю командную строку
SecondParm = COMMAND('2')	!Получить второй параметр из командной строки

Смотри также: SETCOMMAND

## DIRECTORY (получить список файлов в каталоге)

**DIRECTORY**( *очередь, путь, атрибуты* )

<b>DIRECTORY</b>	Выдает список файлов в каталоге (аналогично команде DIR в DOS).
<i>очередь</i>	Метка структуры QUEUE, в которую будет занесен список файлов. Эта очередь должна иметь точно такую структуру как очередь ff_queue, объявленная в файле EQUATES.CLW.
<i>путь</i>	Строковая константа, переменная или выражение, которое задает путь и маску имен файлов, список которых необходимо получить. Маска может включать символы шаблона (* и ?).
<i>атрибуты</i>	Целочисленная константа, переменная или выражение, которое указывает атрибуты файлов список которых должен быть помещен в очередь.

Процедура **DIRECTORY** возвращает список всех файлов в каталоге, указанном параметром путь, и имеющих соответствующие параметру атрибуты.

Параметр очередь должен представлять собой очередь, начало структуры которой совпадает со следующей структурой находящейся в файле EQUATES.CLW:

```
ff_queue  QUEUE,PRE(ff_),TYPE
name      STRING(13)
date      LONG
time      LONG
size      LONG
attrib    BYTE
END
```

В вашей очереди может быть больше полей, но начинаться она должна с этих пяти полей. В них будет занесены данные о каждом файле в каталоге, заданном соответствующим параметром. Поля date и time будут содержать стандартную в Clarion дату и время (преобразование из внутреннего формата операционной системы производится автоматически).

В параметре атрибуты имеет самостоятельное значение каждый бит, и они указывают, имена каких файлов отбираются в очередь. В файле EQUATES.CLW содержатся следующие

мнемонические соответствия:

ff_:NORMAL	EQUATE(0)	
ff_:READONLY	EQUATE(1)	
ff_:HIDDEN	EQUATE(2)	
ff_:SYSTEM	EQUATE(4)	
ff_:DIRECTORY	EQUATE(10H)	
ff_:ARCHIVE	EQUATE(20H)	! Не совместим с Windows 95

Битовый массив атрибутов используется как фильтр с независимыми компонентами: если складывать числа, представленные мнемоническими именами соответствия, то вы будете получать список файлов с любыми, заданными атрибутами. Это означает, что когда устанавливается просто атрибут ff\_:NORMAL, то в результате выбираются только файлы (подкаталоги не выбираются), не имеющие атрибутов “только чтение”, “скрытый файл”, “системный файл” или атрибута “архивирования”. Если к ff\_:NORMAL прибавить атрибут ff\_:DIRECTORY, то будут выбраны все файлы И подкаталоги данного каталога.

### Пример:

DirectoryList	PROCEDURE
AllFiles	QUEUE,PRE(FIL)
name	STRING(13)
date	LONG
time	LONG
size	LONG
attrib	BYTE
	END
LP	LONG
Recs	LONG

```

CODE
DIRECTORY(AllFiles,'*.*',ff_:DIRECTORY) !Получить список файлов в каталоге
Recs = RECORDS(AllFiles)
LOOP LP = Recs TO 1 BY -1
  GET(AllFiles,LP)
  IF BAND(FIL:Attrib,ff_:DIRECTORY) AND FIL:Name <> '..' AND FIL:Name <> '.'
    CYCLE !пусть остаются подкаталоги
  ELSE
    DELETE(AllFiles) !избавимся от остальных элементов
  END
END
END

```



**PATH (получить текущий каталог DOS)****PATH()**

Функция **PATH** возвращает строку, содержащую текущий диск и каталог.

Тип возвращаемого значения: STRING

**Пример:**

```
IF PATH() = 'C:\'                !Если в корневом каталоге  
MESSAGE('You are in the Root Directory')  !Вывести сообщение
```

Смотри также: SETPATH

**RUNCODE (получить код завершения из DOS)****RUNCODE()**

Функция **RUNCODE** возвращает код возврата, переданный DOS-у последней командой, исполненной оператором RUN. Этот код возврата является тем же самым кодом, который передается оператором HALT в Clarion-программе и который проверяется BATCH-командой DOS-а IF ERRORLEVEL. Возвращаемое этой функцией значение представляет собой длинное целое и может быть любым значением, которое возвращается в DOS порожденной программой в качестве кода завершения.

Порожденная программа в качестве кода завершения может передавать только однобайтовое значение без знака, поэтому для него невозможны отрицательные значения. Это позволяет функции RUNCODE зарезервировать следующие значения для обработки ситуаций, когда код возврата отсутствует:

- 0     нормальное завершение
- 1    программа прервана нажатием комбинации Ctrl-C
- 2    программа завершена с сообщением о критической ошибке
- 3    выход из TSR программы
- 4    программа не выполнялась, проверьте значение функции ERROR

Тип возвращаемого значения: LONG

**Пример:**

```
RUN('Nextprog.exe')            !Выполнить следующую программу  
IF RUNCODE() = -4
```

```
IF ERROR() - 'Not Enough Memory' !Если программа не выполнена из-за нехватки памяти
  SHOW(25,14,'Insufficient memory') ! вывести сообщение
  RETURN ! и завершить процедуру
ELSE
  STOP(ERROR()) ! закончить программу
```

**Смотри также:** RUN, RUNSMALL, HALT

## SETCOMMAND (задать параметры командной строки)

**SETCOMMAND**(*командная строка*)

**SETCOMMAND** Устанавливает параметры командной строки.

*командная строка* Строковая константа, переменная или выражение, содержащее новые параметры командной строки.

Оператор **SETCOMMAND** позволяет внутри программы задать параметры командной строки, которые затем можно прочесть с помощью функции **COMMAND**. **SETCOMMAND** устанавливает для любых ранее заданных в командной строке ключей те же самые значения. Для того, чтобы выключить ключ со слэшем спереди, нужно, задавая в параметре командная строка добавить после ключа знак равно (=).

С помощью **SETCOMMAND** нельзя установить системные переключатели, которые должны задаваться при загрузке программы. Переключатели типа установки виртуальной памяти (CLAVM=), или конфигурационного файла (CLAINI=) должны устанавливаться во время загрузки программы и не могут переустанавливаться с помощью **SETCOMMAND**. А указатель каталога для временных файлов может устанавливаться оператором **SETCOMMAND**.

### Пример:

```
SETCOMMAND('/N') !Добавить параметр /N
SETCOMMAND('/N=') !Выключить параметр /N
```

**Смотри также:** COMMAND

**SETPATH (изменить текущий диск и каталог)****SETPATH**(*диск и путь*)**SETPATH**

Изменяет текущий диск и каталог.

*диск и путь*

Строковая константа или метка переменной типа STRING, CSTRING или PSTRING, содержащей новую спецификацию диска и пути к каталогу.

Оператор **SETPATH** изменяет текущий диск и текущий каталог. Если параметр диск и путь содержит неправильную спецификацию, то выдается код ошибочной ситуации “Path Not Found” (путь не найден) и текущие диск и каталог не изменяются.

Если буква, задающая диск, и двоеточие опущены, то подразумевается текущий диск. Если заданы только диск и двоеточие, то происходит переход в текущий каталог на том диске.

Выдаваемые сообщения об ошибках:

03 Path Not Found (путь не найден)

**Пример:**

SETPATH('C:\LEDGER')

!Изменить текущий каталог

SETPATH(UserPath)

!Перейти в каталог пользователя

## Функции обработки ошибочных ситуаций

### ERROR (получить текст сообщения об ошибке)

#### ERROR()

Функция **ERROR** возвращает строку, содержащую описание ошибочной ситуации, которая возникла. Если ошибочной ситуации не было, то функция возвращает пустую строку.

Тип возвращаемого значения: STRING

Пример:

```
PUT(NameQueue)           !Поместить запись в очередь
IF ERROR() "Queue Entry Not Found" !Если такой записи в очереди нет
ADD(NameQueue)           ! добавить новую запись
IF ERRORCODE() THEN STOP(ERROR()). !Проверить безошибочность добавления
```

### ERRORCODE (получить код ошибки)

#### ERRORCODE()

Функция **ERROR** возвращает код ошибочной ситуации, которая возникла. Если ошибочной ситуации не было, то функция возвращает ноль.

Тип возвращаемого значения: LONG

Пример:

```
ADD(Location)           !Добавить новый элемент
IF ERRORCODE() = 8       !Если не хватает памяти
SHOW(1,5,'Out of Memory') ! вывести сообщение
```

### ERRORFILE (получить имя файла, вызвавшего ошибку)

#### ERRORFILE()

Функция **ERRORFILE** возвращает имя файла, при работе с которым возникла ошибка. Если файл открыт, то возвращается его полная спецификация. Если файл закрыт, то возвращается значение атрибута NAME оператора FILE. Если файл закрыт и в его описании нет атрибута NAME, то возвращается метка оператора FILE. Если ошибочной ситуации не

было или она не связана с файлом, то возвращается пустая строка.

Тип возвращаемого значения: STRING

**Пример:**

```
ADD(Location)                !Добавить элемент
IF ERRORCODE()
  SHOW(1,5,'Error with ' & ERRORFILE()) !Вывести сообщение об ошибке
ASK      ! и ждать реакции оператора
.
```

## **FILEERROR (получить сообщение об ошибке от файлового драйвера)**

### **FILEERROR()**

Функция **FILEERROR** возвращает строку, содержащую оригинальное сообщение об ошибке файловой системы (файлового драйвера), используемого для доступа к файлу. Если ошибки не было, то возвращается пустая строка.

Тип возвращаемого значения: STRING

**Пример:**

```
PUT(NameFile)                !Записать данные
IF FILEERRORCODE()
  MESSAGE(FILEERROR())
RETURN
END
```

**Смотри также:** FILEERRORCODE

## **FILEERRORCODE (получить код ошибки от файлового драйвера)**

### **FILEERRORCODE(*файл*)**

Функция **FILEERRORCODE** возвращает строку, содержащую код оригинального сообщения об ошибке файловой системы (файлового драйвера), используемого для доступа к файлу. Если ошибки не было, то возвращается пустая строка.

Тип возвращаемого значения: STRING

**Пример:**

```
PUT(NameFile)                !Записать данные
IF FILEERRORCODE()
  MESSAGE(FILEERROR())
RETURN
END
```

Смотри также: FILEERROR

## REJECTCODE (получить код причины события EVENT:Rejected)

### REJECTCODE( )

Функция **REJECTCODE** возвращает числовой код причины возникновения события EVENT:Rejected. Если такого события не было, то возвращается ноль. В файле EQUATES.CLW содержатся мнемонические имена соответствия для значений, возвращаемых функцией REJECTCODE:

```
REJECT:RangeHigh      ! Превышение верхней границы диапазона в SPIN
REJECT:RangeLow       ! Выход за нижнюю границу диапазона в SPIN
REJECT:Range          ! Прочая ошибка, связанная с диапазоном
REJECT:Invalid        ! Введены неправильные данные
```

Тип возвращаемого значения: LONG

### Пример:

```
CASE EVENT()
OF EVENT:Rejected
  EXECUTE REJECTCODE()
  MESSAGE('Input invalid — out of range — too high')
  MESSAGE('Input invalid — out of range — too low')
  MESSAGE('Input invalid — out of range')
  MESSAGE('Input invalid')
END
END
```

## Другие операторы и функции

### ADDRESS (получить адрес памяти)

**ADDRESS**(*сегмент*[:*смещение*])

*переменная*

*процедура*

#### ADDRESS

Возвращает адрес памяти переменной.

*сегмент*

Метка элемента данных или целочисленной переменной или константы, содержащей сегментную часть абсолютного адреса памяти в формате сегмент:смещение, принятом в реальном режиме работы процессора.

*смещение*

Целочисленная переменная или константа, содержащая вторую составляющую - смещение абсолютного адреса памяти в формате сегмент:смещение, принятом в реальном режиме работы процессора.

*переменная*

Метка элемента данных.

*процедура*

Метка процедуры или функции.

Функция **ADDRESS** возвращает длинное целое, содержащее адрес памяти в стандартном формате селектор:смещение, где селектор представляет собой ссылку на элемент таблицы дескрипторов защищенного режима.

ADDRESS(*переменная*)

Возвращает адрес заданного параметром элемента данных в защищенном режиме.

ADDRESS(*сегмент:смещение*)

Возвращает адрес памяти для защищенного режима в формате селектор:смещение. Это позволяет избежать нарушения защиты памяти при прямом обращении в память в защищенном режиме.

ADDRESS(*процедура*)

Возвращает для защищенного режима адрес процедуры или функции, указанной параметром.

Функция **ADDRESS** позволяет передавать адрес переменной во внешнюю, библиотечную процедуру или функцию, написанную на языке отличном от Clarion.

Тип возвращаемого значения: LONG

#### Пример:

MAP

ClarionProc

!Процедура на языке Clarion

MODULE('External.Obj') !Внешняя библиотека

ExternVarProc(LONG) !Функция на C, принимающая адрес переменной

ExternProc(LONG) !Функция на C, принимающая адрес процедуры

..

Var1 CSTRING(10) !Описать строку, оканчивающуюся нулевым символом

DestVar USHORT !Переменная для считывания в нее данных оператором PEEK

CODE

ExternVarProc(ADDRESS(Var1)) !Передать адрес Var1 во внешнюю процедуру

ExternProc(ADDRESS(ClarionProc)) !Передать адрес ClarionProc

ClarionProc PROCEDURE !Процедура на языке Clarion

CODE

RETURN

БЕЕР (подать звуковой сигнал)

БЕЕР(*звук*)

БЕЕР

звук

Генерирует подачу звукового сигнала через динамик.

Числовая константа, переменная, выражение или мнемоническое имя для

подачи звукового сигнала в Windows.

Оператор **БЕЕР** устанавливает подачу звукового сигнала через динамик системного блока компьютера. Это стандартные звуковые сигналы описанные в разделе [sound] файла WIN.INI . Операторы EQUATE для стандартных значений содержатся в файле EQUATES.CLW .

Пример:

IF ERRORCODE()

БЕЕР(-1)

STOP(ERROR())

END

!Если непредвиденная ошибка

! подать стандартный сигнал

! остановиться по ошибке

CALL (обратиться к процедуре из DLL)

CALL(*файл, процедура*)

CALL

файл

процедура

Обращается к процедуре, которая представлена прототипом в структуре MAP программы, и которая располагается в стандартном DLL Windows.

Строковая константа, переменная или выражение, содержащее имя (с расширением) файла DLL, который следует открыть. Она должна содержать полный путь к библиотеке.

Строковая константа, переменная или выражение, содержащее имя процедуры, которую следует вызвать (которая не может принимать параметров и возвращать значение. Она может содержать порядковый номер процедуры

Функция **CALL** позволяет обратиться к процедуре, которая располагается в стандартном



DLL Windows. Эту процедуру не нужно описывать прототипом в структуре MAP программы. Файл DLL загружается в память (если он еще не был загружен).

Эта функция возвращает ноль (0) в случае успешного обращения к процедуре, и один из следующих кодов ошибки в противном случае:

- 2      Файл не найден
- 3      Путь не найден
- 5      Попытка загрузить программу, а не библиотеку
- 6      Библиотека требует отдельных сегментов данных для каждой задачи
- 10     Неверная версия Windows
- 11     Неверная структура файла .EXE ( программа для DOS или ошибка в заголовке программы)
- 12     Прикладная программа для OS/2
- 13     Прикладная программа для DOS 4.0
- 14     Неизвестный тип .EXE модуля
- 15     Попытка загрузить .EXE модуль для более ранней версии Windows Эта ошибка не возникает, если Windows работает в реальном режиме.
- 16     Попытка загрузить вторую копию .EXE модуля, содержащий несколько сегментов данных допускающих запись в них.
- 17     Ошибка EMS памяти при второй загрузке .DLL
- 18     Попытка загрузить прикладную программу для защищенного режима, в то время как Windows работает в реальном.

Тип возвращаемого значения: LONG

### Пример:

```
X# = CALL(?CUSTOM.DLL?,?1?)      !Вызвать первую процедуру в библиотеке
CUSTOM.DLL
IF X# THEN STOP(X#).              !Проверить успешность выполнения
```

## CHOOSE (получить выбранное значение)

**CHOOSE**(условие , значение, значение [,значение...])

### CHOOSE

*условие*

Возвращает выбранное из списка возможных значений.

Константа, переменная или выражение, по которому определяется какое значение из списка возвращается. Это может быть или выражение, дающее положительное целое число, или условное выражение..

*значение*

Константа, переменная или выражение значение которого функция должна вернуть. Этот параметр не может иметь тип DECIMAL и STRING.

Функция **CHOOSE** вычисляет выражение-условие и возвращает соответствующее ему значение из списка. Если условие дает в результате положительное целое число, то это число и определяет номер значения из списка, которое функция должна вернуть. Если условие дает результат, выходящий за допустимый предел, то **CHOOSE** возвращает последнее значение-параметр. Если условие представляет собой логическое выражение, которое дает в результате значение истина или ложь, то функция **CHOOSE** возвращает первое значение из списка в первом случае и второе значение - во втором.

Тип возвращаемого значения зависит от типов данных значений в списке:

Все значения параметров	Возвращаемое значение
LONG	LONG
DECIMAL или LONG	DECIMAL
STRING	STRING
DECIMAL, LONG, или STRING	DECIMAL
что-либо еще	REAL

Тип возвращаемого значения: LONG, DECIMAL, STRING, или REAL

### Пример:

!CHOOSE(4, 'A', 'B', 'C', 'D', 'E') возвращает 'D'

!CHOOSE(1 > 2, 'A', 'B') возвращает 'B'

?MyControl{PROP:Hide} = CHOOSE(SomeField = 0, TRUE, FALSE)

!На основе значения некоего поля решить скрыть или раскрыть поле

MyView{PROP:Filter} = 'Weight > CHOOSE(Sex = 'M', 180, 120)'

!Фильтр для структуры VIEW выбирающий мужчин и женщин с избыточным весом

## MAXIMUM (получить максимальное значение индекса)

**MAXIMUM**(*переменная, индекс*)

### MAXIMUM

*переменная*

*индекс*

Возвращает максимальное значение индекса.

Метка переменной, объявленной с атрибутом DIM.

Числовая константа, переменная или выражение, задающее номер индекса массива. Параметр индекс указывает, какой из индексов массива передается функции.

Функция **MAXIMUM** возвращает максимальное значение для индекса явно объявленного массива. Эта функция не оперирует с неявно объявленными для переменных типа STRING, CSTRING и PSTRING массивами. Обычно она используется для определения размера массива, переданного в процедуру или функцию в качестве параметра.

Тип возвращаемого значения: LONG

### Пример:

```

Array  BYTE,DIM(14,12)           !Определить двумерный массив
Для приведенного выше массива: MAXIMUM(Array.1)    возвращает 10
MAXIMUM(Array,2)                  возвращает 12
CODE
LOOP X# = 1 TO MAXIMUM(Array,1)    !Цикл по первому индексу
  LOOP Y# = 1 TO MAXIMUM(Array,2)  ! Цикл по второму индексу
    Array[X#,Y#] = 27              ! присвоить каждому элементу значение по умолчанию
  ..                               !Конец обоих циклов

```

**Смотри также:** DIM, Массивы в качестве параметров процедур и функций.

## OMITTED (проверить не опущены ли параметры)

**OMITTED**(номер параметра)

**OMITTED**                      Проверяет не опущен ли параметр.  
*номер параметра*            Целочисленная константа или переменная, которая задает номер параметра, который необходимо проверить.

Функция **OMITTED** проверяет был ли параметр в списке параметров во время обращения к функции или процедуре. Если параметр, указанный номером параметра опущен, то возвращаемое значение равно 1 (истина). И возвращаемое значение равно 0 (ложь), если параметр передан. Любые номера параметров после последнего переданного рассматриваются как номера опущенных параметров.

Параметр может опускаться, если в прототипе процедуры или функции структуры MAP, его тип данных заключен в угловые скобки.

Тип возвращаемого значения: LONG

### Пример:

```

PROGRAM
MAP
  SomeProc(String,<LONG>,String)
  SomeFunction(String,<LONG>),String
END
CODE
  SomeProc(Field1,,Field3)
Для приведенного выше вызова процедуры:

```

OMITTED(1) returns 0  
 OMITTED(2) returns 1  
 OMITTED(3) returns 0  
 OMITTED(4) returns 1

**Смотри также:** Прототипы процедур и функций

## PEEK (прочитать данные из памяти)

**PEEK**(*сегмент:смещение, переменная*)

**PEEK** Считывает данные из памяти.

*сегмент:смещение* Числовая константа, переменная или выражение, которое задает адрес области памяти. Сегмент должен размещаться в старших двух байтах, а смещение - в младших. Для того, чтобы обеспечить 32-х разрядную точность, для хранения промежуточного значения используется целая часть переменной типа REAL. В этом параметре всегда следует использовать функцию ADDRESS, чтобы гарантировать правильность адреса (селектор:смещение) в защищенном режиме.

*переменная* Имя переменной, в которую следует занести считываемые данные.

Оператор **PEEK** считывает данные из области памяти по адресу сегмент:смещение и заносит их в переменную. Этим оператором считывается столько байт, сколько требуется для заполнения переменной.

Если вы считываете данные по адресу памяти, принадлежащей другой программе, легко получить ошибочную ситуацию General Protection Fault (GPF) (нарушение защиты общего характера), так что следует с великой осторожностью использовать оператор PEEK. Для выполнения того, что вы хотите сделать с помощью оператора PEEK, обычно имеется стандартная функция в API - интерфейсе прикладного программирования Windows, и предпочтительнее использовать ее вместо того, чтобы использовать PEEK.

### Пример:

Segment	USHORT
Offset	USHORT
Dest1	BYTE
Dest2	SHORT
Dest3	REAL
KeyboardFlag	BYTE
CODE	

PEEK(ADDRESS(Segment,Offset),Dest1) !Прочитать один байт  
 PEEK(ADDRESS(Segment,Offset),Dest2) ! Прочитать два байта  
 PEEK(ADDRESS(Segment,Offset),Dest3) ! Прочитать восемь байт

PEEK(ADDRESS(0040h,0017h),KeyboardFlag)!Прочитать байт состояния клавиатуры

**Смотри также:** POKE, ADDRESS

## POKE (записать данные в память)

**POKE**(*сегмент:смещение,переменная*)

### POKE

Записывает данные в память

*сегмент:смещение*

Числовая константа, переменная или выражение, которое задает адрес области памяти. Сегмент должен размежаться в старших двух байтах, а смещение - в младших. Для того, чтобы обеспечить 32-х разрядную точность, для хранения промежуточного значения используется целая часть переменной типа REAL. В этом параметре всегда следует использовать функцию ADDRESS, чтобы гарантировать правильность адреса (селектор:смещение) в защищенном режиме.

*переменная*

Имя переменной.

Оператор **POKE** записывает содержимое переменной в память по адресу, указанному параметром сегмент:смещение. Эти оператором записывается столько байт, сколько содержит исходная переменная.

Если вы записываете данные по адресу памяти, принадлежащей другой программе, легко получить ошибочную ситуацию General Protection Fault (GPF) (нарушение защиты общего характера), так что следует с великой осторожностью использовать оператор POKE. Для выполнения того, что вы хотите сделать с помощью оператора POKE, обычно имеется стандартная функция в API - интерфейсе прикладного программирования Windows, и предпочтительнее использовать ее вместо того, чтобы использовать POKE.

### Пример:

Segment USHORT

Offset USHORT

Source1 BYTE

Source2 SHORT

Source3 REAL

KeyboardFlag BYTE

CODE

POKE(ADDRESS(Segment,Offset),Source1) !Записать один байт в память

POKE(ADDRESS(Segment,Offset),Source2) ! Записать два байта в память

POKE(ADDRESS(Segment,Offset),Source3) ! Записать восемь байт в память

PEEK(ADDRESS(0040h,0017h),KeyboardFlag) !Прочитать байт состояния клавиатуры

KeyboardFlag = BOR(KeyboardFlag,40h) ! включить caps lock

POKE(ADDRESS(0040h,0017h),KeyboardFlag) ! и записать байт обратно

**Смотри также:** PEEK, ADDRESS



## Приложение А. Библиотека DDE, OLE, и .OCX

### Динамический Обмен Данными

#### Введение в DDE

---

Динамический обмен данными (Dynamic Data Exchange - DDE) - мощное средство системы Windows, которое позволяет пользователю осуществлять доступ к данным параллельно исполняемого Windows-приложения. В своей программе пользователь может работать с этими данными в их естественном формате ( как они определены в “родном” приложении) и с теми значениями, которые данные имеют на текущий момент.

Суть DDE состоит в установлении “диалогов” (каналов связи) между двумя параллельно исполняемыми Windows-приложениями. Одно из приложений, предоставляя данные, выступает в роли DDE-сервера, другое, получая данные, выступает в роли DDE-клиента. Любое приложение может быть как DDE-клиентом, получая данные от некоторого приложения, так и DDE-сервером, предоставляя данные некоторому приложению. Между DDE-сервером и DDE-клиентом можно одновременно установить несколько “диалогов”.

Для того, чтобы стать DDE-сервером, Clarion-приложение должно:

- \* Открыть, по крайней мере, одно окно, так как с каждым DDE-сервером должно быть связано окно.
- \* Зарегистрироваться в Windows в качестве DDE-сервера, используя функцию DDESERVER.
- \* Используя оператор DDEWRITE, предоставить запрашиваемые данные клиенту.
- \* Когда отпадет потребность в DDE, закрыть канал связи, используя оператор DDECLOSE.

Если пользователь завершает сервер-приложение или закрывает окно, которое инициировало диалог, то автоматически закрывается и сам диалог.

Для того, чтобы стать DDE-клиентом, Clarion-приложение должно:

- \* Открыть - в качестве клиента - канал связи с DDE-сервером, используя функцию DDECLIENT.
- \* С помощью оператора DDEREAD запросить у сервера данные, либо, используя оператор DDEEXECUTE, осуществить запрос на обслуживание.
- \* Когда отпадет потребность в DDE, закрыть канал связи, используя оператор DDECLOSE.

Если пользователь завершает программу или закрывает окно клиента, то автоматически закрывается и сам диалог.

Прототипы функций динамического обмена содержатся в файле DDE.CLW, который должен вставляться оператором INCLUDE в MAP структуру вышей программы. Процесс динамического обмена данными инициирует не связанные с экранными объектами DDE-события для АСCEPT-цикла того окна в программе-сервере, и программе-клиенте, которое установило канал связи между приложениями.

## DDE События

---

DDE-процесс регулируется рядом не связанных с полем DDE-событий. Эти события направляются АСCEPT-циклу того окна и сервера, и клиента, которое установило канал связи между приложениями.

Если Clarion-приложение выступает в качестве сервера, то для него будут инициироваться следующие события:

EVENT:DDErequest Клиент осуществил запрос элемента данных.  
EVENT:DDEadvise Клиент осуществил запрос на предоставление элемента данных  
всякий раз, когда данные обновляются.  
EVENT: DDEexecute Клиент исполнил оператор DDEEXECUTE.  
EVENT:DDEpoke Клиент прислал незапрашивавшиеся данные

Если Clarion-приложение выступает в качестве клиента, то для него будут инициироваться следующие события:

EVENT:DDEdata Сервер предоставил обновленный элемент данных.  
EVENT: DDEclosed Сервер закрыл канал связи.

При возникновении DDE-события, используя приведенные ниже функции, можно выяснить причину этого события:

- \* DDECHANNEL() возвращает номер канала, открытого DDE-сервером или DDE-клиентом.
- \* DDEITEM() возвращает строку, которая была послана серверу при выполнении операторов DDEREAD или DDEEXECUTE, с именем элемента данных или с наименованием команды.
- \* DDEAPP() возвращает имя приложения.
- \* DDETOPIC() возвращает имя раздела данных.

После того, как Clarion-программа создала DDE-сервер, внешние клиенты могут



связаться с сервером и запросить данные. Каждый запрос данных сопровождается строкой (формат которой известен сервер-программе), где конкретизируется запрашиваемый элемент данных. Если значение требуемого элемента уже известно Clarion-серверу, то сервер автоматически снабжает этим значением клиента, без порождения какого-либо события. В противном случае для цикла ACCEPT окна сервера инициируется одно из событий EVENT:DDErequest или EVENT:DDEadvise.

После того, как Clarion-программа создала DDE-клиента, он может связываться с внешними серверами для получения данных. Если сервер первый раз предоставляет значение требуемого элемента, то клиент получает это значение автоматически, без порождения какого-либо события. Если клиент установил с сервером связь типа “горячей линии”, то всякий раз, когда сервер предоставляет клиенту обновленное значение элемента данных, для ACCEPT-цикла окна клиента инициируется событие EVENT:DDEdata.

## DDE Функции

### DDESERVER (возвратить номер канала DDE- сервера)

**DDESERVER**( [ приложение ] [, раздел ] )

<b>DDESERVER</b>	Возвращает номер канала нового DDE-сервера.
<i>приложение</i>	Строковая константа или переменная для указания имени приложения. Обычно это - имя приложения. Если параметр не указан, то используется имя файла (без расширения) программы.
<i>раздел</i>	Строковая константа или переменная для указания имени раздела данных, относящихся к приложению. Если параметр не указан, то приложение будет удовлетворять любой запрос данных.

Функция DDESERVER возвращает номер канала нового DDE-сервера, которым идентифицируются приложение и раздел. Номер канала определяет тот раздел, данные которого будут предоставляться приложением. Это дает возможность одному Clarion-приложению регистрироваться в качестве сервера нескольких разделов.

Тип возвращаемых данных: LONG

#### Пример:

```

DDERetVal  STRING(20)
WinOne     WINDOW,AT(0,0,160,400)
            ENTRY(@s20),USE(DDERetVal)
            END
MyServer   LONG
    
```

```
CODE
OPEN(WinOne)
MyServer = DDESERVER('MyApp', 'DataEntered') !Регистрируется сервер
ACCEPT
CASE EVENT()
OF EVENT:DDErequest !Обслуживание однократного запроса данных
  DDEWRITE(MyServer, DDE:manual, 'DataEntered', DDERetVal)
  !Однократное предоставление данных
OF EVENT:DDEadvise !Обслуживание запроса на постоянное обновление
  DDEWRITE(MyServer, 15, 'DataEntered', DDERetVal)
  !Отслеживать изменения каждые 15 секунд
  ! и предоставить данные как только они обновились
END
END
```

**Смотри также:** DDECLIENT, DDEWRITE

## **DDECLIENT (возвратить номер канала DDE-клиента)**

**DDECLIENT**( [ *приложение* ] [, *раздел* ] )

### **DDECLIENT**

Возвращает номер канала нового DDE-клиента.

*приложение*

Строковая константа или переменная для указания имени сервер-приложения, с которым нужно установить связь. Обычно это - имя приложения. Если параметр не указан, то используется имя приложения первого из зарегистрированных DDE-серверов.

*раздел*

Строковая константа или переменная для указания имени раздела данных, относящихся к приложению. Если параметр не указан, то используется первый из разделов ранее определенных в приложении.

Функция DDECLIENT возвращает номер канала нового DDE-клиента, которым идентифицируются приложение и раздел. Если приложение не было запущено на исполнение, то DDECLIENT возвращает нулевое значение (0).

Обычно, когда DDE-канал открывается клиентом, приложение - это имя сервер-приложения. Раздел - это строка, содержимое которой используется либо как имя доступного раздела приложения, когда это приложение регистрирует его в Windows, либо как некоторое значение, которое говорит приложению какие данные оно должно предоставить. Приложения и разделы, зарегистрированные на данный момент в Windows, можно просмотреть используя функцию DDEQUERY.

Тип возвращаемых данных: LONG

**Пример:**

```

DDEReadVal REAL
WinOne WINDOW,AT(0,0,160,400)
    ENTRY(@s20),USE(DDEReadVal)
    END
ExcelServer LONG
CODE
OPEN(WinOne)
ExcelServer = DDECLIENT('Excel','MySheet.XLS') !Создать клиента Excel-таблицы
IF NOT ExcelServer !Если сервер не был запущен, то
    MESSAGE('Please start Excel') ! предложить пользователю запустить его
    RETURN ! и осуществить следующую попытку
END
DDEREAD(ExcelServer,DDE:auto,'R5C5',DDEReadVal)
ACCEPT
CASE EVENT()
OF EVENT:DDEdata !Когда обновленные данные поступают от Excel
    PassedData(DDEReadVal) ! - обработать их
END
END

```

**Смотри также:** DDEQUERY, DDEWRITE, DDESERVER

## DDEQUERY (просмотр зарегистрированных DDE-серверов)

**DDEQUERY** ( [ *приложение* ] [ , *раздел* ] )

**DDEQUERY** Возвращает список зарегистрированных на данный момент DDE-серверов.

*приложение* Строковая константа или переменная для указания имени опрашиваемого приложения. Для большинства приложений - это имя приложения. Если параметр не указан, то возвращается список всех приложений, которые зарегистрированы вместе с указанным разделом.

*раздел* Строковая константа или переменная для указания имени опрашиваемого раздела данных, относящихся к приложению. Если параметр не указан, то возвращается список всех разделов приложения.

Функция **DDEQUERY** возвращает строку, содержащую имена приложений и их разделов зарегистрированных на данный момент DDE-серверов.

Если параметр *раздел* опущен, то в строке будут указаны все разделы, принадлежащие данному приложению. Если не указан параметр *приложение*, то в строке будут перечислены все зарегистрированные приложения с данным разделом. Если же не определены оба параметра, то **DDEQUERY** возвратит все зарегистрированные на данный момент DDE-серверы.

Данные в возвращаемой строке представляются в формате приложение:раздел, где пары приложение, раздел отделяются - когда их больше одной - друг от друга символом запятой (например, 'Excel:MySheet.XLS,ClarionApp:DataFile.DAT').

Тип возвращаемых данных: STRING

### Пример:

!This example code does not handle DDEADVISE

```
WinOne WINDOW,AT(0,0,160,400)
```

```
END
```

```
SomeServer LONG
```

```
ServerString STRING(200)
```

```
CODE
```

```
OPEN(WinOne)
```

```
LOOP
```

```
ServerString = DDEQUERY() !Указать все серверы
```

```
IF NOT INSTRING('SomeApp:MyTopic',ServerString,1,1)
```

```
MESSAGE('Open SomeApp, Please')
```

```
ELSE
```

```
BREAK
```

```
END
```

```
END
```

```
SomeServer = DDECLIENT('SomeApp','MyTopic') !Стать клиентом сервера 'SomeApp'
```

```
ACCEPT
```

```
END
```

```
DDECLOSE(SomeServer)
```

## DDECHANNEL (получить номер DDE-канала)

**DDECHANNEL( )**

Функция **DDECHANNEL** возвращает целое число типа LONG, определяющее номер DDE-канала, который инициировал последнее по времени DDE-событие для клиент- или сервер-приложения. То же самое значение возвращается функциями DDESERVER или DDECLIENT при установлении DDE-канала.

Тип возвращаемых данных: LONG

### Пример:

```
WinOne WINDOW,AT(0,0,160,400)
```

```
END
```

```
TimeServer LONG
```

```
DateServer LONG
```

```

FormatTime  STRING(5)
FormatDate  STRING(8)
CODE
OPEN(WinOne)
TimeServer = DDESERVER('SomeApp','Time') !Регистрация сервера
DateServer = DDESERVER('SomeApp','Date') !Регистрация сервера
ACCEPT
CASE EVENT()
OF EVENT:DDErequest
CASE DDECHANNEL()
OF TimeServer
FormatTime = FORMAT(CLOCK(),@T1)
DDEWRITE(TimeServer,DDE:manual,'Time',FormatTime)
OF DateServer
FormatDate = FORMAT(TODAY(),@D1)
DDEWRITE(DateServer,DDE:manual,'Date',FormatDate)
END
END
END

```

## DDEAPP (получить имя сервер-приложения)

### DDEAPP( )

Функция **DDEAPP** возвращает строку, содержащую имя приложения того DDE-канала, который инициировал последнее по времени DDE-событие. Обычно - это имя, указываемое первым параметром в функциях DDESERVER или DDECLIENT при установлении DDE-канала.

Тип возвращаемых данных: STRING

### Пример:

```

ClientApp  STRING(20)
WinOne     WINDOW,AT(0,0,160,400)
           STRING(@S20),AT(5,5,90,20),USE(ClientApp)
           END
TimeServer                                LONG
DateServer                                LONG
FormatTime  STRING(5)
FormatDate  STRING(8)

CODE
OPEN(WinOne)
TimeServer = DDESERVER('SomeApp','Time') !Регистрация сервера
DateServer = DDESERVER('SomeApp','Date') !Регистрация сервера

```

```

ACCEPT
CASE EVENT()
OF EVENT:DDErequest
CASE DDECHANNEL()
OF TimeServer
  ClientApp = DDEAPP()           !Получить имя клиента
  DISPLAY                       ! и отобразить на экран
  FormatTime = FORMAT(CLOCK(),@T1)
  DDEWRITE(TimeServer,DDE:manual,'Time',FormatTime)
OF DateServer
  ClientApp = DDEAPP()           ! Получить имя клиента
  DISPLAY                       ! и отобразить на экран
  FormatDate = FORMAT(TODAY(),@D1)
  DDEWRITE(DateServer,DDE:manual,'Date',FormatDate)
END
END
END

```

## **DDEITEM (получить имя элемента данных сервера)**

### **DDEITEM( )**

Для имевшего место DDE-события функция DDEITEM возвращает строку с наименованием элемента динамического обмена между клиентом и сервером. Под элементом подразумевается либо элемент, запрошенный оператором DDEREAD, либо элемент данных, предоставленный DDEPOKE.

Тип возвращаемых данных: STRING

### **Пример:**

```

WinOne  WINDOW,AT(0,0,160,400)
      END
Server  LONG
FormatTime  STRING(5)
FormatDate  STRING(8)
CODE
OPEN(WinOne)
Server = DDESERVER('SomeApp','Clock') !Зарегистр. сервер для раздела 'Clock'
ACCEPT
CASE EVENT()
OF EVENT:DDErequest
CASE DDEITEM()
OF 'Time'
  FormatTime = FORMAT(CLOCK(),@T1)
  DDEWRITE(Server,DDE:manual,'Time',FormatTime)

```

```

OF 'Date'
  FormatDate = FORMAT(TODAY(),@D1)
  DDEWRITE(Server,DDE:manual,'Date',FormatDate)
END
OF EVENT:DDEadvise
CASE DDEITEM()
OF 'Time'
  FormatTime = FORMAT(CLOCK(),@T1)
  DDEWRITE(Server,1,'Time',FormatTime)
OF 'Date'
  FormatDate = FORMAT(TODAY(),@D1)
  DDEWRITE(Server,60,'Date',FormatDate)
END
END
END

```

**Смотри также:** DDEREAD, DDEEXECUTE

## **DDETOPIC (получить имя раздела сервера)**

**DDETOPIC( )**

**DDETOPIC** возвращает строку, содержащую имя раздела того DDE-канала, который инициировал последнее по времени DDE-событие.

Тип возвращаемых данных: STRING

**Пример:**

```

WinOne WINDOW,AT(0,0,160,400)
  END
TimeServer LONG
DateServer LONG
FormatTime STRING(5)
FormatDate STRING(8)
CODE
OPEN(WinOne)
TimeServer = DDESERVER('SomeApp')!Регистрация сервера
ACCEPT
CASE EVENT()
OF EVENT:DDErequest
CASE DDETOPIC() !Получить имя запрошенного раздела
OF 'Time'
  FormatTime = FORMAT(CLOCK(),@T1)
  DDEWRITE(TimeServer,DDE:manual,'Time',FormatTime)
OF 'Date'

```

```
FormatDate = FORMAT(TODAY(),@D1)
DDEWRITE(DateServer,DDE:manual,'Date',FormatDate)
```

```
...
```

**Смотри также:** DDEREAD

## **DDEVALUE (получить значение данных, посланных серверу)**

### **DDEVALUE( )**

Функция **DDEVALUE** возвращает строку с данными, посланными DDE-серверу Clarion-приложения оператором DDEPOKE, либо командой, выполненной оператором DDEEXECUTE.

Тип возвращаемых данных: STRING

**Пример:**

```
WinOne WINDOW,AT(0,0,160,400)
    END
TimeServer LONG
TimeStamp FILE,DRIVER(ASCII),PRE(Tim)
Record RECORD
FormatTime STRING(5)
FormatDate STRING(8)
Message STRING(50)
    ..
CODE
OPEN(WinOne)
TimeServer = DDESERVER('TimeStamp') !Зарегистрировать сервер
ACCEPT
CASE EVENT()
OF EVENT:DDEpoke
    OPEN(TimeStamp)
    Tim:FormatTime = FORMAT(CLOCK(),@T1)
    Tim:FormatDate = FORMAT(TODAY(),@D1)
    Tim:Message = DDEVALUE() !Получить данные
    ADD(TimeStamp)
    CLOSE(TimeStamp)
    CYCLE !Подтвердить получение
END
END
```

**Смотри также:** DDEPOKE



## DDE Процедуры

### DDEREAD (получить данные от DDE-сервера)

**DDEREAD**( канал, режим, элемент [, переменная ] )

<b>DDEREAD</b>	Получает данные из ранее открытого канала DDE-клиента.
<i>канал</i>	Целочисленная константа типа LONG или переменная, указывающие канал клиента - то значение, которое возвращает функция DDECLIENT.
<i>режим</i>	Символическое имя, определяющее тип канала передачи данных: DDE:auto, DDE:manual или DDE:remove (определены в файле EQUATES.CLW).
<i>элемент</i>	Строковая константа или переменная, указывающие связанное с приложением имя запрашиваемого элемента данных.
<i>переменная</i>	Имя переменной, в которую будут занесены запрашиваемые данные. Если параметр не указан и установлен режим DDE:remove, то обрываются все связи с элементом .

Процедура **DDEREAD** дает возможность программе DDE-клиента читать данные из канала в переменную. Параметром режим определяется способ обновления данных. Значение параметра элемент в качестве строки передается сервер-приложению для указания элемента данных, который запрашивается клиентом. Формат и структура строки элемента зависит от сервер-приложения.

Если установлен режим DDE:auto, то сервер автоматически обновляет переменную (“горячая линия”). Если установлен режим DDE:manual, то переменная обновляется однократно, и для обнаружения изменения значения данных нужно снова послать серверу DDEREAD запрос (“холодная линия”). Если для параметра режим указано значение DDE:remove, то разрывается ранее установленная “горячая линия” связи с переменной. Если для параметра режим указано значение DDE:remove, а параметр переменная не указан, то разрываются все ранее установленные “горячие линии” связи с элементом, независимо от того, какие переменные были с ним связаны. Поэтому, для обнаружения изменения значения данных клиенту нужно снова послать серверу DDEREAD запрос.

#### Сообщения об ошибках:

601 Invalid DDE Channel	- Недопустимый DDE-канал
602 DDE Channel Not Open	- Не открыт DDE-канал
605 Time Out	- Ошибка по “тайм-ауту”

Генерируемые события:

Клиент-приложению посылаются следующие события:

EVENT:DDEdata   Сервер прислал обновленный элемент данных по “горячей линии”.  
 EVENT:DDEclosed   Сервер завершил диалог.

### Пример:

```
WinOne   WINDOW,AT(0,0,160,400)
      END
ExcelServer   LONG(0)
DDEReadVal   REAL
CODE
OPEN(WinOne)
ExcelServer = DDECLIENT('Excel', 'MySheet.XLS') !Клиент Excel таблицы
IF NOT ExcelServer                           !Если сервер не был запущен -
  MESSAGE('Please start Excel')           ! предложить пользователю запустить его
  CLOSE(WinOne)
  RETURN
END
END
DDEREAD(ExcelServer,DDE:auto,'R5C5',DDEReadVal)
      !Запрос серверу на постоянное обновление
ACCEPT
CASE EVENT()
OF EVENT:DDEdata                           !При поступлении от Excel обновленных данных
  PassedData(DDEReadVal)               ! вызвать процедуру их обработки
END
END
```

Смотри также: DDEQUERY, DDEWRITE

## DDEWRITE (предоставить данные DDE-клиенту)

**DDEWRITE**( *канал, режим, элемент [ , переменная ]* )

### DDEWRITE

*канал*

*режим*

*элемент*

Передает данные в открытый канал DDE-сервера.  
 Целочисленная константа типа LONG или переменная, указывающие канал сервера - то значение, которое возвращает функция DDESERVER.  
 Либо целочисленная константа или переменная для указания интервала времени (в секундах), в по истечению которого каждый раз производится опрос переменной для проверки изменения ее значения, либо символическое имя, определяющее тип канала передачи данных: DDE:auto, DDE>manual или DDE:remove (определены в файле EQUATES.CLW).  
 Строковая константа или переменная, указывающие связанное с

приложением имя предоставляемого элемента данных.  
*переменная* Имя переменной предоставляющей данные. Если параметр не указан и установлен режим DDE:remove, то обрываются все каналы связи с элементом.

Процедура **DDEWRITE** дает возможность программе DDE-сервера предоставлять клиенту значение переменной. Параметром режим определяется тип обновления данных. Значение параметра элемент представляет собой строку для указания предоставляемого элемента данных. Формат и структура строки элемента зависит от сервер-приложения. Параметром режим определяется способ обновления данных.

Если параметр режим принимает значение DDE:auto, то клиент-программа получает текущее значение переменной и на все последующие запросы этого (или другого) клиента внутренние библиотеки будут предоставлять тоже самое значение. Когда клиент запросил обмен данными по “горячей линии”, то Clarion-программа должна сама отслеживать любые изменения значения переменной и посылать клиенту обновленные данные посредством исполнения оператора DDEWRITE.

Если параметр режим принимает значение DDE>manual, то переменная обновляется только один раз. Когда клиент запросил обмен данными по “горячей линии”, то Clarion-программа должна сама отслеживать любые изменения значения переменной и посылать клиенту обновленные данные посредством исполнения оператора DDEWRITE. Для установки или получения величины интервала времени для связи DDE (по умолчанию пять секунд).

Если параметр режим принимает целые положительные значения, то внутренние библиотеки по истечению указанного промежутка времени (в секундах) каждый раз проводят проверку значения переменной. Если значение изменилось, то внутренние библиотеки автоматически (дополнительные Clarion-операторы не нужны) передают клиенту обновленное значение. Следует отметить, что при этом - в зависимости от данных - возникают значительные накладные расходы, и использовать данный режим рекомендуется только по необходимости.

Если для параметра режим указано значение DDE:remove, то разрывается ранее установленная “горячая линия” связи с переменной. Если для параметра режим указано значение DDE:remove, а параметр переменная не указан, то разрываются все ранее установленные “горячие линии” связи с элементом, независимо от того, какие переменные были с ним связаны. Поэтому, для обнаружения изменения значения данных клиенту нужно снова послать серверу DDEREAD запрос.

Сообщения об ошибках:

- |                          |                          |
|--------------------------|--------------------------|
| 601 Invalid DDE Channel  | - недопустимый DDE-канал |
| 602 DDE Channel Not Open | - Не открыт DDE-канал    |

605 Time Out - Ошибка по “тайм-ауту”

Генерируемые события:

EVENT:DDErequest	Запрос клиента на элемент данных ( “холодная линия” ).
EVENT:DDEadvise	Запрос клиента на постоянное обновление элемента данных (“горячая линия”).

Пример:

```

DDERetVal  STRING(20)
WinOne     WINDOW,AT(0,0,160,400)
           ENTRY(@s20),USE(DDERetVal)
           END
MyServer    LONG
CODE
OPEN(WinOne)
MyServer = DDESERVER('MyApp', 'DataEntered') !Open as server
ACCEPT
CASE EVENT()
OF EVENT:DDErequest           !Сервер для однократно запрашиваемых данных
  DDEWRITE(MyServer,DDE:manual,'DataEntered',DDERetVal)
  !Однократное предоставление данных
OF EVENT:DDEadvise           !Сервер запроса непрерывного обновления
  DDEWRITE(MyServer,15,'DataEntered',DDERetVal)
  !Проверка на изменение каждые 15 сек
  ! и предостав-ие измененных данных
END
END

```

Смотри также: DDEQUERY, DDEREAD

## DDEEXECUTE (послать команду DDE серверу)

**DDEEXECUTE**( *канал, команда* )

<b>DDEEXECUTE</b>	Посылает командную строку в открытый канал DDE-клиента.
<i>канал</i>	Целочисленная константа типа LONG или переменная, указывающие канал клиента - то значение, которое возвращает функция DDECLIENT.
<i>команда</i>	Строковая константа или переменная, содержащие команду, предназначенную для исполнения сервер-приложением.

Процедура **DDEEXECUTE** позволяет программе DDE-клиента передавать команду серверу. Формат команды должен быть таким, что сервер сможет ее понять и исполнить. В

качестве сервера может быть и не Clarion-программа. По принятому соглашению строка-команда заключается в квадратные скобки ([ ]).

DDE-сервер Clarion-приложения может воспользоваться функцией DDEVALUE() для того, чтобы узнать какую команду послал клиент. Оператор CYCLE в конце обработки события DDE:EVENTexecute присылает клиенту положительное подтверждение приема посланной им команды.

Сообщения об ошибках:

601 Invalid DDE Channel	- недопустимый DDE-канал
602 DDE Channel Not Open	- Не открыт DDE-канал
604 DDEEXECUTE Failed	- Ошибка команды DDEEXECUTE
605 Time Out	- Ошибка по "тайм-ауту"

Генерируемые события:

EVENT:DDEcommand	Клиент прислал запрос-команду.
EVENT:DDEexecute	Клиент прислал запрос-команду.

### Пример:

!Фрагмент программы сервер-приложения:

```
WinOne WINDOW,AT(0,0,160,400)
    END
SomeServer LONG
DDEChannel LONG
CODE
OPEN(WinOne)
DDEChannel = DDECLIENT('PROGMAN','PROGMAN')
    !Открыть канал связи с Windows Program
    ! Manager
DDEEXECUTE(DDEChannel,['CreateGroup(Clarion Applications)'])
    !Создать группу для новой программы
DDEEXECUTE(DDEChannel,['ShowGroup(1)']) !Отобразить ее на экран
DDEEXECUTE(DDEChannel,['AddItem(MYAPP.EXE,My Program,PROGMAN.EXE,2)'])
    !Создать новый элемент группы
    ! используя вторую пиктограмму
    ! progman.exe
```

## DDEPOKE (послать неспрашиваемые данные DDE-серверу)

**DDEPOKE**(*канал, элемент, значение* )

<b>DDEPOKE</b>	Посылает DDE-серверу неспрашиваемые данные через открытый канал DDE-клиента.
<i>канал</i>	Целочисленная константа типа LONG или переменная, указывающие канал клиента - то значение, которое возвращает функция DDECLIENT.
<i>элемент</i>	Строковая константа или переменная, указывающие относящийся к приложению элемент, на имя которого посылаются неспрашиваемые данные.
<i>значение</i>	Строковая константа или переменная, в которых находятся данные, предназначенные для элемента.

Процедура **DDEPOKE** позволяет программе DDE-клиента пересылать неспрашиваемые данные серверу. Чтобы сервер смог распознать и обработать данные, параметры элемент и значение должны быть представлены в формате данных сервер-приложения. В качестве сервера может быть и не Clarion-программа.

DDE-сервер Clarion-приложения, чтобы выяснить что ему прислал клиент, может воспользоваться функциями DDEITEM() и DDEVALUE().

Опреатор CYCLE в конце обработки события DDE:EVENTroke присылает клиенту положительное подтверждение приема неспрашиваемых данных.

Сообщения об ошибках:

601 Invalid DDE Channel	- Недопустимый DDE-канал
602 DDE Channel Not Open	- Не открыт DDE-канал
604 DDEPOKE Failed	- Ошибка команды DDEPOKE
605 Time Out	- Ошибка по "тайм-ауту"

Генерируемые события:

EVENT:DDEpoke Клиент прислал неспрашиваемые данные.

### Пример:

```
WinOne WINDOW,AT(0,0,160,400)
    END
DDEChannel LONG
CODE
OPEN(WinOne)
DDEChannel = DDECLIENT('Excel','System')      !Открыть канал связи с Excel
DDEEXECUTE(DDEChannel,['NEW(1)'])!Создать новую таблицу
DDEEXECUTE(DDEChannel,['Save.As("DDE_CHART.XLS)'])
```

```

!Сохранить ее в файле DDE_CHART.XLS
DDECLOSE(DDEChannel) !Завершить диалог
DDEChannel = DDECLIENT('Excel', 'DDE_CHART.XLS')
!Установить канал связи с новой диаграммой
DDEPOKE(DDEChannel, 'R1C2', 'Widgets') !Переслать в нее данные
DDEPOKE(DDEChannel, 'R1C3', 'Gadgets')
DDEPOKE(DDEChannel, 'R2C1', 'East')
DDEPOKE(DDEChannel, 'R3C1', 'West')
DDEPOKE(DDEChannel, 'R2C2', '450')
DDEPOKE(DDEChannel, 'R3C2', '275')
DDEPOKE(DDEChannel, 'R2C3', '340')
DDEPOKE(DDEChannel, 'R3C3', '390')
DDEEXECUTE(DDEChannel, '[SELECT("R1C1:R3C2")]') !Подсветить переданные данные
DDEEXECUTE(DDEChannel, '[NEW(2,2)]') ! и создать новую диаграмму
!Посылаем ряд команд по форматированию диаграммы и работы и нею
DDECLOSE(DDEChannel) !В конце - завершить диалог
    
```

## DDECLOSE (завершить диалог с DDE-сервером)

### DDECLOSE( канал )

**DDECLOSE**                      Закрывает открытый канал DDE.  
*канал*                              Целочисленная константа типа LONG или переменная, указывающие номер канала - то значение, которое возвращают функции DDESERVER или DDECLIENT.

Процедура **DDECLOSE** предоставляет программе DDE-клиента возможность закрыть указанный канал. Когда закрывается окно, открывшее канал, то автоматически закрывается и сам канал.

Сообщения об ошибках:

601 Invalid DDE Channel	- Недопустимый DDE-канал
602 DDE Channel Not Open	- Не открыт DDE-канал
605 Time Out	- Ошибка по "тайм-ауту"

### Пример:

```

WinOne WINDOW, AT(0,0,160,400)
END
SomeServer LONG
CODE
OPEN(WinOne)
SomeServer = DDECLIENT('SomeApp', 'MyTopic') !Канал клиента
ACCEPT
END
DDECLOSE(SomeServer)
    
```

## **Связывание и внедрение объектов**

### **Введение в OLE**

---

Механизм связывания и внедрения объектов (Object Linking and Embedding - OLE) позволяет “объекты” из одного приложения Windows связать или включить в “документ” (структуру данных) другого приложения. Приложение, создающее и поддерживающее объект, является приложением-сервером OLE, тогда как приложение, которое содержит объект, называется OLE контроллером (его иногда еще называют OLE клиентом). Объекты внедрения или связывания - это структуры данных, присущие приложению - серверу OLE (такие как график из электронной таблицы или рисунок из приложения - графического редактора). Этот объект помещается в окно-контейнер приложения контроллера. В приложении на языке Clarion окно контейнер и является управляющим элементом типа OLE.

Реализация OLE в Clarion для Windows позволяет, написанным на нем приложениям, служить OLE контроллером, связывая или включая объекты от любого приложения - OLE сервера. Реализация в Clarion OLE поддерживает также автоматизацию OLE, которая представляет собой способность OLE контроллера динамически управлять OLE сервером, используя его макроязык.

### **Связывание объектов**

Связывание объектов в общем случае означает, что в OLE контроллере содержится “ссылка” на объект, будь то объект целой структурой данных (подобно файлу электронной таблице) или компонентом такой структуры (как, например, диапазон ячеек в той же таблице).

При связывании объекта с OLE контроллером контроллер содержит только информацию, необходимую для нахождения связанных данных. Эта информация может храниться или в поле BLOB или файле - комплексном хранилище (OLE Compound Storage file).

### **Внедрение объектов**

Внедрение объектов означает, что OLE контроллер хранит весь объект целиком, независимо от приложения-сервера OLE. Внедренный в приложение-контроллер объект не существует в виде отдельного файла, к которому могло бы обращаться приложение-сервер. Приложение - OLE контроллер полностью содержит активный объект, который хранится либо в BLOB, либо в файле - комплексном хранилище (OLE Compound Storage file).

### **Поддержка объектов OLE**

Любой OLE объект в приложении-контроллере, внедренный или связанный,



поддерживается приложением-сервером, создавшим этот объект. Это означает, что когда пользователь хочет изменить этот объект, то для того чтобы сделать эти изменения, OLE контроллер активизирует приложение-сервер. Есть два способа активизации сервера: активизация “на месте” (in place) и активизация в режиме открытия (“open-mode”).

### Активизация “на месте”

Активизация “на месте” означает, что пользователю кажется, будто бы он остается в приложении OLE контроллере, но меню и панель инструментов сервера объединяются с меню и инструментами контроллера, а текущим исполняемым приложением является приложение-сервер. Редактируемый объект имеет мерцающую рамку, чтобы обозначить то, что он находится в режиме редактирования.

Если в приложении сервере имеется одна или несколько панелей инструментов, то эта панель инструментов появляется либо в виде всплывающей панели, либо присоединенной к одному из краев рамки, либо в сочетании этих подходов. Это может приводить к “выпаданию” вниз объектов управления вашего окна, так что будьте внимательны при проектировании окна с OLE объектами.

### Активизация в режиме открытия

Активизация в режиме открытия означает, что пользователь переключается на приложение-сервер, выполняемый в отдельном окне. Редактируемый объект находится в приложении-сервере, в то время как в приложении-контроллере он мерцает, чтобы обозначить то, что объект редактируется в отдельном окне.

### Свойства контейнера объекта OLE

С контейнером объекта OLE связаны несколько свойств, относящихся только к объектам типа OLE (но не OCX).

## **Свойства-атрибуты**

---

PROP:Create	Атрибут CREATE (пробел, если нет такого атрибута) (только присвоение значения)
PROP:Open	Атрибут OPEN (пробел, если нет такого атрибута) (только присвоение значения)
PROP:Document	Атрибут DOCUMENT (пробел, если нет такого атрибута) (только присвоение значения)
PROP:Link	Атрибут LINK (пробел, если нет такого атрибута) (только присвоение значения)
PROP:Clip	Атрибут CLIP. Атрибут-переключатель. Присвоение нулевой строки (“) выключает его, любое другое значение - включает. (Только присвоение значения)
PROP:Stretch	Атрибут STRETCH. Атрибут-переключатель. Присвоение нулевой строки (“) выключает его, любое другое значение - включает. (Только

	присвоение значения)
PROP:Autosize	Атрибут AUTOSIZE. Атрибут-переключатель. Присвоение нулевой строки (") выключает его, любое другое значение - включает. (Только присвоение значения)
PROP:Zoom	Атрибут ZOOM. Атрибут-переключатель. Присвоение нулевой строки (") выключает его, любое другое значение - включает. (Только присвоение значения)
PROP:Compatibility	Атрибут COMPATIBILITY (пробел, если нет такого атрибута) (только присвоение значения)

## Необъявленные свойства

---

PROP:Blob	Преобразует объект в/из переменную BLOB. (ЧТЕНИЕ/ЗАПИСЬ)
PROP:SaveAs	Сохраняет объект в файле - комплексном хранилище. (Только ЗАПИСЬ)

Для занесения объекта в комплексное хранилище используется синтаксис 'имя\_файла\!компонент'.

Например:

```
?controlx{PROP:SaveAs} = 'myfile\!objectx'
```

PROP:DoVerb	Выполняет команду из следующего набора команд. (Только ЗАПИСЬ)
DOVERB:Primary (0)	Вызывает первичные действия объекта. Что это за действия, определяется самим объектом, а не контейнером. Если для объекта поддерживается активизация "на месте", то обычно первичное действие это и делает.
DOVERB:Show (-1)	Говорит объекту о том, что он должен открыться для просмотра и редактирования. С помощью этой команды отображается вновь созданный объект и высвечивается источник связи. Обычно эта команда является алиасом для некоего другого действия, определяемого самим объектом.
DOVERB:Open (-2)	Говорит объекту о том, что он должен открыться для просмотра и редактирования в отдельном от контейнера окне (это относится объектам, поддерживающим активизацию "на месте"). Если объект активизацию "на месте" не поддерживает, то эта команда для него эквивалентна команде DOVERB:Show.
DOVERB:Hide (-3)	Говорит объекту о том, что он должен убрать свой интерфейс

пользователя. Эта команда применима только к объектам, активизируемым “на месте”.

DOVERB:UIActivate (-4)

Активизирует объект “на месте” наряду с его полным набором инструментальных средств, включая меню, панели инструментов и его имя в строке заголовка окна-контейнера.

DOVERB:InPlaceActivate (-5)

Активизирует объект “на месте” не включая набор инструментальных средств (меню и панели инструментов), которые необходимы пользователю для изменения внешнего вида и поведения объекта.

DOVERB:DiscardUndoState (-6)

Предписывает объекту уничтожить содержимое буфера изменений (UNDO), которые объект возможно поддерживает, не деактивируя при этом сам объект.

DOVERB:Properties (-7)

Вызывает модуль просмотра системных модальных свойств объекта, чтобы позволить пользователю установить их значения.

PROP:Deactivate Деактивизирует OLE объект, активизированный “на месте”. (ЧТЕНИЕ/ЗАПИСЬ/ВЫПОЛНЕНИЕ)

PROP:Update Предписывает объекту обновить самого себя. (ЧТЕНИЕ/ЗАПИСЬ/ВЫПОЛНЕНИЕ)

PROP:CanPaste Можно ли объект помещать во внутренний буфер обмена (clipboard). (Только ЧТЕНИЕ)

PROP:Paste Помещает объект из внутреннего буфера обмена в экранный контейнер. (ЧТЕНИЕ/ЗАПИСЬ/ВЫПОЛНЕНИЕ)

PROP:CanPasteLink Можно ли объект, находящийся во внутреннем буфере обмена поместить в контейнер в виде связи? (Только ЧТЕНИЕ)

PROP:PasteLink Помещает и связывает объект, находящийся во внутреннем буфере, в OLE контейнер. (ЧТЕНИЕ/ЗАПИСЬ/ВЫПОЛНЕНИЕ)

PROP:Copy Копирует объект из OLE контейнера во внутренний буфер. (ЧТЕНИЕ/ЗАПИСЬ/ВЫПОЛНЕНИЕ)

PROP:ReportException Послать объекту OLE прерывание (Только ЗАПИСЬ).

PROP:OLE Определить находится ли в контейнере объект OCX или OLE? (Только ЧТЕНИЕ)

### Example:

```

PROGRAM
MAP
    INCLUDE('OCX.CLW')
SelectOleServer    FUNCTION(OleQ PickQ),STRING
END
    INCLUDE 'XL.CLW'    !Константы, используемые Excel
    INCLUDE 'ERRORS.CLW' !Включить коды ошибок
    
```

```

SaveLinks  FILE,DRIVER('TopSpeed'),PRE(SAV),CREATE
Object      BLOB
Record      RECORD
LinkType    STRING(1)                !F = File, B = BLOB
LinkFile    STRING(64)               ! Имя файла комплексного хранилища OLE
          END
          END
i          LONG                      !Счетчики цикла
j          LONG
ResultQ     QUEUE                    !Очередь для хранения данных из OLEDIRECTORY
Name        CSTRING(64)
CLSID       CSTRING(64)
ProgID      CSTRING(64)
          END
MainWin     WINDOW('OLE Demo'),AT(,,350,200),STATUS(-1,-
1),SYSTEM,GRAY,RESIZE,MAX,TIMER(1)
          MENUBAR
          MENU('&File')
          ITEM('e&xit'),USE(?exit)
          END
          MENU('&Objects')
          ITEM('Create Object'),USE(?CreateObject)
          ITEM('Paste Object'),USE(?PasteObject)
          ITEM('PasteLink Object'),USE(?PasteLinkObject)
          ITEM('Save Object to BLOB'),USE(?SaveObjectBlob),DISABLE
          ITEM('Save Object to OLE File'),USE(?SaveObjectFile),DISABLE
          ITEM('Retrieve Saved Object'),USE(?GetObject),DISABLE
          END
          MENU('&Activate')
          ITEM('&Spreadsheet'),USE(?ActiveExcel)
          ITEM('&Any OLE Object'),USE(?ActiveOLE),DISABLE
          END
          END
          OLE,AT(5,10,160,100),COLOR(0808000H),USE(?ExcelObject)
          MENUBAR
          MENU('&Clarion App')
          ITEM('&Deactivate Excel'),USE(?DeactExcel)
          END
          END
          END
          OLE,AT(170,10,160,100),USE(?AnyOLEObject),AUTOSIZE
          MENUBAR
          MENU('&Clarion App')
          ITEM('&Deactivate Object'),USE(?DeactOLE)
          END
          END
          END

```

```
END
CODE
OPEN(SaveLinks)
IF ERRORCODE() !Проверить, что Open без ошибок
  IF ERRORCODE() = NoFileErr ! Если файл не существует
    CREATE(SaveLinks) ! то создать его
    IF ERRORCODE() THEN HALT(,ERROR()).
    OPEN(SaveLinks) ! а затем открыть
    IF ERRORCODE() THEN HALT(,ERROR()).
  ELSE
    HALT(,ERROR())
  END
END
OPEN(MainWin)
?ExcelObject{PROP:Create} = 'Excel.Sheet.5' !Создать объект - таблицу Excel
DO BuildSheetData ! заполнить ее произвольными данными
IF RECORDS(SaveLinks) !Проверить существование сохраненной записи
  SET(SaveLinks) ! взять ее
  NEXT(SaveLinks)
  POST(EVENT:Accepted,?GetObject) ! и вывести
  DO MenuEnable
ELSE
  ADD(SaveLinks) ! или добавить пустую запись
END
IF ERRORCODE() THEN HALT(,ERROR()).
ACCEPT
CASE EVENT()
OF EVENT:CloseWindow
  ?ExcelObject{PROP:Deactivate} !Закрыть приложение - сервер
  ?AnyOLEObject{PROP:Deactivate}
OF EVENT:Timer
  IF CLIPBOARD()
    IF ?AnyOLEObject{PROP:CanPaste} !Допустима вставка из буфера обмена
      IF ?PasteObject{PROP:Disable} THEN ENABLE(?PasteObject).
      ELSIF NOT ?PasteObject{PROP:Disable}
        DISABLE(?PasteObject)
      END
      IF ?AnyOLEObject{PROP:CanPasteLink}
        !Можно вставить объект-связь из буфера?
        IF ?PasteLinkObject{PROP:Disable} THEN ENABLE(?PasteLinkObject).
        ELSIF NOT ?PasteLinkObject{PROP:Disable}
          DISABLE(?PasteLinkObject)
        END
      END
    END
  END
OF EVENT:Accepted
CASE FIELD()
OF ?Exit
```

```

    POST(EVENT:CloseWindow)
  OF ?CreateObject
    OLEDIRECTORY(ResultQ,0)!Получить список установленных OLE серверов
    ?AnyOLEObject{PROP:Create} = SelectOleServer(ResultQ)
!Дать пользователю выбрать
    ?AnyOLEObject{PROP:DoVerb} = 0
!Активизировать OLE сервер в режиме "по умолчанию"
    DO MenuEnable
  OF ?PasteObject
    ?AnyOLEObject{PROP:Paste}          !Вставить объект
    SETCLIPBOARD('Paste Completed')    !Поместить в буфер простой текст
    DO MenuEnable
  OF ?PasteLinkObject
    ?AnyOLEObject{PROP:PasteLink}      !PasteLink the object
    SETCLIPBOARD('PasteLink Completed') ! Поместить в буфер простой текст
    DO MenuEnable
  OF ?SaveObjectBlob          !Запомнить объект в поле BLOB
    SAV:Object{PROP:Handle} = ?AnyOLEObject{PROP:Blob}
    SAV:LinkType = 'B'
    PUT(SaveLinks)
    IF ERRORCODE() THEN STOP(ERROR()).
  OF ?SaveObjectFile          !Сохранить в OLE Compound Storage file
    ?AnyOLEObject{PROP:SaveAs} = 'TEST1.OLE\!Object'
    SAV:LinkFile = 'TEST1.OLE\!Object'
    SAV:LinkType = 'F'
    PUT(SaveLinks)
    IF ERRORCODE() THEN STOP(ERROR()).
  OF ?GetObject
    IF SAV:LinkType = 'F'      !Сохранить в OLE Compound Storage file?
      ?AnyOLEObject{PROP:Open} = SAV:LinkFile
    ELSEIF SAV:LinkType = 'B'  !Сохранить в BLOB?
      ?AnyOLEObject{PROP:Blob} = SAV:Object{PROP:Handle}
    END
    DISPLAY
  OF ?ActiveExcel
    ?ExcelObject{PROP:DoVerb} = 0      !Запустить Excel "на месте"
  OF ?ActiveOLE
    ?AnyOLEObject{PROP:DoVerb} = 0
!Активизировать OLE сервер в режиме "по умолчанию"
  OF ?DeactExcel
    ?ExcelObject{PROP:Deactivate}      !Вернуться в Clarion программу
  OF ?DeactOLE
    ?AnyOLEObject{PROP:Deactivate}!Вернуться в Clarion программу
  END
END
END
END

```

#### BuildSheetData ROUTINE

```
!Использовать для построения таблицы OLE автоматизацию
?ExcelObject{PROP:ReportException} = TRUE
?ExcelObject{Application.Calculation} = xlManual!выключить автопересчет
LOOP i = 1 TO 3                !Заполнить таблицу значениями
    LOOP j = 1 TO 3
        ?ExcelObject{Cells(' & i & ',' & j & ').Value} = Random(100,900)
    END
    ?ExcelObject{Cells(4,' & i & ').Value} = 'Sum'
    ?ExcelObject{Cells(5,' & i & ').FormulaR1C1} = '=SUM(R[-4]C:R[-2]C)'
    ?ExcelObject{Cells(6,' & i & ').Value} = 'Average'
    ?ExcelObject{Cells(7,' & i & ').FormulaR1C1} = '=AVERAGE(R[-6]C:R[-4]C)'
END
?ExcelObject{Application.Calculation} = xlAutomatic !включить автопересчет
DISPLAY
```

#### MenuEnable ROUTINE

!Включить пункты меню

```
ENABLE(?ActiveOLE)
ENABLE(?SaveObjectBlob,?GetObject)
```

#### SelectOleServer FUNCTION(OleQ PickQ)

```
window WINDOW('Choose OLE Server'),AT(.,122,159),CENTER,SYSTEM,GRAY
    LIST,AT(11,8,100,120),USE(?List),HVSCROLL, |
    FORMAT('146L~Name~@s64@135L~CLSID~@s64@20L~ProgID~@s64@'), |
    FROM(PickQ)
    BUTTON('Select'),AT(42,134),USE(?Select)
END
CODE
OPEN(window)
SELECT(?List,1)
ACCEPT
CASE ACCEPTED()
    OF ?Select
        GET(PickQ,CHOICE(?List))
        IF ERRORCODE() THEN STOP(ERROR()).
        POST(EVENT:CloseWindow)
    END
END
RETURN(PickQ.ProgID)
```

### Свойства, влияющие на интерфейс

#### PROP:Object

Взять изначальный интерфейс объекта (ТОЛЬКО ЧТЕНИЕ)

В Visual Basic для панели инструментов и объекта-иерархической структуры для отображения пиктограмм на кнопках инструментов и в дереве используется объект "image-list" (список изображений). Для того, чтобы связать объект - изображение с инструментальной панелью,

используется следующий оператор:

```
?toolbar{'ImageList'} = ?imagelist{prop:object}
```

PROP:SelectInterface Выбирает интерфейс, который следует использовать с данным объектом (ТОЛЬКО ПРИСВОЕНИЕ)

```
?x{PROP:SelectInterface} = 'x.y'
```

```
?x{'z(1)'} = 1
```

```
?x{'z(2)'} = 2
```

имеет то же значение, что и

```
?x{'x.y.z(1)'} = 1
```

```
?x{'x.y.z(2)'} = 2
```

PROP:AddRef Увеличить счетчик обращений для интерфейса (ТОЛЬКО ПРИСВОЕНИЕ)

PROP:Release Уменьшить счетчик обращений для интерфейса (ТОЛЬКО ПРИСВОЕНИЕ)

## OLEDIRECTORY (получить список установленных OLE/OCX)

**OLEDIRECTORY**( *список* , *флаг* )

**OLEDIRECTORY** Получить список установленных OLE серверов или объектов OCX.  
*список* Метка структуры QUEUE, в которую заносится список.  
*флаг* Целочисленная константа или переменная, которая определяет, получить ли список OLE серверов (флаг = 0) или список объектов OCX.

С помощью OLEDIRECTORY получается список всех установленных OLE серверов или OCX объектов и заносится в очередь список.

Очередь список должна иметь точно такую же структуру, что и очередь OleQ, объявленная в EQUATES.CLW:

```
OleQ QUEUE,TYPE
```

```
Name CSTRING(64) !Имя приложения - OLE сервера
```

```
CLSID CSTRING(64) !Уникальный идентификатор для операционной системы
```

```
ProgID CSTRING(64) !Регистрационное имя, как напр.: Excel.Sheet.5
```

```
END
```

### Пример:

```
ResultQ QUEUE(OleQ) !Объявить ResultQ такую же как OleQ QUEUE in EQUATES.CLW
```

```
END
```

```
CODE
```

```
OLEDIRECTORY(ResultQ,0) !Получить список установленных OLE серверов  
! поместить его в ResultQ
```



?OleControl{PROP:Create} = SelectOleServer(ResultQ) ! и дать пользователю  
выбрать

SelectOleServer FUNCTION(OleQ PickQ) !Функция выбора пользователем OLE сервера  
window WINDOW('Choose OLE Server'),AT(,,122,159),CENTER,SYSTEM,GRAY  
LIST,AT(11,8,100,120),USE(?List),HVSCROLL, |

FORMAT('146L~Name~@s64@135L~CLSID~@s64@20L~ProgID~@s64@'),FROM(PickQ)  
BUTTON('Select'),AT(42,134),USE(?Select)  
END  
CODE  
OPEN(window)  
SELECT(?List,1)  
ACCEPT  
CASE ACCEPTED()  
OF ?Select  
GET(PickQ,CHOICE(?List))  
IF ERRORCODE() THEN STOP(ERROR()).  
POST(EVENT:CloseWindow)  
END  
END  
RETURN(PickQ.ProgID)

## Пользовательские объекты OLE (.OCX)

### Введение

Пользовательские OLE объекты обычно имеют расширение .OCX. Поэтому обычно их называют объектами .OCX. Эти объекты подобны объектам .VBX в том смысле, что они самодостаточны и, будучи использованным в программе, выполняют определенную функцию. Однако, объекты .OCX не имеют ограничений присущих . VBX, поскольку построены по спецификациям Microsoft OLE 2, которые разработаны с учетом межязыковой совместимости (для языков, отличных от Visual Basic).

### Свойства объектов .OCX

PROP:Create	Атрибут CREATE (если атрибут отсутствует, то принимается равным пробелу). (ТОЛЬКО ПРИСВОЕНИЕ)
PROP:DesignMode	Имеет ли объект контейнер для режима редактирования (имеется ли ограничительная рамка вокруг него)? (ТОЛЬКО ПРИСВОЕНИЕ)
PROP:Ctrl	Это объект .OCX? (ТОЛЬКО ЧТЕНИЕ)
PROP:GrabHandles	Высвечивает для объекта “зацепки” для изменения размеров. (ТОЛЬКО ПРИСВОЕНИЕ)
PROP:OLE	Находится ли в контейнере объект OCX или OLE? (ТОЛЬКО ЧТЕНИЕ)
PROP:IsRadio	Это OCX - кнопка радио (с зависимой фиксацией)? (ТОЛЬКО ЧТЕНИЕ)
PROP:LastEventName	Получить имя последнего события присланного объекту .OCX. (ТОЛЬКО ЧТЕНИЕ)
PROP:SaveAs	Сохранить объект в файле - комплексном хранилище (OLE Compound Storage file). (ТОЛЬКО ПРИСВОЕНИЕ) Синтаксис оператора для помещения объекта в файл - хранилище таков: ‘имя_файла\!компонент’ Например: ?controlx{PROP:SaveAs} = ‘myfile\!objectx’
PROP:DoVerb	Выполняет команду из следующего набора команд. (Только ЗАПИСЬ) DOVERB:Primary (0) Вызывает первичные действия объекта. Что это за действия, определяется самим объектом, а не контейнером. Если для объекта поддерживается активизация “на месте”, то обычно первичное действие это и делает. DOVERB:Show (-1) Говорит объекту о том, что он должен открыться для просмотра или редактирования. С помощью этой команды отображается вновь созданный объект и высвечивается источник связи. Обычно эта команда является

алиасом для некоего другого действия, определяемого самим объектом.

**DOVERB:Open (-2)**

Говорит объекту о том, что он должен открыться для просмотра и редактирования в отдельном от контейнера окне (это относится к объектам, поддерживающим активизацию “на месте”). Если объект активизацию “на месте” не поддерживает, то эта команда для него эквивалентна команде DOVERB:Show.

**DOVERB:Hide (-3)**

Говорит объекту о том, что он должен убрать свой интерфейс пользователя. Эта команда применима только к объектам, активизируемым “на месте”.

**DOVERB:UIActivate (-4)**

Активизирует объект “на месте” наряду с его полным набором инструментальных средств, включая меню, панели инструментов и его имя в строке заголовка окна-контейнера.

**DOVERB:InPlaceActivate (-5)**

Активизирует объект “на месте” не включая набор инструментальных средств (меню и панели инструментов), которые необходимы пользователю для изменения внешнего вида и поведения объекта.

**DOVERB:DiscardUndoState (-6)**

Предписывает объекту уничтожить содержимое буфера изменений (UNDO), которые объект возможно поддерживает, не деактивируя при этом сам объект.

**DOVERB:Properties (-7)**

Вызывает модуль просмотра системных модальных свойств объекта, чтобы позволить пользователю установить их значения.

## Оконные функции

---

Оконные функции являются стандартным средством программирования под Windows во многих языках программирования. Оконная функция представляет собой процедуру или функцию, которую вы (программист) пишете для обработки особых ситуаций, относительно которых операционная система полагает, что программисту возможно необходимо отреагировать на нее. Оконная функция вызывается операционной системой всякий раз, когда нужно обработать такую ситуацию. Поэтому, оконная функция не выглядит частью логической последовательности выполнения программы, а кажется обособленной и “магической”, не имеющей логической связи с другими процедурами и функциями программы.

Язык Clarion для Windows не требует от вас писать собственные оконные функции для выполнения наиболее общих действий, как это имеет место в других языках программирования, поскольку эти действия отслеживаются функциями библиотеки

времени выполнения и циклом АСCEPT, Однако, поскольку объекты ОСХ обычно написаны на других языках, в которых требуется написание своих оконных функций, вам понадобится их написать для обработки событий и других программных аспектов связанных с работой объектов ОСХ, использованных в программе на языке Clarion.

Существует три вида оконных функций, которые вам возможно придется написать: обработчик событий, контроллер редактирования свойств и обработчик изменения свойств. Вы можете называть их как угодно, но они имеют специфические требования к передаваемым им параметрам.

## **Оконная функция - обработчик событий ОСХ**

---

Прототип этой функции должен выглядеть так:

OcxEventFuncName      FUNCTION(\*SHORT,SIGNED,LONG),LONG

От операционной системы передаются следующие параметры:

*SHORT	Параметр-указатель, который передается другим библиотечным функциям объекта ОСХ: OCXGETPARAM, OCXGETPARAMCOUNT, и OCXSETPARAM в качестве первого параметра.
SIGNED	Номер поля, присвоенный объекту. Это тот самый номер, который представляется мнемонической меткой соответствия данного объекта.
LONG	Номер события, относящегося к ОСХ. Мнемонические имена для некоторых заранее определенных номеров событий содержатся в файле OCXEVENT.CLW.

Возвращаемое значение типа LONG сообщает операционной системе необходима ли некая дальнейшая обработка. Возврат нуля (0) означает необходимость дополнительных действий (таких как изменение значения USE-переменной или выключение радио кнопки), тогда как возврат любого другого значения говорит о том, что обработка события завершена.

Обработка событий, генерируемых объектом .ОСХ, должна выполняться быстро, поскольку некоторые события критичны ко времени. Поэтому у пользователя не должно быть возможности помешать выполнению этой функции (т.е. нельзя использовать функцию MESSAGE или оператор ASK или другие операторы касающиеся обработки данного окна). Программный код функции - обработчика события должен выполнять только необходимые действия и как можно быстрее (обычно это подразумевает исключение обработки событий, связанных с мышью).

## **Функция - контроллер редактирования свойств ОСХ**

Прототип этой функции должен выглядеть следующим образом:

OcxPropEditFuncName FUNCTION(SIGNED,STRING),LONG

От операционной системы передаются следующие параметры:

SIGNED Номер поля, присвоенный объекту. Это тот самый номер, который представляется мнемонической меткой соответствия данного объекта.

STRING Название свойства, значение которого должно изменяться.

Возвращаемое значение типа LONG сообщает операционной системе допустимо ли редактирование значения свойства. Если возвращаемое значение равно (0), то функция не разрешает изменение значения свойства и пользователь не может этого делать. Если функция возвращает любое другое значение, то это означает возможность изменения пользователем данного свойства.

### **Функция - обработчик изменения свойств OCX**

Прототип этой функции должен выглядеть следующим образом:

OcxPropChangeProcName PROCEDURE(SIGNED,STRING)

От операционной системы передаются следующие параметры:

SIGNED Номер поля, присвоенный объекту. Это тот самый номер, который представляется мнемонической меткой соответствия данного объекта.

STRING Название изменяемого свойства.

Эта процедура вызывается, при изменении значения свойства.

### **Пример:**

! В этой программе используется OCX - календарь,  
! который Microsoft поставляет вместе с Access95  
! (в составе MS Office Professional для Windows 95).

```

PROGRAM
MAP
    INCLUDE('OCX.C LW')
EventFunc    FUNCTION(*SHORT Reference,SIGNED OleControl,LONG CurrentEvent),LONG
PropChange   PROCEDURE(SIGNED OleControl,STRING
CurrentProp)
PropEdit     FUNCTION(SIGNED OleControl,STRING CurrentProp),LONG
END
    INCLUDE('OCXEVENT.C LW') !Константы, используемые событиями OCX
    INCLUDE('ERRORS.C LW')   ! Константы кодов ошибок

GlobalQue    QUEUE
F1           STRING(255)
END
    
```

```

SaveDate  FILE,DRIVER('TopSpeed'),PRE(SAV),CREATE
Record    RECORD
DateField STRING(10)
          END
          END

MainWin   WINDOW('OCX Demo'),AT(,350,200),STATUS(-1,-1),SYSTEM,GRAY,MAX,RESIZE
          MENUBAR
          MENU('&File')
            ITEM('Save Date to File'),USE(?SaveObjectValue)
            ITEM('Retrieve Saved Date'),USE(?GetObject)
            ITEM('E&xit'),USE(?exit)
          END
          MENU('&Object')
            ITEM('About Box'),USE(?AboutObject)
            ITEM('Set Date to TODAY'),USE(?SetObjectValueToday)
            ITEM('Set Date to 1st of Month'),USE(?SetObjectValueFirst)
          END
          ITEM('&Properties!'),USE(?ActiveObj)
          END
          LIST,AT(237,6,100,100),USE(?List1),HVSCROLL,FROM(GlobalQue)
          OLE,AT(5,10,200,150),USE(?OcxObject)
          END
          END
          CODE
          OPEN(SaveDate)
          IF ERRORCODE() !Проверить успешность открытия
            IF ERRORCODE() = NoFileErr ! если файл не существует
              CREATE(SaveDate) ! создать его
              IF ERRORCODE() THEN HALT(,ERROR()).
              OPEN(SaveDate) ! и открыть
              IF ERRORCODE() THEN HALT(,ERROR()).
            ELSE
              HALT(,ERROR())
            END
          END
          END
          OPEN(MainWin)
          ?OcxObject{PROP:Create} = 'MSACAL.MSACALCtrl.7'
          ! Объект OCX MS Access 95 календарь
          IF RECORDS(SaveDate) !Проверить существование сохраненной записи
            SET(SaveDate) ! и взять ее
            NEXT(SaveDate)
            IF ERRORCODE() THEN STOP(ERROR()).
            POST(EVENT:Accepted,?GetObject)
          ELSE
            ADD(SaveDate) ! иначе добавить запись
            IF ERRORCODE() THEN STOP(ERROR()).

```

```

END
IF ?OcxObject{PROP:OLE}      !Это объект OLE ?
    GlobalQue  = 'An Object is in the OLE control'
    ADD(GlobalQue)
    IF ?OcxObject{PROP:Ctrl}   !НЕ является ли объект OCX
        GlobalQue  = 'It is an OCX Object'
        ADD(GlobalQue)
    END
END
DISPLAY
OCXREGISTEREVENTPROC(?OcxObject,EventFunc)
!Зарегистрировать функцию - обработчик событий
OCXREGISTERPROPCHANGE(?OcxObject,PropChange)
! Зарегистрировать функцию - контроллер
OCXREGISTERPROPEEDIT(?OcxObject,PropEdit)
! Зарегистрировать функцию - редактор свойств
?OcxObject{PROP:ReportException} = 1
!Включить возможность сообщений об ошибках в OCX
ACCEPT
CASE EVENT()
OF EVENT:Accepted
    CASE FIELD()
    OF ?Exit
        POST(EVENT:CloseWindow)
    OF ?AboutObject
        ?OcxObject{'AboutBox'}    !вывести окно About
    OF ?SetObjectValueToday
        ?OcxObject{'Value'} = FORMAT(TODAY(),@D1) !Установить сегодняшнюю дату
    OF ?SetObjectValueFirst
        ?OcxObject{'Value'} = MONTH(TODAY()) & '/1/' & SUB(YEAR(TODAY()),3,2)
    OF ?SaveObjectValue          !Сохранить значение в файле
        SAV:DateField = ?OcxObject{'Value'}
        PUT(SaveDate)
        IF ERRORCODE() THEN STOP(ERROR()).
    OF ?GetObject                !Взять значение из файла
        ?OcxObject{'Value'} = SAV:DateField
    OF ?ActiveObj
        ?OcxObject{PROP:DoVerb} = 0 !Открыть диалог свойств объекта
    END
END
END
END

```

```

EventFunc  FUNCTION(*SHORT Reference,SIGNED OleControl,LONG CurrentEvent)
Count      LONG                      !Функция - обработчик событий
Res        CSTRING(200)
Parm       CSTRING(30)
CODE

```





## Библиотечные функции OCX

### OCXREGISTERPROEDIT (установить контроллер свойств)

**OCXREGISTERPROEDIT**( *объект, функция* )

**OCXREGISTERPROEDIT** Установить функцию - контроллер редактирования свойств.

*объект* Целочисленное выражение, содержащее номер поля или мнемоническая метка соответствия объекта к которому относится функция.

*функция* Имя функции редактирования свойств объекта.

Функция **OCXREGISTERPROEDIT** устанавливает функцию - контроллер редактирования свойств объекта. Она управляет редактированием свойств, запрещая или разрешая редактирование.

**Пример:**

OCXREGISTERPROEDIT(?OleControl,CallbackFunc)

### OCXREGISTERPROPCHANGE (установить функцию - редактор)

**OCXREGISTERPROPCHANGE**( *объект, процедура* )

**OCXREGISTERPROPCHANGE** Устанавливает процедуру - редактор свойств объекта.

*объект* Целочисленное выражение, содержащее номер поля или мнемоническая метка соответствия объекта к которому относится действие.

*процедура* Имя процедуры изменения свойств объекта.

Функция **OCXREGISTERPROPCHANGE** устанавливает процедуру - редактор свойств объекта. Эта процедура вызывается, когда происходит изменение свойств объекта.

**Пример:**

OCXREGISTERPROPCHANGE(?OleControl,CallbackProc)

### OCXREGISTEREVENTPROC (установить процедуру обработки событий)

**OCXREGISTEREVENTPROC**( *объект, процедура* )

**OCXREGISTEREVENTPROC** Устанавливает для объекта OCX процедуру обработки событий.

*объект* Целочисленное выражение, содержащее номер поля или мнемоническая метка соответствия объекта к которому относится действие.

*процедура*      Имя процедуры - обработчика событий для объекта.

Функция **OSXREGISTEREVENTPROC** устанавливает для объекта OSX процедуру обработки событий. К этой процедуре происходит обращение при передаче операционной системой любого события относящегося к данному объекту.

**Пример:**

OSXREGISTEREVENTPROC(?OleControl, CallbackProc)

### **OSXUNREGISTERPROPEdit (отменить функцию - контроллер свойств)**

**OSXUNREGISTERPROPEdit( объект )**

**OSXUNREGISTERPROPEdit**      Деинсталлирует функцию - контроллер свойств.

*объект*      Целочисленное выражение, содержащее номер поля или мнемоническая метка соответствия объекта к которому относится действие.

Функция **OSXUNREGISTERPROPEdit** деинсталлирует функцию - контроллер свойств.

**Пример:**

OSXUNREGISTERPROPEdit(?OleControl)

### **OSXUNREGISTERPROPCHANGE (отменить редактор свойств)**

**OSXUNREGISTERPROPCHANGE( объект )**

**OSXUNREGISTERPROPCHANGE**      Деинсталлирует функцию - редактор свойств.

*объект*      Целочисленное выражение, содержащее номер поля или мнемоническая метка соответствия объекта к которому относится действие.

Функция **OSXUNREGISTERPROPCHANGE** отменяет процедуру - редактор свойств объекта. Эта процедура вызывается, когда происходит изменение свойств объекта.

**Пример:**

OSXUNREGISTERPROPCHANGE(?OleControl)

## OCXUNREGISTEREVENTPROC (отменить процедуру обработки событий)

**OCXUNREGISTEREVENTPROC**( *объект* )

**OCXUNREGISTEREVENTPROC** Деинсталлирует функцию - обработчик события.  
*объект* Целочисленное выражение, содержащее номер поля или  
 мнемоническая метка соответствия объекта к которому относится действие.

**OCXUNREGISTEREVENTPROC** отменяет процедуру обработки событий для объекта OCX.

**Пример:**

OCXUNREGISTEREVENTPROC(?OleControl)

## OCXGETPARAMCOUNT (получить число параметров для события)

**OCXGETPARAMCOUNT**( *указатель* )

**OCXGETPARAMCOUNT** Возвращает число параметров, связанных с текущим событием для объекта OCX.

*указатель* Метка первого параметра процедуры обработки событий.

Функция **OCXGETPARAMCOUNT** возвращает число параметров, связанных с текущим событием для объекта OCX. Использование этой функции допустимо, только когда установлена функция - обработчик событий для объекта OCX.

Тип возвращаемого значения: USHORT

**Пример:**

```
OEvent    FUNCTION(Reference,OleControl,CurrentEvent) !Процедура обработки событий
Count     LONG
Res       CSTRING(200)
Parm      CSTRING(30)
          CODE
          IF CurrentEvent <> OCXEVENT:MouseMove
          !Отключить события по перемещению мыши
          Res = 'Control ' & OleControl & ' Event ' & OleControl{PROP:LastEventName} & ':'
          LOOP Count = 1 TO OCXGETPARAMCOUNT(Reference) !Цикл по параметрам
          Parm = OCXGETPARAM(Reference,Count) ! получим имя каждого параметра
          Res = CLIP(Res) & ' ' & Parm      ! и соединим их
          END
          GlobalQue = Res      !Поместим в глобальную очередь
          ADD(GlobalQue)
```

```
END
RETURN(True)
```

## OCXGETPARAM (получить строку параметра)

**OCXGETPARAM**( *указатель ,номер* )

**OCXGETPARAM** Возвращает значение параметра, связанного с текущим событием, относящимся к объекту OCX.

*указатель* Метка первого параметра процедуры - обработчика событий.

*номер* Номер выбираемого параметра.

Функция **OCXGETPARAM** возвращает значение параметра с указанным номером для текущего события, связанного с объектом OCX. Использование этой функции допустимо, только когда установлена функция - обработчик событий для объекта OCX.

Тип возвращаемого значения:     STRING

### Пример:

```
OEvent  FUNCTION(Reference,OleControl,CurrentEvent) !Процедура обработки событий
Count   LONG
Res     CSTRING(200)
Parm    CSTRING(30)
CODE
IF CurrentEvent <> OCXEVENT:MouseMove
!Отключить события по перемещению мыши
  Res = 'Control ' & OleControl & ' Event ' & OleControl{PROP:LastEventName} & ':'
  LOOP Count = 1 TO OCXGETPARAMCOUNT(Reference)
!Цикл по всем параметрам
    Parm = OCXGETPARAM(Reference,Count)      ! получение имени параметра
    Res = CLIP(Res) & ' ' & Parm! и соединение их
  END
  GlobalQue = Res          !Положить в буфер глобальной очереди
  ADD(GlobalQue)          ! и добавить элемент в очередь
END                       ! всех событий OCX и их параметров
RETURN(True)
```

## OCXSETPARAM (установить строку параметра)

**OCXSETPARAM**( *указатель* ,номер ,значение )

**OCXSETPARAM** Устанавливает значение параметра, относящегося к текущему событию, связанному с OCX.

*указатель* Метка первого параметра процедуры - обработчика событий для OCX.

*номер* Номер параметра, значение которого должно быть установлено.

*значение* Строковая константа или переменная, содержащая устанавливаемое значение.

**OCXSETPARAM** присваивает для текущего события значение параметру, указанному номером . Это допустимо только для параметров, которые передаются “по адресу” (относительно устанавливаемых параметров смотрите руководство по конкретному объекту OCX. Если модификация параметра недопустима, то она игнорируется. Использование этой функции допустимо, только когда установлена функция - обработчик событий для объекта OCX.

### Пример:

```
OEvent  FUNCTION(Reference,OleControl,CurrentEvent) ! Процедура обработки событий
Count   LONG
Res     CSTRING(200)
Parm    CSTRING(30)
CODE
IF CurrentEvent <> OCXEVENT:MouseMove
! Отключить события по перемещению мыши
Res = 'Control ' & OleControl & ' Event ' & OleControl{PROP:LastEventName} & ':'
LOOP Count = 1 TO OCXGETPARAMCOUNT(Reference)!Цикл по всем параметрам
    Parm = OCXGETPARAM(Reference,Count)      ! взять имя каждого параметра
    Res = CLIP(Res) & ' ' & Parm! и соединить их вместе
    OCXSETPARAM(Reference,1,'1')             !изменить значение параметра
END
GlobalQue = Res                !Занести строку в буфер
ADD(GlobalQue)! и добавить ее в очередь
END                             ! всех ссобытий OCX и их
RETURN(True)                   ! параметров
```

**OCXLOADIMAGE (получить графический объект)****OCXLOADIMAGE( *имя* )****OCXLOADIMAGE** Возвращает графический объект.*имя* Строковое выражение, содержащее имя файла или ресурса, который следует загрузить.

Функция **OCXLOADIMAGE** возвращает графический объект. Этот объект можно присвоить любому экранному объекту, который использует графический объект (например, “VB imagelist”)ю

Тип возвращаемого значения:    **STRING****Пример:**

```
?imagelist{‘ListImages.Add(,’ & OCXLOADIMAGE(‘CLOCK.BMP’) & ‘)’}  
!добавить изображение в объект ImageList
```

## Приложение В Коды клавиш

### Коды клавиш в Clarion.

#### Формат кода клавиши в Windows

---

Каждой клавише клавиатуры назначен код клавиши. Код клавиши представляет собой 16-битовое величину, в которой младшие 8 бит (значения от 0 до 255) представляют код нажатой клавиши, а старшие 8 бит показывают состояние клавиш Shift, Ctrl и Alt. Код нажатой клавиши можно получить посредством функций KEYCODE() или KEYBOARD(). Код клавиши имеет следующий формат:

	A	C	S	Код	
Биты :	10	9	8	7	0

Код - код нажатой клавиши

A - Бит клавиши Alt

C - Бит клавиши Ctrl

S - Бит клавиши Shift

В большинстве случаев не нужно вычислять числовое значение кода клавиши, поскольку большинство возможных комбинаций представлено мнемоническими именами в операторах EQUATE (включите в программу оператором INCLUDE файл KEYCODES.CLW и используйте вместо числовых кодов клавиш мнемонические имена). Содержимое файла KEYCODES.CLW приведено далее.

#### KEYCODES.CLW

---

Метки соответствия кодов клавиш присваивают каждому Clarion-коду клавиши мнемоническую метку. Файл меток соответствия (KEYCODES.CLW) представляет собой файл исходного текста на языке Clarion, который содержит операторы EQUATE для каждого кода клавиши и комбинации клавиш. Этот файл располагается в каталоге, в котором установлен Clarion Database Developer. В исходный текст программы его можно вставить директивой компилятора: INCLUDE('KEYCODES.CLW')

Этот файл содержит в алфавитном порядке операторы EQUATE для всех кодов клавиш:

Key0	EQUATE(0030H)	!Клавиша 0
Key1	EQUATE(0031H)	!Клавиша 1
Key2	EQUATE(0032H)	!Клавиша 2
Key3	EQUATE(0033H)	!Клавиша 3
Key4	EQUATE(0034H)	!Клавиша 4
Key5	EQUATE(0035H)	!Клавиша 5

Key6	EQUATE(0036H)	!Клавиша 6
Key7	EQUATE(0037H)	!Клавиша 7
Key8	EQUATE(0038H)	!Клавиша 8
Key9	EQUATE(0039H)	!Клавиша 9
AKey	EQUATE(0041H)	!Клавиша A
BKey	EQUATE(0042H)	!Клавиша B
CKey	EQUATE(0043H)	!Клавиша C
DKey	EQUATE(0044H)	!Клавиша D
EKey	EQUATE(0045H)	!Клавиша E
FKey	EQUATE(0046H)	!Клавиша F
GKey	EQUATE(0047H)	!Клавиша G
HKey	EQUATE(0048H)	!Клавиша H
IKey	EQUATE(0049H)	!Клавиша I
JKey	EQUATE(004AH)	!Клавиша J
KKey	EQUATE(004BH)	!Клавиша K
LKey	EQUATE(004CH)	!Клавиша L
MKey	EQUATE(004DH)	!Клавиша M
NKey	EQUATE(004EH)	!Клавиша N
OKey	EQUATE(004FH)	!Клавиша O
PKey	EQUATE(0050H)	!Клавиша P
QKey	EQUATE(0051H)	!Клавиша Q
RKey	EQUATE(0052H)	!Клавиша R
SKey	EQUATE(0053H)	!Клавиша S
TKey	EQUATE(0054H)	!Клавиша T
UKey	EQUATE(0055H)	!Клавиша U
VKey	EQUATE(0056H)	!Клавиша V
WKey	EQUATE(0057H)	!Клавиша W
XKey	EQUATE(0058H)	!Клавиша X
YKey	EQUATE(0059H)	!Клавиша Y
ZKey	EQUATE(005AH)	!Клавиша Z
F1Key	EQUATE(0070H)	!Клавиша F1
F2Key	EQUATE(0071H)	!Клавиша F2
F3Key	EQUATE(0072H)	!Клавиша F3
F4Key	EQUATE(0073H)	!Клавиша F4
F5Key	EQUATE(0074H)	!Клавиша F5
F6Key	EQUATE(0075H)	!Клавиша F6
F7Key	EQUATE(0076H)	!Клавиша F7
F8Key	EQUATE(0077H)	!Клавиша F8
F9Key	EQUATE(0078H)	!Клавиша F9
F10Key	EQUATE(0079H)	!Клавиша F10
F11Key	EQUATE(007AH)	!Клавиша F11
F12Key	EQUATE(007BH)	!Клавиша F12
AstKey	EQUATE(006AH)	!Клавиша * (звездочка)
BSKey	EQUATE(0008H)	!Клавиша “забой” (Backspace)



CapsLockKey	EQUATE(0014H)	!Клавиша CapsLock
DecimalKey	EQUATE(0006EH)	!Клавиша “десятичная точка”
DeleteKey	EQUATE(0002EH)	!Клавиша Delete
DownKey	EQUATE(00028H)	!Клавиша “Стрелка вниз”
EndKey	EQUATE(00023H)	!Клавиша End
EnterKey	EQUATE(0000DH)	!Клавиша Enter
EscKey	EQUATE(0001BH)	!Клавиша Esc
HomeKey	EQUATE(00024H)	!Клавиша Home
InsertKey	EQUATE(0002DH)	!Клавиша Insert
LeftKey	EQUATE(00025H)	!Клавиша “Стрелка влево”
MinusKey	EQUATE(0006DH)	!Клавиша “Минус”
PauseKey	EQUATE(00013H)	!Клавиша Pause
PgDnKey	EQUATE(00022H)	!Клавиша PgDn
PgUpKey	EQUATE(00021H)	!Клавиша PgUp
PlusKey	EQUATE(0006BH)	!Клавиша Plus
PrintKey	EQUATE(0002CH)	!Клавиша PrintScreen
RightKey	EQUATE(00027H)	!Клавиша “Стрелка вправо”
SlashKey	EQUATE(0006FH)	!Клавиша “Слэш” (Slash)
SpaceKey	EQUATE(00020H)	!Клавиша “Пробел”
TabKey	EQUATE(00009H)	!Клавиша Tab
UpKey	EQUATE(00026H)	!Клавиша “Стрелка вверх”
KeyPad0	EQUATE(00060H)	!Клавиша 0 на цифровой клавиатуре
KeyPad1	EQUATE(00061H)	!Клавиша 1 на цифровой клавиатуре
KeyPad2	EQUATE(00062H)	!Клавиша 2 на цифровой клавиатуре
KeyPad3	EQUATE(00063H)	!Клавиша 3 на цифровой клавиатуре
KeyPad4	EQUATE(00064H)	!Клавиша 4 на цифровой клавиатуре
KeyPad5	EQUATE(00065H)	!Клавиша 5 на цифровой клавиатуре
KeyPad6	EQUATE(00066H)	!Клавиша 6 на цифровой клавиатуре
KeyPad7	EQUATE(00067H)	!Клавиша 7 на цифровой клавиатуре
KeyPad8	EQUATE(00068H)	!Клавиша 8 на цифровой клавиатуре
KeyPad9	EQUATE(00069H)	!Клавиша 9 на цифровой клавиатуре
MouseLeft	EQUATE(00001H)	!Левая кнопка мыши
MouseRight	EQUATE(00002H)	!Правая кнопка мыши
MouseCenter	EQUATE(00003H)	!Средняя кнопка мыши
MouseLeft2	EQUATE(00005H)	!Двойной щелчок левой кнопкой мыши
MouseRight2	EQUATE(00006H)	!Двойной щелчок правой кнопкой мыши
MouseCenter2	EQUATE(00007H)	!Двойной щелчок средней кнопкой мыши
Alt0	EQUATE(0430H)	!Комбинация Alt-0
Alt1	EQUATE(0431H)	!Комбинация Alt-1
Alt2	EQUATE(0432H)	!Комбинация Alt-2
Alt3	EQUATE(0433H)	!Комбинация Alt-3
Alt4	EQUATE(0434H)	!Комбинация Alt-4
Alt5	EQUATE(0435H)	!Комбинация Alt-5
Alt6	EQUATE(0436H)	!Комбинация Alt-6

Alt7	EQUATE(0437H)	!Комбинация Alt-7
Alt8	EQUATE(0438H)	!Комбинация Alt-8
Alt9	EQUATE(0439H)	!Комбинация Alt-9
AltA	EQUATE(0441H)	!Комбинация Alt-A
AltB	EQUATE(0442H)	!Комбинация Alt-B
AltC	EQUATE(0443H)	!Комбинация Alt-C
AltD	EQUATE(0444H)	!Комбинация Alt-D
AltE	EQUATE(0445H)	!Комбинация Alt-E
AltF	EQUATE(0446H)	!Комбинация Alt-F
AltG	EQUATE(0447H)	!Комбинация Alt-G
AltH	EQUATE(0448H)	!Комбинация Alt-H
AltI	EQUATE(0449H)	!Комбинация Alt-I
AltJ	EQUATE(044AH)	!Комбинация Alt-J
AltK	EQUATE(044BH)	!Комбинация Alt-K
AltL	EQUATE(044CH)	!Комбинация Alt-L
AltM	EQUATE(044DH)	!Комбинация Alt-M
AltN	EQUATE(044EH)	!Комбинация Alt-N
AltO	EQUATE(044FH)	!Комбинация Alt-O
AltP	EQUATE(0450H)	!Комбинация Alt-P
AltQ	EQUATE(0451H)	!Комбинация Alt-Q
AltR	EQUATE(0452H)	!Комбинация Alt-R
AltS	EQUATE(0453H)	!Комбинация Alt-S
AltT	EQUATE(0454H)	!Комбинация Alt-T
AltU	EQUATE(0455H)	!Комбинация Alt-U
AltV	EQUATE(0456H)	!Комбинация Alt-V
AltW	EQUATE(0457H)	!Комбинация Alt-W
AltX	EQUATE(0458H)	!Комбинация Alt-X
AltY	EQUATE(0459H)	!Комбинация Alt-Y
AltZ	EQUATE(045AH)	!Комбинация Alt-Z
AltF1	EQUATE(0470H)	!Комбинация Alt-F1
AltF2	EQUATE(0471H)	!Комбинация Alt-F2
AltF3	EQUATE(0472H)	!Комбинация Alt-F3
AltF4	EQUATE(0473H)	!Комбинация Alt-F4
AltF5	EQUATE(0474H)	!Комбинация Alt-F5
AltF6	EQUATE(0475H)	!Комбинация Alt-F6
AltF7	EQUATE(0476H)	!Комбинация Alt-F7
AltF8	EQUATE(0477H)	!Комбинация Alt-F8
AltF9	EQUATE(0478H)	!Комбинация Alt-F9
AltF10	EQUATE(0479H)	!Комбинация Alt-F10
AltF11	EQUATE(047AH)	!Комбинация Alt-F11
AltF12	EQUATE(047BH)	!Комбинация Alt-F12
AltAst	EQUATE(046AH)	!Комбинация Alt-звездочка
AltBS	EQUATE(0408H)	!Комбинация Alt-забой
AltDecimal	EQUATE(046EH)	!Комбинация Alt-десятичная точка

AltDelete	EQUATE(042EH)	!Комбинация Alt-Delete
AltDown	EQUATE(0428H)	!Комбинация Alt-стрелка вниз
AltEnd	EQUATE(0423H)	!Комбинация Alt-End
AltEnter	EQUATE(040DH)	!Комбинация Alt-Enter
AltEsc	EQUATE(041BH)	!Комбинация Alt-Esc
AltHome	EQUATE(0424H)	!Комбинация Alt-Home
AltInsert	EQUATE(042DH)	!Комбинация Alt-Insert
AltLeft	EQUATE(0425H)	!Комбинация Alt-стрелка влево
AltMinus	EQUATE(046DH)	!Комбинация Alt-минус
AltPause	EQUATE(0413H)	!Комбинация Alt-Pause
AltPgDn	EQUATE(0422H)	!Комбинация Alt-PgDn
AltPgUp	EQUATE(0421H)	!Комбинация Alt-PgUp
AltPlus	EQUATE(046BH)	!Комбинация Alt-плюс
AltPrint	EQUATE(042CH)	!Комбинация Alt-PrintScreen
AltRight	EQUATE(0427H)	!Комбинация Alt-Стрелка вправо
AltSlash	EQUATE(046FH)	!Комбинация Alt-Слэш
AltSpace	EQUATE(0420H)	!Комбинация Alt-Пробел
AltTab	EQUATE(0409H)	!Комбинация Alt-Tab
AltUp	EQUATE(0426H)	!Комбинация Alt-Стрелка вверх
AltPad0	EQUATE(0460H)	!Комбинация Alt-0 на цифровой клавиатуре
AltPad1	EQUATE(0461H)	!Комбинация Alt-1 на цифровой клавиатуре
AltPad2	EQUATE(0462H)	!Комбинация Alt-2 на цифровой клавиатуре
AltPad3	EQUATE(0463H)	!Комбинация Alt-3 на цифровой клавиатуре
AltPad4	EQUATE(0464H)	!Комбинация Alt-4 на цифровой клавиатуре
AltPad5	EQUATE(0465H)	!Комбинация Alt-5 на цифровой клавиатуре
AltPad6	EQUATE(0466H)	!Комбинация Alt-6 на цифровой клавиатуре
AltPad7	EQUATE(0467H)	!Комбинация Alt-7 на цифровой клавиатуре
AltPad8	EQUATE(0468H)	!Комбинация Alt-8 на цифровой клавиатуре
AltPad9	EQUATE(0469H)	!Комбинация Alt-9 на цифровой клавиатуре
AltMouseLeft	EQUATE(0401H)	!Alt-левая кнопка мыши
AltMouseRight	EQUATE(0402H)	!Alt-правая кнопка мыши
AltMouseCenter	EQUATE(0403H)	!Alt-средняя кнопка мыши
AltMouseLeft2	EQUATE(0405H)	!Alt-Двойной щелчок левой кнопкой мыши
AltMouseRight2	EQUATE(0406H)	!Alt-Двойной щелчок правой кнопкой мыши
AltMouseCenter2	EQUATE(0407H)	!Alt-Двойной щелчок средней кнопкой мыши
Ctrl0	EQUATE(0230H)	!Комбинация Ctrl-0
Ctrl1	EQUATE(0231H)	!Комбинация Ctrl-1
Ctrl2	EQUATE(0232H)	!Комбинация Ctrl-2
Ctrl3	EQUATE(0233H)	!Комбинация Ctrl-3
Ctrl4	EQUATE(0234H)	!Комбинация Ctrl-4
Ctrl5	EQUATE(0235H)	!Комбинация Ctrl-5
Ctrl6	EQUATE(0236H)	!Комбинация Ctrl-6
Ctrl7	EQUATE(0237H)	!Комбинация Ctrl-7
Ctrl8	EQUATE(0238H)	!Комбинация Ctrl-8

Ctrl9	EQUATE(0239H)	!Комбинация Ctrl-9
CtrlA	EQUATE(0241H)	!Комбинация Ctrl-A
CtrlB	EQUATE(0242H)	!Комбинация Ctrl-B
CtrlC	EQUATE(0243H)	!Комбинация Ctrl-C
CtrlD	EQUATE(0244H)	!Комбинация Ctrl-D
CtrlE	EQUATE(0245H)	!Комбинация Ctrl-E
CtrlF	EQUATE(0246H)	!Комбинация Ctrl-F
CtrlG	EQUATE(0247H)	!Комбинация Ctrl-G
CtrlH	EQUATE(0248H)	!Комбинация Ctrl-H
CtrlI	EQUATE(0249H)	!Комбинация Ctrl-I
CtrlJ	EQUATE(024AH)	!Комбинация Ctrl-J
CtrlK	EQUATE(024BH)	!Комбинация Ctrl-K
CtrlL	EQUATE(024CH)	!Комбинация Ctrl-L
CtrlM	EQUATE(024DH)	!Комбинация Ctrl-M
CtrlN	EQUATE(024EH)	!Комбинация Ctrl-N
CtrlO	EQUATE(024FH)	!Комбинация Ctrl-O
CtrlP	EQUATE(0250H)	!Комбинация Ctrl-P
CtrlQ	EQUATE(0251H)	!Комбинация Ctrl-Q
CtrlR	EQUATE(0252H)	!Комбинация Ctrl-R
CtrlS	EQUATE(0253H)	!Комбинация Ctrl-S
CtrlT	EQUATE(0254H)	!Комбинация Ctrl-T
CtrlU	EQUATE(0255H)	!Комбинация Ctrl-U
CtrlV	EQUATE(0256H)	!Комбинация Ctrl-V
CtrlW	EQUATE(0257H)	!Комбинация Ctrl-W
CtrlX	EQUATE(0258H)	!Комбинация Ctrl-X
CtrlY	EQUATE(0259H)	!Комбинация Ctrl-Y
CtrlZ	EQUATE(025AH)	!Комбинация Ctrl-Z
CtrlF1	EQUATE(0270H)	!Комбинация Ctrl-F1
CtrlF2	EQUATE(0271H)	!Комбинация Ctrl-F2
CtrlF3	EQUATE(0272H)	!Комбинация Ctrl-F3
CtrlF4	EQUATE(0273H)	!Комбинация Ctrl-F4
CtrlF5	EQUATE(0274H)	!Комбинация Ctrl-F5
CtrlF6	EQUATE(0275H)	!Комбинация Ctrl-F6
CtrlF7	EQUATE(0276H)	!Комбинация Ctrl-F7
CtrlF8	EQUATE(0277H)	!Комбинация Ctrl-F8
CtrlF9	EQUATE(0278H)	!Комбинация Ctrl-F9
CtrlF10	EQUATE(0279H)	!Комбинация Ctrl-F10
CtrlF11	EQUATE(027AH)	!Комбинация Ctrl-F11
CtrlF12	EQUATE(027BH)	!Комбинация Ctrl-F12
CtrlAst	EQUATE(026AH)	!Комбинация Ctrl- звездочка
CtrlBS	EQUATE(0208H)	!Комбинация Ctrl-Backspace
CtrlDecimal	EQUATE(026EH)	!Комбинация Ctrl-Decimal
CtrlDelete	EQUATE(022EH)	!Комбинация Ctrl-Delete
CtrlDown	EQUATE(0228H)	!Комбинация Ctrl-Стрелка вниз
CtrlEnd	EQUATE(0223H)	!Комбинация Ctrl-End

CtrlEnter	EQUATE(020DH)	!Комбинация Ctrl-Enter
CtrlEsc	EQUATE(021BH)	!Комбинация Ctrl-Esc
CtrlHome	EQUATE(0224H)	!Комбинация Ctrl-Home
CtrlInsert	EQUATE(022DH)	!Комбинация Ctrl-Insert
CtrlLeft	EQUATE(0225H)	!Комбинация Ctrl-стрелка влево
CtrlMinus	EQUATE(026DH)	!Комбинация Ctrl-Minus
CtrlPause	EQUATE(0213H)	!Комбинация Ctrl-Pause
CtrlPgDn	EQUATE(0222H)	!Комбинация Ctrl-PgDn
CtrlPgUp	EQUATE(0221H)	!Комбинация Ctrl-PgUp
CtrlPlus	EQUATE(026BH)	!Комбинация Ctrl-Plus
CtrlPrint	EQUATE(022CH)	!Комбинация Ctrl-PrintScreen
CtrlRight	EQUATE(0227H)	!Комбинация Ctrl-стрелка вправо
CtrlSlash	EQUATE(026FH)	!Комбинация Ctrl-слэш
CtrlSpace	EQUATE(0220H)	!Комбинация Ctrl-пробел
CtrlTab	EQUATE(0209H)	!Комбинация Ctrl-Tab
CtrlUp	EQUATE(0226H)	!Комбинация Ctrl-стрелка вверх
CtrlPad0	EQUATE(0260H)	!Комбинация Ctrl-0 на цифровой клавиатуре
CtrlPad1	EQUATE(0261H)	!Комбинация Ctrl-1 на цифровой клавиатуре
CtrlPad2	EQUATE(0262H)	!Комбинация Ctrl-2 на цифровой клавиатуре
CtrlPad3	EQUATE(0263H)	!Комбинация Ctrl-3 на цифровой клавиатуре
CtrlPad4	EQUATE(0264H)	!Комбинация Ctrl-4 на цифровой клавиатуре
CtrlPad5	EQUATE(0265H)	!Комбинация Ctrl-5 на цифровой клавиатуре
CtrlPad6	EQUATE(0266H)	!Комбинация Ctrl-6 на цифровой клавиатуре
CtrlPad7	EQUATE(0267H)	!Комбинация Ctrl-7 на цифровой клавиатуре
CtrlPad8	EQUATE(0268H)	!Комбинация Ctrl-8 на цифровой клавиатуре
CtrlPad9	EQUATE(0269H)	!Комбинация Ctrl-9 на цифровой клавиатуре
CtrlMouseLeft	EQUATE(0201H)	!Комбинация Ctrl-левая кнопка мыши
CtrlMouseRight	EQUATE(0202H)	!Комбинация Ctrl-правая кнопка мыши
CtrlMouseCenter	EQUATE(0203H)	!Комбинация Ctrl-средняя кнопка мыши
CtrlMouseLeft2	EQUATE(0205H)	!Комбинация Ctrl-Двойной щелчок левой кнопкой мыши
CtrlMouseRight2	EQUATE(0206H)	!Комбинация Ctrl-Двойной щелчок правой кнопкой мыши
CtrlMouseCenter2	EQUATE(0207H)	!Комбинация Ctrl-Двойной щелчок средней кнопкой мыши
Shift0	EQUATE(0130H)	!Комбинация Shift-0
Shift1	EQUATE(0131H)	!Комбинация Shift-1
Shift2	EQUATE(0132H)	!Комбинация Shift-2
Shift3	EQUATE(0133H)	!Комбинация Shift-3
Shift4	EQUATE(0134H)	!Комбинация Shift-4
Shift5	EQUATE(0135H)	!Комбинация Shift-5
Shift6	EQUATE(0136H)	!Комбинация Shift-6
Shift7	EQUATE(0137H)	!Комбинация Shift-7
Shift8	EQUATE(0138H)	!Комбинация Shift-8
Shift9	EQUATE(0139H)	!Комбинация Shift-9
ShiftA	EQUATE(0141H)	!Комбинация Shift-A

ShiftB	EQUATE(0142H)	!Комбинация Shift-B
ShiftC	EQUATE(0143H)	!Комбинация Shift-C
ShiftD	EQUATE(0144H)	!Комбинация Shift-D
ShiftE	EQUATE(0145H)	!Комбинация Shift-E
ShiftF	EQUATE(0146H)	!Комбинация Shift-F
ShiftG	EQUATE(0147H)	!Комбинация Shift-G
ShiftH	EQUATE(0148H)	!Комбинация Shift-H
ShiftI	EQUATE(0149H)	!Комбинация Shift-I
ShiftJ	EQUATE(014AH)	!Комбинация Shift-J
ShiftK	EQUATE(014BH)	!Комбинация Shift-K
ShiftL	EQUATE(014CH)	!Комбинация Shift-L
ShiftM	EQUATE(014DH)	!Комбинация Shift-M
ShiftN	EQUATE(014EH)	!Комбинация Shift-N
ShiftO	EQUATE(014FH)	!Комбинация Shift-O
ShiftP	EQUATE(0150H)	!Комбинация Shift-P
ShiftQ	EQUATE(0151H)	!Комбинация Shift-Q
ShiftR	EQUATE(0152H)	!Комбинация Shift-R
ShiftS	EQUATE(0153H)	!Комбинация Shift-S
ShiftT	EQUATE(0154H)	!Комбинация Shift-T
ShiftU	EQUATE(0155H)	!Комбинация Shift-U
ShiftV	EQUATE(0156H)	!Комбинация Shift-V
ShiftW	EQUATE(0157H)	!Комбинация Shift-W
ShiftX	EQUATE(0158H)	!Комбинация Shift-X
ShiftY	EQUATE(0159H)	!Комбинация Shift-Y
ShiftZ	EQUATE(015AH)	!Комбинация Shift-Z
ShiftF1	EQUATE(0170H)	!Комбинация Shift-F1
ShiftF2	EQUATE(0171H)	!Комбинация Shift-F2
ShiftF3	EQUATE(0172H)	!Комбинация Shift-F3
ShiftF4	EQUATE(0173H)	!Комбинация Shift-F4
ShiftF5	EQUATE(0174H)	!Комбинация Shift-F5
ShiftF6	EQUATE(0175H)	!Комбинация Shift-F6
ShiftF7	EQUATE(0176H)	!Комбинация Shift-F7
ShiftF8	EQUATE(0177H)	!Комбинация Shift-F8
ShiftF9	EQUATE(0178H)	!Комбинация Shift-F9
ShiftF10	EQUATE(0179H)	!Комбинация Shift-F10
ShiftF11	EQUATE(017AH)	!Комбинация Shift-F11
ShiftF12	EQUATE(017BH)	!Комбинация Shift-F12
ShiftAst	EQUATE(016AH)	!Комбинация Shift-звездочка
ShiftBS	EQUATE(0108H)	!Комбинация Shift-Backspace
ShiftDecimal	EQUATE(016EH)	!Комбинация Shift-десятичная точка
ShiftDelete	EQUATE(012EH)	!Комбинация Shift-Delete
ShiftDown	EQUATE(0128H)	!Комбинация Shift-стрелка вниз
ShiftEnd	EQUATE(0123H)	!Комбинация Shift-End
ShiftEnter	EQUATE(010DH)	!Комбинация Shift-Enter
ShiftEsc	EQUATE(011BH)	!Комбинация Shift-Esc
ShiftHome	EQUATE(0124H)	!Комбинация Shift-Home

ShiftInsert	EQUATE(012DH)	!Комбинация Shift-Insert
ShiftLeft	EQUATE(0125H)	!Комбинация Shift-стрелка влево
ShiftMinus	EQUATE(016DH)	!Комбинация Shift-Minus
ShiftPause	EQUATE(0113H)	!Комбинация Shift-Pause
ShiftPgDn	EQUATE(0122H)	!Комбинация Shift-PgDn
ShiftPgUp	EQUATE(0121H)	!Комбинация Shift-PgUp
ShiftPlus	EQUATE(016BH)	!Комбинация Shift-Plus
ShiftPrint	EQUATE(012CH)	!Комбинация Shift-PrintScreen
ShiftRight	EQUATE(0127H)	!Комбинация Shift-стрелка вправо
ShiftSlash	EQUATE(016FH)	!Комбинация Shift-Slash
ShiftSpace	EQUATE(0120H)	!Комбинация Shift-пробел
ShiftTab	EQUATE(0109H)	!Комбинация Shift-Tab
ShiftUp	EQUATE(0126H)	!Комбинация Shift-стрелка вверх
ShiftPad0	EQUATE(0160H)	!Комбинация Shift-0 на цифровой клавиатуре
ShiftPad1	EQUATE(0161H)	!Комбинация Shift-1 на цифровой клавиатуре
ShiftPad2	EQUATE(0162H)	!Комбинация Shift-2 на цифровой клавиатуре
ShiftPad3	EQUATE(0163H)	!Комбинация Shift-3 на цифровой клавиатуре
ShiftPad4	EQUATE(0164H)	!Комбинация Shift-4 на цифровой клавиатуре
ShiftPad5	EQUATE(0165H)	!Комбинация Shift-5 на цифровой клавиатуре
ShiftPad6	EQUATE(0166H)	!Комбинация Shift-6 на цифровой клавиатуре
ShiftPad7	EQUATE(0167H)	!Комбинация Shift-7 на цифровой клавиатуре
ShiftPad8	EQUATE(0168H)	!Комбинация Shift-8 на цифровой клавиатуре
ShiftPad9	EQUATE(0169H)	!Комбинация Shift-9 на цифровой клавиатуре
ShiftMouseLeft	EQUATE(0101H)	!Комбинация Shift-левая кнопка мыши
ShiftMouseRight	EQUATE(0102H)	!Комбинация Shift-правая кнопка мыши
ShiftMouseCenter	EQUATE(0103H)	!Комбинация Shift-средняя кнопка мыши
ShiftMouseLeft2	EQUATE(0105H)	!Комбинация Shift-двойной щелчок левой кнопкой мыши
ShiftMouseRight2	EQUATE(0106H)	!Комбинация Shift-двойной щелчок правой кнопкой мыши
ShiftMouseCenter2	EQUATE(0107H)	!Комбинация Shift-двойной щелчок средней кнопкой мыши
AltShift0	EQUATE(0530H)	!Комбинация Alt-Shift-0
AltShift1	EQUATE(0531H)	!Комбинация Alt-Shift-1
AltShift2	EQUATE(0532H)	!Комбинация Alt-Shift-2
AltShift3	EQUATE(0533H)	!Комбинация Alt-Shift-3
AltShift4	EQUATE(0534H)	!Комбинация Alt-Shift-4
AltShift5	EQUATE(0535H)	!Комбинация Alt-Shift-5
AltShift6	EQUATE(0536H)	!Комбинация Alt-Shift-6
AltShift7	EQUATE(0537H)	!Комбинация Alt-Shift-7
AltShift8	EQUATE(0538H)	!Комбинация Alt-Shift-8
AltShift9	EQUATE(0539H)	!Комбинация Alt-Shift-9
AltShiftA	EQUATE(0541H)	!Комбинация Alt-Shift-A
AltShiftB	EQUATE(0542H)	!Комбинация Alt-Shift-B
AltShiftC	EQUATE(0543H)	!Комбинация Alt-Shift-C

AltShiftD	EQUATE(0544H)	!Комбинация Alt-Shift-D
AltShiftE	EQUATE(0545H)	!Комбинация Alt-Shift-E
AltShiftF	EQUATE(0546H)	!Комбинация Alt-Shift-F
AltShiftG	EQUATE(0547H)	!Комбинация Alt-Shift-G
AltShiftH	EQUATE(0548H)	!Комбинация Alt-Shift-H
AltShiftI	EQUATE(0549H)	!Комбинация Alt-Shift-I
AltShiftJ	EQUATE(054AH)	!Комбинация Alt-Shift-J
AltShiftK	EQUATE(054BH)	!Комбинация Alt-Shift-K
AltShiftL	EQUATE(054CH)	!Комбинация Alt-Shift-L
AltShiftM	EQUATE(054DH)	!Комбинация Alt-Shift-M
AltShiftN	EQUATE(054EH)	!Комбинация Alt-Shift-N
AltShiftO	EQUATE(054FH)	!Комбинация Alt-Shift-O
AltShiftP	EQUATE(0550H)	!Комбинация Alt-Shift-P
AltShiftQ	EQUATE(0551H)	!Комбинация Alt-Shift-Q
AltShiftR	EQUATE(0552H)	!Комбинация Alt-Shift-R
AltShiftS	EQUATE(0553H)	!Комбинация Alt-Shift-S
AltShiftT	EQUATE(0554H)	!Комбинация Alt-Shift-T
AltShiftU	EQUATE(0555H)	!Комбинация Alt-Shift-U
AltShiftV	EQUATE(0556H)	!Комбинация Alt-Shift-V
AltShiftW	EQUATE(0557H)	!Комбинация Alt-Shift-W
AltShiftX	EQUATE(0558H)	!Комбинация Alt-Shift-X
AltShiftY	EQUATE(0559H)	!Комбинация Alt-Shift-Y
AltShiftZ	EQUATE(055AH)	!Комбинация Alt-Shift-Z
AltShiftF1	EQUATE(0570H)	!Комбинация Alt-Shift-F1
AltShiftF2	EQUATE(0571H)	!Комбинация Alt-Shift-F2
AltShiftF3	EQUATE(0572H)	!Комбинация Alt-Shift-F3
AltShiftF4	EQUATE(0573H)	!Комбинация Alt-Shift-F4
AltShiftF5	EQUATE(0574H)	!Комбинация Alt-Shift-F5
AltShiftF6	EQUATE(0575H)	!Комбинация Alt-Shift-F6
AltShiftF7	EQUATE(0576H)	!Комбинация Alt-Shift-F7
AltShiftF8	EQUATE(0577H)	!Комбинация Alt-Shift-F8
AltShiftF9	EQUATE(0578H)	!Комбинация Alt-Shift-F9
AltShiftF10	EQUATE(0579H)	!Комбинация Alt-Shift-F10
AltShiftF11	EQUATE(057AH)	!Комбинация Alt-Shift-F11
AltShiftF12	EQUATE(057BH)	!Комбинация Alt-Shift-F12
AltShiftAst	EQUATE(056AH)	!Комбинация Alt-Shift-звездочка
AltShiftBS	EQUATE(0508H)	!Комбинация Alt-Shift-забой
AltShiftDecimal	EQUATE(056EH)	!Комбинация Alt-Shift-десятичная точка
AltShiftDelete	EQUATE(052EH)	!Комбинация Alt-Shift-Delete
AltShiftDown	EQUATE(0528H)	!Комбинация Alt-Shift-стрелка вниз
AltShiftEnd	EQUATE(0523H)	!Комбинация Alt-Shift-End
AltShiftEnter	EQUATE(050DH)	!Комбинация Alt-Shift-Enter
AltShiftEsc	EQUATE(051BH)	!Комбинация Alt-Shift-Esc
AltShiftHome	EQUATE(0524H)	!Комбинация Alt-Shift-Home
AltShiftInsert	EQUATE(052DH)	!Комбинация Alt-Shift-Insert
AltShiftLeft	EQUATE(0525H)	!Комбинация Alt-Shift-стрелка влево



AltShiftMinus	EQUATE(056DH)	!Комбинация Alt-Shift-Minus
AltShiftPause	EQUATE(0513H)	!Комбинация Alt-Shift-Pause
AltShiftPgDn	EQUATE(0522H)	!Комбинация Alt-Shift-PgDn
AltShiftPgUp	EQUATE(0521H)	!Комбинация Alt-Shift-PgUp
AltShiftPlus	EQUATE(056BH)	!Комбинация Alt-Shift-Plus
AltShiftPrint	EQUATE(052CH)	!Комбинация Alt-Shift-PrintScreen
AltShiftRight	EQUATE(0527H)	!Комбинация Alt-Shift-стрелка вправо
AltShiftSlash	EQUATE(056FH)	!Комбинация Alt-Shift-слэш
AltShiftSpace	EQUATE(0520H)	!Комбинация Alt-Shift-пробел
AltShiftTab	EQUATE(0509H)	!Комбинация Alt-Shift-Tab
AltShiftUp	EQUATE(0526H)	!Комбинация Alt-Shift-стрелка вверх
AltShiftPad0	EQUATE(0560H)	!Комбинация Alt-Shift-0 на цифровой клавиатуре
AltShiftPad1	EQUATE(0561H)	!Комбинация Alt-Shift-1 на цифровой клавиатуре
AltShiftPad2	EQUATE(0562H)	!Комбинация Alt-Shift-2 на цифровой клавиатуре
AltShiftPad3	EQUATE(0563H)	!Комбинация Alt-Shift-3 на цифровой клавиатуре
AltShiftPad4	EQUATE(0564H)	!Комбинация Alt-Shift-4 на цифровой клавиатуре
AltShiftPad5	EQUATE(0565H)	!Комбинация Alt-Shift-5 на цифровой клавиатуре
AltShiftPad6	EQUATE(0566H)	!Комбинация Alt-Shift-6 на цифровой клавиатуре
AltShiftPad7	EQUATE(0567H)	!Комбинация Alt-Shift-7 на цифровой клавиатуре
AltShiftPad8	EQUATE(0568H)	!Комбинация Alt-Shift-8 на цифровой клавиатуре
AltShiftPad9	EQUATE(0569H)	!Комбинация Alt-Shift-9 на цифровой клавиатуре
AltShiftMouseLeft	EQUATE(0501H)	!Комбинация Alt-Shift-левая кнопка мыши
AltShiftMouseRight	EQUATE(0502H)	!Комбинация Alt-Shift-правая кнопка мыши
AltShiftMouseCenter	EQUATE(0503H)	!Комбинация Alt-Shift-средняя кнопка мыши
AltShiftMouseLeft2	EQUATE(0505H)	!Комбинация Alt-Shift-двойной щелчок левой кнопкой мыши
AltShiftMouseRight2	EQUATE(0506H)	!Комбинация Alt-Shift-двойной щелчок правой кнопкой мыши
AltShiftMouseCenter2	EQUATE(0507H)	!Комбинация Alt-Shift-двойной щелчок средней кнопкой мыши
CtrlShift0	EQUATE(0330H)	!Комбинация Ctrl-Shift-0
CtrlShift1	EQUATE(0331H)	!Комбинация Ctrl-Shift-1
CtrlShift2	EQUATE(0332H)	!Комбинация Ctrl-Shift-2
CtrlShift3	EQUATE(0333H)	!Комбинация Ctrl-Shift-3
CtrlShift4	EQUATE(0334H)	!Комбинация Ctrl-Shift-4
CtrlShift5	EQUATE(0335H)	!Комбинация Ctrl-Shift-5
CtrlShift6	EQUATE(0336H)	!Комбинация Ctrl-Shift-6
CtrlShift7	EQUATE(0337H)	!Комбинация Ctrl-Shift-7
CtrlShift8	EQUATE(0338H)	!Комбинация Ctrl-Shift-8
CtrlShift9	EQUATE(0339H)	!Комбинация Ctrl-Shift-9
CtrlShiftA	EQUATE(0341H)	!Комбинация Ctrl-Shift-A
CtrlShiftB	EQUATE(0342H)	!Комбинация Ctrl-Shift-B
CtrlShiftC	EQUATE(0343H)	!Комбинация Ctrl-Shift-C

CtrlShiftD	EQUATE(0344H)	!Комбинация Ctrl-Shift-D
CtrlShiftE	EQUATE(0345H)	!Комбинация Ctrl-Shift-E
CtrlShiftF	EQUATE(0346H)	!Комбинация Ctrl-Shift-F
CtrlShiftG	EQUATE(0347H)	!Комбинация Ctrl-Shift-G
CtrlShiftH	EQUATE(0348H)	!Комбинация Ctrl-Shift-H
CtrlShiftI	EQUATE(0349H)	!Комбинация Ctrl-Shift-I
CtrlShiftJ	EQUATE(034AH)	!Комбинация Ctrl-Shift-J
CtrlShiftK	EQUATE(034BH)	!Комбинация Ctrl-Shift-K
CtrlShiftL	EQUATE(034CH)	!Комбинация Ctrl-Shift-L
CtrlShiftM	EQUATE(034DH)	!Комбинация Ctrl-Shift-M
CtrlShiftN	EQUATE(034EH)	!Комбинация Ctrl-Shift-N
CtrlShiftO	EQUATE(034FH)	!Комбинация Ctrl-Shift-O
CtrlShiftP	EQUATE(0350H)	!Комбинация Ctrl-Shift-P
CtrlShiftQ	EQUATE(0351H)	!Комбинация Ctrl-Shift-Q
CtrlShiftR	EQUATE(0352H)	!Комбинация Ctrl-Shift-R
CtrlShiftS	EQUATE(0353H)	!Комбинация Ctrl-Shift-S
CtrlShiftT	EQUATE(0354H)	!Комбинация Ctrl-Shift-T
CtrlShiftU	EQUATE(0355H)	!Комбинация Ctrl-Shift-U
CtrlShiftV	EQUATE(0356H)	!Комбинация Ctrl-Shift-V
CtrlShiftW	EQUATE(0357H)	!Комбинация Ctrl-Shift-W
CtrlShiftX	EQUATE(0358H)	!Комбинация Ctrl-Shift-X
CtrlShiftY	EQUATE(0359H)	!Комбинация Ctrl-Shift-Y
CtrlShiftZ	EQUATE(035AH)	!Комбинация Ctrl-Shift-Z
CtrlShiftF1	EQUATE(0370H)	!Комбинация Ctrl-Shift-F1
CtrlShiftF2	EQUATE(0371H)	!Комбинация Ctrl-Shift-F2
CtrlShiftF3	EQUATE(0372H)	!Комбинация Ctrl-Shift-F3
CtrlShiftF4	EQUATE(0373H)	!Комбинация Ctrl-Shift-F4
CtrlShiftF5	EQUATE(0374H)	!Комбинация Ctrl-Shift-F5
CtrlShiftF6	EQUATE(0375H)	!Комбинация Ctrl-Shift-F6
CtrlShiftF7	EQUATE(0376H)	!Комбинация Ctrl-Shift-F7
CtrlShiftF8	EQUATE(0377H)	!Комбинация Ctrl-Shift-F8
CtrlShiftF9	EQUATE(0378H)	!Комбинация Ctrl-Shift-F9
CtrlShiftF10	EQUATE(0379H)	!Комбинация Ctrl-Shift-F10
CtrlShiftF11	EQUATE(037AH)	!Комбинация Ctrl-Shift-F11
CtrlShiftF12	EQUATE(037BH)	!Комбинация Ctrl-Shift-F12
CtrlShiftAst	EQUATE(036AH)	!Комбинация Ctrl-Shift-звездочка
CtrlShiftBS	EQUATE(0308H)	!Комбинация Ctrl-Shift-забой
CtrlShiftDecimal	EQUATE(036EH)	!Комбинация Ctrl-Shift-десятичная точка
CtrlShiftDelete	EQUATE(032EH)	!Комбинация Ctrl-Shift-Delete
CtrlShiftDown	EQUATE(0328H)	!Комбинация Ctrl-Shift-стрелка вниз
CtrlShiftEnd	EQUATE(0323H)	!Комбинация Ctrl-Shift-End
CtrlShiftEnter	EQUATE(030DH)	!Комбинация Ctrl-Shift-Enter
CtrlShiftEsc	EQUATE(031BH)	!Комбинация Ctrl-Shift-Esc
CtrlShiftHome	EQUATE(0324H)	!Комбинация Ctrl-Shift-Home
CtrlShiftInsert	EQUATE(032DH)	!Комбинация Ctrl-Shift-Insert

CtrlShiftLeft	EQUATE(0325H)	!Комбинация Ctrl-Shift-стрелка влево
CtrlShiftMinus	EQUATE(036DH)	!Комбинация Ctrl-Shift-минус
CtrlShiftPause	EQUATE(0313H)	!Комбинация Ctrl-Shift-Pause
CtrlShiftPgDn	EQUATE(0322H)	!Комбинация Ctrl-Shift-PgDn
CtrlShiftPgUp	EQUATE(0321H)	!Комбинация Ctrl-Shift-PgUp
CtrlShiftPlus	EQUATE(036BH)	!Комбинация Ctrl-Shift-Plus
CtrlShiftPrint	EQUATE(032CH)	!Комбинация Ctrl-Shift-PrintScreen
CtrlShiftRight	EQUATE(0327H)	!Комбинация Ctrl-Shift-стрелка вправо
CtrlShiftSlash	EQUATE(036FH)	!Комбинация Ctrl-Shift-слэш
CtrlShiftSpace	EQUATE(0320H)	!Комбинация Ctrl-Shift-пробел
CtrlShiftTab	EQUATE(0309H)	!Комбинация Ctrl-Shift-Tab
CtrlShiftUp	EQUATE(0326H)	!Комбинация Ctrl-Shift-стрелка вверх
CtrlShiftPad0	EQUATE(0360H)	!Комбинация Ctrl-Shift-0 на цифровой клавиатуре
CtrlShiftPad1	EQUATE(0361H)	!Комбинация Ctrl-Shift-1 на цифровой клавиатуре
CtrlShiftPad2	EQUATE(0362H)	!Комбинация Ctrl-Shift-2 на цифровой клавиатуре
CtrlShiftPad3	EQUATE(0363H)	!Комбинация Ctrl-Shift-3 на цифровой клавиатуре
CtrlShiftPad4	EQUATE(0364H)	!Комбинация Ctrl-Shift-4 на цифровой клавиатуре
CtrlShiftPad5	EQUATE(0365H)	!Комбинация Ctrl-Shift-5 на цифровой клавиатуре
CtrlShiftPad6	EQUATE(0366H)	!Комбинация Ctrl-Shift-6 на цифровой клавиатуре
CtrlShiftPad7	EQUATE(0367H)	!Комбинация Ctrl-Shift-7 на цифровой клавиатуре
CtrlShiftPad8	EQUATE(0368H)	!Комбинация Ctrl-Shift-8 на цифровой клавиатуре
CtrlShiftPad9	EQUATE(0369H)	!Комбинация Ctrl-Shift-9 на цифровой клавиатуре
CtrlShiftMouseLeft	EQUATE(0301H)	!Комбинация Ctrl-Shift-левая кнопка мыши
CtrlShiftMouseRight	EQUATE(0302H)	!Комбинация Ctrl-Shift-правая кнопка мыши
CtrlShiftMouseCenter	EQUATE(0303H)	!Комбинация Ctrl-Shift-средняя кнопка мыши
CtrlShiftMouseLeft2	EQUATE(0305H)	!Комбинация Ctrl-Shift-двойной щелчок левой
кнопкой мыши		
CtrlShiftMouseRight2	EQUATE(0306H)	!Комбинация Ctrl-Shift-двойной щелчок правой
кнопкой мыши		
CtrlShiftMouseCenter2	EQUATE(0307H)	!Комбинация Ctrl-Shift-двойной щелчок средней
кнопкой мыши		
CtrlAlt0	EQUATE(0630H)	!Комбинация Ctrl-Alt-0
CtrlAlt1	EQUATE(0631H)	!Комбинация Ctrl-Alt-1
CtrlAlt2	EQUATE(0632H)	!Комбинация Ctrl-Alt-2
CtrlAlt3	EQUATE(0633H)	!Комбинация Ctrl-Alt-3
CtrlAlt4	EQUATE(0634H)	!Комбинация Ctrl-Alt-4
CtrlAlt5	EQUATE(0635H)	!Комбинация Ctrl-Alt-5
CtrlAlt6	EQUATE(0636H)	!Комбинация Ctrl-Alt-6
CtrlAlt7	EQUATE(0637H)	!Комбинация Ctrl-Alt-7
CtrlAlt8	EQUATE(0638H)	!Комбинация Ctrl-Alt-8
CtrlAlt9	EQUATE(0639H)	!Комбинация Ctrl-Alt-9
CtrlAltA	EQUATE(0641H)	!Комбинация Ctrl-Alt-A
CtrlAltB	EQUATE(0642H)	!Комбинация Ctrl-Alt-B

CtrlAltC	EQUATE(0643H)	!Комбинация Ctrl-Alt-C
CtrlAltD	EQUATE(0644H)	!Комбинация Ctrl-Alt-D
CtrlAltE	EQUATE(0645H)	!Комбинация Ctrl-Alt-E
CtrlAltF	EQUATE(0646H)	!Комбинация Ctrl-Alt-F
CtrlAltG	EQUATE(0647H)	!Комбинация Ctrl-Alt-G
CtrlAltH	EQUATE(0648H)	!Комбинация Ctrl-Alt-H
CtrlAltI	EQUATE(0649H)	!Комбинация Ctrl-Alt-I
CtrlAltJ	EQUATE(064AH)	!Комбинация Ctrl-Alt-J
CtrlAltK	EQUATE(064BH)	!Комбинация Ctrl-Alt-K
CtrlAltL	EQUATE(064CH)	!Комбинация Ctrl-Alt-L
CtrlAltM	EQUATE(064DH)	!Комбинация Ctrl-Alt-M
CtrlAltN	EQUATE(064EH)	!Комбинация Ctrl-Alt-N
CtrlAltO	EQUATE(064FH)	!Комбинация Ctrl-Alt-O
CtrlAltP	EQUATE(0650H)	!Комбинация Ctrl-Alt-P
CtrlAltQ	EQUATE(0651H)	!Комбинация Ctrl-Alt-Q
CtrlAltR	EQUATE(0652H)	!Комбинация Ctrl-Alt-R
CtrlAltS	EQUATE(0653H)	!Комбинация Ctrl-Alt-S
CtrlAltT	EQUATE(0654H)	!Комбинация Ctrl-Alt-T
CtrlAltU	EQUATE(0655H)	!Комбинация Ctrl-Alt-U
CtrlAltV	EQUATE(0656H)	!Комбинация Ctrl-Alt-V
CtrlAltW	EQUATE(0657H)	!Комбинация Ctrl-Alt-W
CtrlAltX	EQUATE(0658H)	!Комбинация Ctrl-Alt-X
CtrlAltY	EQUATE(0659H)	!Комбинация Ctrl-Alt-Y
CtrlAltZ	EQUATE(065AH)	!Комбинация Ctrl-Alt-Z
CtrlAltF1	EQUATE(0670H)	!Комбинация Ctrl-Alt-F1
CtrlAltF2	EQUATE(0671H)	!Комбинация Ctrl-Alt-F2
CtrlAltF3	EQUATE(0672H)	!Комбинация Ctrl-Alt-F3
CtrlAltF4	EQUATE(0673H)	!Комбинация Ctrl-Alt-F4
CtrlAltF5	EQUATE(0674H)	!Комбинация Ctrl-Alt-F5
CtrlAltF6	EQUATE(0675H)	!Комбинация Ctrl-Alt-F6
CtrlAltF7	EQUATE(0676H)	!Комбинация Ctrl-Alt-F7
CtrlAltF8	EQUATE(0677H)	!Комбинация Ctrl-Alt-F8
CtrlAltF9	EQUATE(0678H)	!Комбинация Ctrl-Alt-F9
CtrlAltF10	EQUATE(0679H)	!Комбинация Ctrl-Alt-F10
CtrlAltF11	EQUATE(067AH)	!Комбинация Ctrl-Alt-F11
CtrlAltF12	EQUATE(067BH)	!Комбинация Ctrl-Alt-F12
CtrlAltAst	EQUATE(066AH)	!Комбинация Ctrl-Alt-звездочка
CtrlAltBS	EQUATE(0608H)	!Комбинация Ctrl-Alt-забой
CtrlAltDecimal	EQUATE(066EH)	!Комбинация Ctrl-Alt-десятичная точка
CtrlAltDelete	EQUATE(062EH)	!Комбинация Ctrl-Alt-Delete
CtrlAltDown	EQUATE(0628H)	!Комбинация Ctrl-Alt-стрелка вниз
CtrlAltEnd	EQUATE(0623H)	!Комбинация Ctrl-Alt-End
CtrlAltEnter	EQUATE(060DH)	!Комбинация Ctrl-Alt-Enter
CtrlAltEsc	EQUATE(061BH)	!Комбинация Ctrl-Alt-Esc
CtrlAltHome	EQUATE(0624H)	!Комбинация Ctrl-Alt-Home

CtrlAltInsert	EQUATE(062DH)	!Комбинация Ctrl-Alt-Insert
CtrlAltLeft	EQUATE(0625H)	!Комбинация Ctrl-Alt-стрелка влево
CtrlAltMinus	EQUATE(066DH)	!Комбинация Ctrl-Alt-Minus
CtrlAltPause	EQUATE(0613H)	!Комбинация Ctrl-Alt-Pause
CtrlAltPgDn	EQUATE(0622H)	!Комбинация Ctrl-Alt-PgDn
CtrlAltPgUp	EQUATE(0621H)	!Комбинация Ctrl-Alt-PgUp
CtrlAltPlus	EQUATE(066BH)	!Комбинация Ctrl-Alt-Plus
CtrlAltPrint	EQUATE(062CH)	!Комбинация Ctrl-Alt-PrintScreen
CtrlAltRight	EQUATE(0627H)	!Комбинация Ctrl-Alt-стрелка вправо
CtrlAltSlash	EQUATE(066FH)	!Комбинация Ctrl-Alt-слэш
CtrlAltSpace	EQUATE(0620H)	!Ctrl-Alt-пробел
CtrlAltTab	EQUATE(0609H)	!Комбинация Ctrl-Alt-Tab
CtrlAltUp	EQUATE(0626H)	!Комбинация Ctrl-Alt-стрелка вверх
CtrlAltPad0	EQUATE(0660H)	!Комбинация Ctrl-Alt-0 на цифровой клавиатуре
CtrlAltPad1	EQUATE(0661H)	!Комбинация Ctrl-Alt-1 на цифровой клавиатуре
CtrlAltPad2	EQUATE(0662H)	!Комбинация Ctrl-Alt-2 на цифровой клавиатуре
CtrlAltPad3	EQUATE(0663H)	!Комбинация Ctrl-Alt-3 на цифровой клавиатуре
CtrlAltPad4	EQUATE(0664H)	!Комбинация Ctrl-Alt-4 на цифровой клавиатуре
CtrlAltPad5	EQUATE(0665H)	!Комбинация Ctrl-Alt-5 на цифровой клавиатуре
CtrlAltPad6	EQUATE(0666H)	!Комбинация Ctrl-Alt-6 на цифровой клавиатуре
CtrlAltPad7	EQUATE(0667H)	!Комбинация Ctrl-Alt-7 на цифровой клавиатуре
CtrlAltPad8	EQUATE(0668H)	!Комбинация Ctrl-Alt-8 на цифровой клавиатуре
CtrlAltPad9	EQUATE(0669H)	!Комбинация Ctrl-Alt-9 на цифровой клавиатуре
CtrlAltMouseLeft	EQUATE(0601H)	!Комбинация Ctrl-Alt-левая кнопка мыши
CtrlAltMouseRight	EQUATE(0602H)	!Комбинация Ctrl-Alt-правая кнопка мыши
CtrlAltMouseCenter	EQUATE(0603H)	!Комбинация Ctrl-Alt-средняя кнопка мыши
CtrlAltMouseLeft2	EQUATE(0605H)	!Комбинация Ctrl-Alt-двойной щелчок левой кнопкой мыши
CtrlAltMouseRight2	EQUATE(0606H)	!Комбинация Ctrl-Alt-двойной щелчок правой кнопкой мыши
CtrlAltMouseCenter2	EQUATE(0607H)	!Комбинация Ctrl-Alt-двойной щелчок средней кнопкой мыши



## **Приложение С Присвоение значений свойствам**

### **Присвоение значений свойствам структур данных**

Для атрибутов (свойств) многих из структур данных типа APPLICATION, WINDOW и REPORT и составляющих их полей и структур допустимо задать при объявлении только константные значения (не переменные). То же справедливо и в отношении структур данных типа FILE, VIEW и QUEUE. Это кажется ограничением, однако значение большинства из этих константных свойств можно узнать или изменить, используя простые операторы присваивания, содержащие имена свойств.

Имена свойств представляют атрибуты (свойства) и параметры атрибутов, объявленные в структурах APPLICATION, WINDOW, REPORT, FILE, VIEW и QUEUE и их компонентах. Большинство атрибутов имеют соответствующее имя свойства. Однако некоторые атрибуты (такие как OVER и TREAD) представляют собой в действительности директивы компилятора, которые не имеют соответствующих имен свойств. Кроме того, существуют имена свойств, которые не соответствуют никаким объявленным атрибутам (необъявляемые свойства).

Имена свойств можно использовать в операторах присваивания в качестве назначения. Таким способом изменяется значение атрибута (или параметра атрибута), связанного со свойством. Имя свойства также можно использовать в любом строковом выражении, для того чтобы определить текущее значение атрибута (или параметра).

### **Встроенные переменные**

---

В библиотеке исполняемой системы Clarion for Windows есть три встроенные переменные: TARGET, PRINTER, и SYSTEM. Они используются в синтаксических конструкциях присвоения значений свойствам только для идентификации свойства, которому присваивается значение.

TARGET обычно указывает окно, которое в данный момент имеет фокус ввода. К тому же, оно может указывать на окно в другом исполняемом процессе или на печатаемый в данный момент документ, позволяя изменять значения свойств объектов окна в другом исполняемом процессе, или динамически изменять объекты документа при печати. Для изменения значения переменной TARGET используется процедура SETTARGET.

PRINTER ссылается на свойства относящиеся к принтеру, используемые следующим открываемым документом (и всеми последующими отчетами). Эта переменная используется только со свойствами, относящимися к принтеру.

SYSTEM представляет собой встроенную переменную, которая указывает глобальные

свойства, используемые всем приложением в целом. Существует несколько особых необъявляемых свойств, которые использует переменную SYSTEM для установки или опроса глобальных относительно приложения свойств.

## Имена свойств

[назначение] [\$] [объект] { свойство[,элемент] }

<i>назначение</i>	Метка структуры APPLICATION, WINDOW, REPORT, VIEW, или FILE, метка большого двоичного объекта (BLOB) или одной из встроенных переменных TARGET, PRINTER, или SYSTEM. Если назначение опущено, то подразумевается текущее значение переменной TARGET.
\$	Обязательный разделитель, когда и назначение, и объект одновременно указаны. Если назначение или объект опущен, то можно опустить и разделитель.
<i>объект</i>	Номер поля или метка соответствия экранного объекта в структуре-назначении, на который распространяется действие выражения. Если объект опущен, то должно быть указано назначение. Если назначение представляет собой структуру FILE, метку BLOB, встроенную переменную PRINTER или SYSTEM, то объект должен быть опущен.
<i>свойство</i>	Целочисленная константа, мнемоническое имя, заданное оператором EQATE, или переменная, которая указывает свойство(атрибут), подлежащий изменению. Это также может быть строка, если производится ссылка на свойство объекта .VBX .
<i>элемент</i>	Целочисленная константа или переменная, которая указывает, какой элемент свойства (для тех атрибутов, которые являются массивом и имеют несколько значений) подлежит изменению.

Такой синтаксис имени свойства позволяет осуществить обращение к атрибутам структур APPLICATION, WINDOW или REPORT, или любого объекта внутри этих структур. Для того, чтобы задать некий атрибут структуры APPLICATION, WINDOW или REPORT (структуры, а не составляющих ее объектов) опустите часть объект в имени свойства. А чтобы задать объект в текущем окне, опустите в имени свойства назначение.

Структура данных REPORT никогда не является назначением по умолчанию. Поэтому, для того, чтобы изменить используемое по умолчанию назначение, нужно использовать или оператор SETTARGET, или перед тем, как изменять какой-либо свойство структуры REPORT, или какого-либо содержащегося в ней объекта, явно указать в качестве назначения метку структуры REPORT.

Имя свойства может использоваться в операторах языка Clarion везде, где допустимо использование строкового выражения, или как назначение в простом операторе присваивания. Следовательно, назначение нового значения свойству является



присваиванием со свойством в качестве назначения и новым значением в качестве источника. Определение текущего значения свойства является присваиванием, в котором свойство служит источником, а назначением - переменная, в которую заносится текущее значение свойства. Выражение представляющее свойство может использоваться и само по себе, когда свойство представляет метод принадлежащий объекту VBX, OLE, или OCX.

Все свойства рассматриваются во время выполнения программы как строковые данные; компилятор автоматически подключает все необходимые преобразования данных. Все атрибуты, не имеющие параметров, являются двоичными. Двоичный атрибут или “указан”, или “опущен”. При проверке двоичного атрибута возвращается значение ‘1’ для присутствующего атрибута и значение ‘ ‘ (пустая строка) для опущенного атрибута. Изменение значения двоичного атрибута на ‘ ‘ (пустая строка), ‘0’ (ноль) или любое нечисловое строковое значение устанавливает, что этот атрибут опущен. Изменение значения двоичного свойства на любое другое устанавливает, что он присутствует.

Большинство свойств можно и проверить (прочитать), и изменить (записать в него значение). Однако, некоторые свойства доступны только для чтения и не могут быть изменены. Присвоение значения такому свойству просто не влияет на него. Для других свойств возможно только присвоение им значений, а чтении для них бессмысленно. Некоторые свойства являются массивами. Синтаксис обращения к конкретному элементу массива использует в качестве разделителя между свойством и номером элемента запятую (а не квадратные скобки).

### Пример:

```
MainWin  APPLICATION('My Application'),SYSTEM,MAX,ICON('Mylcon.ICO'),STATUS,RESIZE
        MENUBAR
        MENU('File'),USE(?FileMenu)
        ITEM('Open...'),USE(?OpenFile)
        ITEM('Close'),USE(?CloseFile),DISABLE
        ITEM('E&xit'),USE(?MainExit)
        END
        MENU('Help'),USE(?HelpMenu)
        ITEM('Contents'),USE(?HelpContents),STD(STD:HelpIndex)
        ITEM('Search for Help On...'),USE(?HelpSearch),STD(STD:HelpSearch)
        ITEM('How to Use Help'),USE(?HelpOnHelp),STD(STD:HelpOnHelp)
        ITEM('About MyApp...'),USE(?HelpAbout)
        END
        END
        TOOLBAR
        BUTTON('Open'),USE(?OpenButton),ICON(ICON:Open)
        END
        END
        CODE
        OPEN(MainWin)
```

```

MainWin{PROP:text} = 'A New Title'      !Изменить заголовок окна
?OpenButton{PROP:icon} = ICON:Asterisk !Изменить пиктограмму на кнопке
?OpenButton{PROP:at,1} = 5             !Изменить положение кнопки по горизонтали
?OpenButton{PROP:at,2} = 5             !Изменить положение кнопки по вертикали
IF MainWin$?HelpContents{PROP:std} <> STD:HelpIndex
  MainWin$?HelpContents{PROP:std} = STD:HelpIndex
END
MainWin{PROP:maximize} = 1 !Раскрыть на весь экран
ACCEPT
CASE ACCEPTED()                       !Какое поле выбрано
OF ?OpenFile                          !Выбран пункт меню Open
OROF ?OpenButton                      !Или кнопка Open на панели инструментов
  START(OpenFileProc)                 !Начать новую исполняемый процесс
OF ?MainExit                          !Выбран пункт меню Exit
OROF ?MainExitButton                  !Или кнопка Exit на панели инструментов
  BREAK                               !Break ACCEPT loop
OF ?HelpAbout                         !Выбран пункт меню About
  HelpAboutProc                       !Обратиться к процедуре описания приложения
END
END
CLOSE(MainWin)                        !Закрыть окно APPLICATION
RETURN

```

**Смотри также:**    SETTARGET

## **Мнемонические имена свойств атрибутов**

Операторы EQUATE, задающие мнемонические имена свойств, содержатся в файле PRORERTY.CWL. Кроме того, в этом файле содержатся мнемонические имена стандартных значений, используемых для некоторых из этих свойств. Некоторые свойства допускают только чтение и их значения нельзя изменить, другие допускают только запись в них значения, нельзя определить их текущее значение. Для каждого поля или объекта, которого касаются такие ограничения, они оговорены отдельно.

Каждое из приведенных далее свойств имеет отношение к какому-либо атрибуту (или его параметру) окна, отчета или объекта. Атрибут, на который указывает свойство, описывается в соответствующем разделе, и следует найти его объяснение, чтобы более подробно выяснить влияние на окно или объект, которое он производит.

В описании некоторых свойств есть фраза: (“ Атрибут-переключатель. Присвоение пустой строки (”) выключает его, а любое другое значение - включает”. Это означает, что данный атрибут или активен окна, отчета или объекта управления, или неактивен. Опрос свойства дает в результате пустую строку, когда атрибут не активен. Присвоение пустой строки такому атрибуту выключает его, а присвоение любого другого значения - включает.

PROP:Text Параметр заголовок окна APPLICATION, WINDOW, Или экранного объекта. Может содержать любое значение, допустимое в качестве параметра для экранного объекта. Например, ?Image{PROP:Text} = 'My.BMP' выведет в поле типа IMAGE новую картинку.

Остальные свойства атрибутов (в алфавитном порядке):

PROP:Absolute Атрибут ABSOLUTE (" если опущен, иначе - указан).  
 PROP:Type Содержит тип экранного объекта. Допустимые для присвоения ему значения представлены мнемоническими именами CREATE:xxxx (перечисленными в файле EQUATES.CLW) (свойства можно только читать).

PROP:Alone Атрибут ALONE (" если опущен, иначе - указан).  
 PROP:Alrt Атрибут ALRT. Массив.  
 PROP:Angle Атрибут ANGLE (если нет, то пробел).  
 PROP:At Атрибут AT. Массив (4 значения).  
 PROP:Xpos Параметр AT(x), эквивалентен {PROP:At,1}  
 PROP:Ypos Параметр AT(y), эквивалентен {PROP:At,2}  
 PROP:Width Параметр AT(„ширина), эквивалентен {PROP:At,3}  
 PROP:Height Параметр AT(„высота), эквивалентен {PROP:At,4}  
 PROP:Auto Атрибут AUTO(" если опущен, иначе - указан).  
 PROP:Autosize Атрибут AUTOSIZE. Атрибут-переключатель. Присвоение пустой строки (") выключает его, а любое другое значение - включает. (WRITE ONLY)

PROP:Ave Атрибут AVE (" если опущен, иначе - указан).  
 PROP:Bevel Атрибут BEVEL.Массив.  
 PROP:BevelOuter BEVEL(outer,) параметр, эквивалентно {PROP:Bevel,1}.  
 PROP:BevelInner BEVEL(inner,) параметр, эквивалентно {PROP:Bevel,2}.  
 PROP:BevelStyle BEVEL(„style) параметр, эквивалентно {PROP:Bevel,3}.  
 PROP:Boxed Атрибут ABSOLUTE (" если опущен, иначе - указан).  
 PROP:Cap Атрибут ABSOLUTE (" если опущен, иначе - указан).  
 PROP:Center Атрибут CENTER (" если опущен, иначе - указан).  
 PROP:CenterOffset Параметр CENTER(отступ), эквивалентен {PROP:Center,2}.  
 PROP:Check Атрибут CHECK, (" если опущен, иначе - указан).  
 PROP:Class Атрибут CLASS. Массив (2 значения).  
 PROP:VbxFile Параметр CLASS(файл), эквивалентен {PROP:Class,1}.  
 PROP:VbxName Параметр CLASS(имя), эквивалентен {PROP:Class,2}.  
 PROP:Clip Атрибут CLIP. Атрибут-переключатель. Присвоение пустой строки (") выключает его, а любое другое значение - включает. (WRITE ONLY)

PROP:Cnt Атрибут CNT (" если опущен, иначе - указан).  
 PROP:Color Атрибут COLOR (COLOR:none если цвет не указан).  
 PROP:Compatibility Атрибут COMPATIBILITY. (WRITE ONLY)  
 PROP:Create Атрибут CREATE (если нет, то пробел).(WRITE ONLY)

PROP:Background	COLOR(фон,,) параметр, эквивалентно {PROP:Color,1} (COLOR:none если отсутствует).
PROP:SelectedColor	COLOR(,выбранный_передний,) параметр, эквивалентно {PROP:Color,2} (COLOR:none, если отсутствует).
PROP:SelectedFillColor	COLOR(,выбранный_фон) параметр, эквивалентно {PROP:Color,3} (COLOR:none, если отсутствует).
PROP:Column	Атрибут COLUMN (0 = выключено, иначе номер выделенного в данный момент столбца).
PROP:Cursor	Атрибут CURSOR (" если опущен, иначе - указан).
PROP:Decimal	Атрибут DECIMAL (" если опущен, иначе - указан).
PROP:DecimalOffset	Параметр DECIMAL(смещение), эквивалентен {PROP:Decimal,2}.
PROP:Default	Атрибут DEFAULT, (" если опущен, иначе - указан).
PROP:Disable	Атрибут DISABLE, (" если опущен, иначе - указан).
PROP:Document	Атрибут DOCUMENT (если нет, то пробел). (WRITE ONLY)
PROP:Double	Атрибут DOUBLE (" если опущен, иначе - указан).
PROP:Down	Атрибут DOWN. Атрибут-переключатель. Присвоение пустой строки (") выключает его, а любое другое значение - включает.
PROP:Dragid	Атрибут DRAGID. Массив.
PROP:Drop	Атрибут DROP (0 если нет). Нельзя изменить значение на 0, или нулевое значение на ненулевое.
PROP:Dropid	Атрибут DROPID. Массив.
PROP:Fill	Атрибут FILL (если нет, то COLOR:none ).
PROP:First	Атрибут FIRST (" если опущен, иначе - указан).
PROP:Font	Атрибут FONT. Массив (4 значения).
PROP:FontName	Параметр FONT(начертание), эквивалентен {PROP:Font,1}.
PROP:FontSize	Параметр FONT(,размер), эквивалентен {PROP:Font,2}.
PROP:FontColor	Параметр FONT(,цвет), эквивалентен {PROP:Font,3}.
PROP:FontStyle	Параметр FONT(,стиль), эквивалентен {PROP:Font,4}.
PROP:Format	Атрибут FORMAT (" если опущен, иначе - указан). Это свойство обновляется всякий раз, когда пользователь во время работы программы изменяет формат окна списка.
PROP:From	Атрибут FROM (очередь, поле из очереди, или строка). (ТОЛЬКО ЗАПИСЬ ЗНАЧЕНИЯ)
PROP:Full	Атрибут FULL, (" если опущен, иначе - указан).
PROP:Gray	Атрибут GRAY, (" если опущен, иначе - указан).
PROP:Grid	Атрибут GRID (если нет, то пробел).
PROP:Hide	Атрибут HIDE, (" если опущен, иначе - указан).
PROP:Hlp	Атрибут HLP (если нет, то пробел).
PROP:Hscroll	Атрибут HSCROLL, (" если опущен, иначе - указан).
PROP:Icon	Атрибут ICON, (если нет, то пробел).
PROP:Iconize	Атрибут ICONIZE, (" если опущен, иначе - указан).
PROP:Imm	Атрибут IMM, (" если опущен, иначе - указан).
PROP:Ins	Атрибут INS (" если опущен, иначе - указан).

PROP:Join	Атрибут JOIN. Атрибут-переключатель. Присвоение пустой строки (") выключает его, а любое другое значение - включает.
PROP:Key	Атрибут KEY (если нет, то пробел).
PROP:Landscape	Атрибут LANDSCAPE, (" если опущен, иначе - указан).
PROP:Last	Атрибут LAST (" если опущен, иначе - указан).
PROP:Left	Атрибут LEFT (" если опущен, иначе - указан).
PROP:LeftOffset	Параметр LEFT(отступ), эквивалентен {PROP:Left,2}.
PROP:Link	Атрибут LINK (если нет, то пробел). (WRITE ONLY)
PROP:Mark	Атрибут MARK (очередь или поле из очереди). (ТОЛЬКО ЗАПИСЬ ЗНАЧЕНИЯ)
PROP:Mask	Атрибут MASK, (" если опущен, иначе - указан).
PROP:Max	Атрибут MAX (" если опущен, иначе - указан).
PROP:Maximize	Атрибут MAXIMIZE, (" если опущен, иначе - указан).
PROP:Mdi	Атрибут MDI (" если опущен, иначе - указан). (ТОЛЬКО ДЛЯ ЧТЕНИЯ)
PROP:Meta	Атрибут META, (" если опущен, иначе - указан).
PROP:Min	Атрибут MIN (" если опущен, иначе - указан).
PROP:Mm	Атрибут MM (" если опущен, иначе - указан).
PROP:Modal	Атрибут MODAL (" если опущен, иначе - указан). (ТОЛЬКО ДЛЯ ЧТЕНИЯ)
PROP:Msg	Атрибут MSG (если нет, то пробел).
PROP:NoBar	Атрибут NOBAR, (" если опущен, иначе - указан).
PROP:NoFrame	Атрибут NOFRAME (" если опущен, иначе - указан).
PROP:NoMerge	Атрибут NOMERGE, (" если опущен, иначе - указан).
PROP:NoSheet	Атрибут NOSHEET. Атрибут-переключатель. Присвоение пустой строки (") выключает его, а любое другое значение - включает.
PROP:Open	Атрибут OPEN (если нет, то пробел).(WRITE ONLY)
PROP:Ovr	Атрибут OVR (" если опущен, иначе - указан).
PROP:Page	Атрибут PAGE (" если опущен, иначе - указан).
PROP:PageAfter	Атрибут PAGEAFTER, (" если опущен, иначе - указан).
PROP:PageAfterNum	Параметр PAGEAFTER(), эквивалентен {PROP:PageAfter,2}.
PROP:PageBefore	Атрибут PAGEBEFORE, (" если опущен, иначе - указан).
PROP:PageBeforeNum	Параметр PAGEBEFORE(), эквивалентен {PROP:PageBefore,2}.
PROP:Pageno	Атрибут PAGENO (" если опущен, иначе - указан).
PROP:Palette	Атрибут PALETTE. Единичное значение.
PROP:Password	Атрибут PASSWORD, (" если опущен, иначе - указан).
PROP:Points	Атрибут POINTS (" если опущен, иначе - указан).
PROP:Preview	Атрибут PREVIEW (очередь или поле из очереди). (ТОЛЬКО ЗАПИСЬ ЗНАЧЕНИЯ)
PROP:Range	Атрибут RANGE. Массив (2 элемента).
PROP:RangeHigh	Параметр RANGE(начало), эквивалентен {PROP:Range,2}.
PROP:RangeLow	Параметр RANGE(конец), эквивалентен {PROP:Range,1}.

PROP:ReadOnly	Параметр READONLY, (" если опущен, иначе - указан).
PROP:Req	Атрибут REQ, (" если опущен, иначе - указан).
PROP:Reset	Атрибут RESET (0 = выключено, иначе номер уровня вложенности проверки значения полей)
PROP:Resize	Атрибут RESIZE (" если опущен, иначе - указан).
PROP:Right	Атрибут RIGHT (" если опущен, иначе - указан ).
PROP:RightOffset	Параметр RIGHT(отступ), эквивалентен {PROP:Right,2}.
PROP:Round	Атрибут ROUND, (" если опущен, иначе - указан).
PROP:Scroll	Атрибут SCROLL, (" если опущен, иначе - указан).
PROP:Separate	Атрибут SEPARATE, (" если опущен, иначе - указан).
PROP:Single	Атрибут SINGLE. Атрибут-переключатель. Присвоение пустой строки (") выключает его, а любое другое значение - включает.
PROP:Skip	Атрибут SKIP, (" если опущен, иначе - указан).
PROP:Spread	Атрибут SPREAD (" если опущен, иначе - указан).
PROP:Status	Атрибут STATUS. Массив (заканчивается нулем).
PROP:StatusText	Текст на линейке состояния. Массив (заканчивается нулем).
PROP:Std	Атрибут STD (" если опущен, иначе - указан).
PROP:Step	Атрибут STEP (" если опущен, иначе - указан).
PROP:Stretch	Атрибут STRETCH. Атрибут-переключатель. Присвоение пустой строки (") выключает его, а любое другое значение - включает. (WRITE ONLY)
PROP:Sum	Атрибут SUM (" если опущен, иначе - указан).
PROP:System	Атрибут SYSTEM, (" если опущен, иначе - указан).
PROP:Tally	Атрибут TALLY (если нет, то пробел).
PROP:Thous	Атрибут THOUS (" если опущен, иначе - указан).
PROP:Timer	Атрибут TIMER (0 - если нет).
PROP:Toolbox	Атрибут TOOLBOX (" если опущен, иначе - указан).
PROP:ToolTip	Атрибут TIP (" если опущен, иначе - указан).
PROP:Trn	Атрибут TRN, (" если опущен, иначе - указан).
PROP:Up	Атрибут UP. Атрибут-переключатель. Присвоение пустой строки (") выключает его, а любое другое значение - включает.
PROP:Upr	Атрибут UPR, (" если опущен, иначе - указан).
PROP:Use	Атрибут USE (имя переменной). Присвоение значения этому свойству изменяет значение USE-переменной. При чтении свойства, считывается значение USE-переменной.
PROP:Value	Атрибут VALUE (" если опущен, иначе - указан).
PROP:TrueValue	Параметр VALUE(истина), эквивалентно {PROP:Value,1}.
PROP:FalseValue	Параметр VALUE(ложь), эквивалентно {PROP:Value,2}.
PROP:Vcr	Атрибут VCR (" если опущен, иначе - указан).
PROP:VcrFreq	Параметр VCR(), эквивалентен {PROP:Vcr,2}.
PROP:Vscroll	Атрибут VSCROLL, (" если опущен, иначе - указан).
PROP:WithNext	Атрибут WITHNEXT (0 если нет).
PROP:WithPrior	Атрибут WITHPRIOR (0 если нет).

PROP:Wizard           Атрибут WIZARD (" если опущен, иначе - указан).  
 PROP:Zoom           Атрибут ZOOM. Атрибут-переключатель. Присвоение пустой строки  
 (") выключает его, а любое другое значение - включает. (WRITE ONLY)

### Пример:

```
CheckField STRING(1)
Screen       WINDOW
          ENTRY(@N3),USE(Ctl:Code)
          ENTRY(@S30),USE(Ctl:Name),REQ
          CHECK('True or False'),USE(CheckField)
          IMAGE('SomePic.BMP'),USE(?Image)
          BUTTON('OK'),USE(?OkButton),KEY(EnterKey)
          BUTTON('Cancel'),USE(?CanxButton),KEY(EscKey)
END
CODE
OPEN(Screen)
Screen{PROP:At,1} = 0       !Поместить окно в верхний левый угол
Screen{PROP:At,2} = 0
Screen{PROP:Gray} = 1      !Придать окну рельефность
Screen{PROP:Status,1} = -1 !Создать строку состояния из двух частей
Screen{PROP:Status,2} = 180
Screen{PROP:Status,3} = 0       !Завершить массив строки состояния
Screen{PROP:StatusText,2} = FORMAT(TODAY(),@D2)
!Поместить дату во вторую часть строки состояния
?CtlCode{PROP:Alrt,1} = F10Key
?CtlCode{PROP:Text} = '@N4'
?Image{PROP:Text} = 'MyPic.BMP'       !Изменить имя файла пиктограммы
?OkButton{PROP:Default} = '1'   !Установить атрибут DEFAULT кнопки ОК
?MyButton{PROP:Icon} = 'C:\Windows\MORICONS.DLL[10]'
!вывести 10-ю пиктограмму в MORICONS.DLL
?CheckField{PROP:TrueValue} = 'T'
?CheckField{PROP:FalseValue} = 'F'
ACCEPT
END
```

## Свойства строки форматирования окна списка

При помощи имен свойств можно также изменить свойства отдельных полей в многоколоночном окне списка или комбинированном окне списка. Каждое из этих свойств связано с одним элементом строки-параметра атрибута FORMAT. Эти свойства исключают необходимость в создании новой строки форматирования только для того, чтобы изменить одно свойство отдельного поля в окне списка.

Все эти свойства представляют собой массивы, для которых, чтобы указать на какое поле окна списка или поля COMBO это свойство влияет, следом за мнемоническим именем свойства нужно явно указать номер элемента массива, отделив его запятой

PROPLIST:Center С означает центрирование, (пробел если опущен, 1 если указан).

PROPLIST:CenterOffset Целое число, которое задает отступ, (пробел если опущен, 1 если указан).

PROPLIST:Color Звездочка (\*) означает, что информация о цвете поля содержится в четырех полях типа LONG, следующих в очереди (или строке указанной атрибутом FROM) сразу полем данных (пробел, если опускается, 1 - если указан).

PROPLIST:Decimal D - означает выравнивание по десятичной точке, (пробел если опущен, 1 если указан).

PROPLIST:DecimalOffset Целое число, которое задает отступ, (пробел если опущен, 1 если указан).

PROPLIST:Fixed F, который указывает, что поле фиксируется по левой границе списка (пробел если опущен, 1 если указан).

PROPLIST:Header ~Заголовок~ - текст заголовка для группы или поля (пробел если опущен, 1 если указан).

PROPLIST:HeaderCenter С означает центрирование заголовка (пробел если опущен, 1 если указан).

PROPLIST:HeaderCenterOffset Целое число, задающее отступ (пробел если опущено, 1 если указано).

PROPLIST:HeaderDecimal D - означает выравнивание заголовка по десятичной точке, (пробел если опущен, 1 если указан).

PROPLIST:HeaderDecimalOffset Целое число, которое задает отступ, (пробел если опущен, 1 если указан).

PROPLIST:HeaderLeft L - означает выравнивание заголовка влево, (пробел если опущен, 1 если указан).

PROPLIST:HeaderLeftOffset Целое число, которое задает отступ, (пробел если опущен, 1 если указан).

PROPLIST:HeaderRight R - означает выравнивание заголовка вправо, (пробел если опущен, 1 если указан).

PROPLIST:HeaderRightOffset Целое число, которое задает отступ, (пробел если опущен, 1 если указан).

PROPLIST:Icon I означает, что номер пиктограммы для поля содержится в поле типа LONG, которое в очереди (или строке, указанной атрибутом FROM) идет сразу за полем данных (пробел если опущен, 1 если указан).

PROPLIST:LastOnLine / (слэш), который означает, что следующее поле в группе выводится на следующей строке (пробел если опущен, 1 если указан).

PROPLIST:Left L означает выравнивание влево, (пробел если опущен, 1 если указан).

PROPLIST:LeftOffset Целое число, которое задает отступ слева, (пробел если опущен, 1 если



указан).

PROPLIST:Locator ? (знак вопроса), который обозначает поле для локатора (пробел если опущен, 1 если указан).

PROPLIST:Picture @Шаблон@ форматирования для поля (пробел если опущен, 1 если указан).

PROPLIST:Resize M, с помощью которого разрешается изменение размера поля или (пробел если опущен, 1 если указан).

PROPLIST:RightBorder Вертикальная черта (|), являющаяся разделителем справа от поля или группы (пробел если опущен, 1 если указан).

PROPLIST:Right R - означает правое выравнивание, (пробел если опущен, 1 если указан).

PROPLIST:RightOffset Целое число, которое задает отступ справа, (пробел если опущен, 1 если указан).

PROPLIST:Scroll S (целое число), помощью которого устанавливается линейка скроллинга для поля или группы. Указывает челую часть числа, (пробел если опущен, 1 если указан).

PROPLIST:Tree T означает в окне списка представляется древовидная структура (пробел если опущен, 1 если указан).

PROPLIST:TreeLines T(L) означает, что для древовидной структуры не выводятся линии соединяющие уровни (пробел если опущен, 1 если указан).

PROPLIST:TreeBoxes T(B) означает, что в древовидной структуре не выводятся квадратики, означающие развернутость или скрытость ветви (пробел если опущен, 1 если указан).

PROPLIST:TreeIndent T(I) означает, что в древовидной структуре не делается отступ для следующих уровней вложенности (что неявно предполагает подавление и соединяющий уровни линий и квадратики, означающие развернутость или скрытость ветви), (пробел если опущен, 1 если указан).

PROPLIST:Underline Символ подчеркивания (\_), который указывает подчеркивание поля или группы, (пробел если опущен, 1 если указан).

PROPLIST:Width Целое число, задающее ширину поля или группы.

Любое из этих свойств можно применить к группе полей, добавляя к свойству PROPLIST:Group.

PROPLIST:Group Для того, чтобы обратить действие свойства на группу полей, добавляется к свойству поля

### Пример:

```
?List{PROPLIST:Header,1} = 'First Field'    !Изменить текст заголовка первого поля
?List{PROPLIST:Header + PROPLIST:Group,1} = 'First Group'
!Изменить текст заголовка первого поля
```

**Смотри также:**    FORMAT

## Прочие свойства

### Свойства, связанные с действиями мышью в окне списка

С помощью следующих свойств можно получить положение курсора мыши внутри поля LIST или COMBO, в момент нажатия или отпускания кнопки. Им также можно присваивать значения, что впрочем не имеет никаких последствий кроме того, что временно изменяется значение, возвращаемое свойством при следующих опросах (при выполнении данной итерации цикла ACCEPT).

PROPLIST:MouseDownField Возвращает номер поля, на котором стоял курсор в момент нажатия кнопки мыши  
 PROPLIST:MouseDownRow Возвращает номер строки, на котором стоял курсор в момент нажатия кнопки мыши.  
 PROPLIST:MouseDownZone Возвращает номер зоны, на котором стоял курсор в момент нажатия кнопки мыши.  
 PROPLIST:MouseMoveField При перемещении курсора возвращает номер поля.  
 PROPLIST:MouseMoveRow При перемещении курсора возвращает номер строки.  
 PROPLIST:MouseMoveZone При перемещении курсора возвращает номер зоны.  
 PROPLIST:MouseUpField При отпускании кнопки мыши возвращает номер поля.  
 PROPLIST:MouseUpRow При отпускании кнопки мыши возвращает номер строки.  
 PROPLIST:MouseUpZone При отпускании кнопки мыши возвращает номер зоны.

Все три свойства "Row" возвращают 1 для строк заголовка и 2, если курсор стоит ниже последнего выведенного элемента.

В файле EQUATES.CW находятся мнемонические имена следующих зон:

LISTZONE:Field Поле в окне списка  
 LISTZONE:Right Зона изменения правой границы объекта  
 LISTZONE:Header Заголовок поля или группы  
 LISTZONE:ExpandBox Квадратик раскрытия древовидной структуры  
 LISTZONE:Tree Линия, соединяющая ветви древовидной структуры  
 LISTZONE:Icon Пиктограмма (в дереве или где либо еще)  
 LISTZONE:Nowhere Область отличная от вышеперечисленных

### Пример:

```
Que      QUEUE
F1       STRING(50)
F2       STRING(50)
F3       STRING(50)
         END
WinView  WINDOW('View'),AT(,340,200),SYSTEM,CENTER,ALRT(MouseLeft)
         LIST,AT(20,0,300,200),USE(?List),FROM(Que),IMM,HVSCROLL |
```

```

    FORMAT('80L~F1~80L~F2~80L~F3~'),IMM
END
CODE
OPEN(WinView)
DO BuildListQue
X# = 0
ACCEPT
CASE EVENT()
OF EVENT:AlertKey
  IF ?List{PROPLIST:MouseUpRow} = -1  !Проверить щелчок на заголовке
    CASE ?List{PROPLIST:MouseDownField} + X# !на каком заголовке
    OF 1
      SORT(Que,Que:F1)
      ?List{PROP:Format} = '80L~F1~#1#80L~F2~#2#80L~F3~#3#'
      X# = 0
    OF 2
      SORT(Que,Que:F2)
      ?List{PROP:Format} = '80L~F2~#2#80L~F3~#3#80L~F1~#1#'
      X# = 1
    OF 3
      SORT(Que,Que:F3)
      ?List{PROP:Format} = '80L~F3~#3#80L~F1~#1#80L~F2~#2#'
      X# = 2
    END
  DISPLAY
...
FREE(Que)

```

### Свойства, относящиеся к печати

Эти свойства относятся к отчетам и поведению принтера. Всех их можно использовать или со встроенной переменной PRINTER, или меткой структуры REPORT в качестве назначения, однако не все они влияют на оба эти назначения.

PROPPRINT:Collate Указывает, что принтер должен сортировать печатные листы: (копии “в подбор” или нет) 0=off, 1=on (поддерживается не всеми принтерами)

PROPPRINT:Color Флаг цветной или монохромной печати: 1=монохромная, 2=цветная (поддерживается не всеми принтерами)

PROPPRINT:Context Возвращает “хэндел” на контекст печатающего устройства, если уже выполнялся оператор PRINT, или на информационный контекст, если этот оператор еще не выполнялся. Это свойство не может применяться к глобальной встроенной переменной PRINTER и обычно только читается (а не устанавливается).

PROPPRINT:Copies Число копий, которое следует напечатать (поддерживается не всеми принтерами).

PROPPRINT:Device Имя принтера, так как оно появляется в диалоговом окне Windows

“Принтеры”. Если имена нескольких принтеров начинаются одинаково, то используется первый попавшийся с таким именем. Для встроенной переменной PRINTER можно устанавливать только тогда, когда отчет еще не открыт.

PROPPRINT:DevMode В инструментальной системе Windows Software Development Kit определена целая структура (devmode), определяющая режимы устройства. Это свойство обеспечивает прямое обращение через интерфейс прикладной программы (API) ко всем свойствам принтера. Прежде чем использовать этот способ, посмотрите руководство по Windows API.

DevMode GROUP

DeviceName STRING(32) !PROPPRINT:Device

SpecVersion USHORT

DriverVersion USHORT

Size USHORT

DriverExtra USHORT

Fields ULONG

Orientation SHORT

PaperSize SHORT !PROPPRINT:Paper

PaperLength SHORT !PROPPRINT:PaperHeight

PaperWidth SHORT !PROPPRINT:PaperWidth

Scale SHORT !PROPPRINT:Percent

Copies SHORT !PROPPRINT:Copies

DefaultSource SHORT !PROPPRINT:PaperBin

PrintQuality SHORT !PROPPRINT:Resolution

Color SHORT !PROPPRINT:Color

Duplex SHORT !PROPPRINT:Duplex

END

PROPPRINT:Driver Имя файл - драйвера принтера ( без расширения .DLL)

PROPPRINT:Duplex Режим двухсторонней печати (поддерживается не всеми принтерами). Мнемонические имена соответствия (DUPLEX::xxx) для стандартных значений перечисляются в файле PRNPROP.CLW

PROPPRINT:FontMode Режим фонтов TrueType. Мнемонические имена соответствия (FONTMODE::xxx) для стандартных значений перечисляются в файле PRNPROP.CLW

PROPPRINT:FromMin Будучи установленным для встроенной переменной PRINTER это свойство устанавливает, что печать должна начинаться со страницы заданной в поле “From:” (начальный номер страницы) в диалоговом окне PRINTERDIALOG. Указание -1 отменяет печать диапазона страниц.

PROPPRINT:FromPage Номер страницы, с которой начинается печать. Задание -1 означает печать с начала документа.

PROPPRINT:Paper Стандартный размер бумаги. Мнемонические имена соответствия

- (PAPER:xxx) для стандартных значений перечисляются в файле PRNPROP.CLW. Это свойство также определяет размер файлов .WMF, создаваемых “ядром печати” исполняемой библиотеки Clarion.
- PROPPRINT:PaperBin      Источник подачи бумаги. Мнемонические имена соответствия (PAPERBIN:xxx) для стандартных значений перечисляются в файле PRNPROP.CLW.
- PROPPRINT:PaperHeight    Длина листа бумаги в десятых долях миллиметра (мм/10). В дюйме 25,4 мм. Это свойство используется в тех случаях, когда для PROPPRINT:Paper установлено значение PAPER:Custom (для лазерных принтеров обычно не применяется).
- PROPPRINT:PaperWidth    Ширина листа бумаги в десятых долях миллиметра (мм/10). В дюйме 25,4 мм. Это свойство используется в тех случаях, когда для PROPPRINT:Paper установлено значение PAPER:Custom (для лазерных принтеров обычно не применяется).
- PROPPRINT:Percent    Коэффициент масштабирования, используемый для увеличения или уменьшения размеров печатаемого изображения, в процентах. По умолчанию 100%. Чтобы напечатать с желаемым изменением размеров, установите значение этого свойства (если ваши принтер и драйвер принтера Windows поддерживают масштабирование. Например, чтобы увеличить размеры вдвое, установите 200, а чтобы уменьшить в 2 раза - установите 50).
- PROPPRINT:Port      Логическое имя порта для вывода на печать ((LPT1, COM1, и т.д.).
- PROPPRINT:PrintToFile    Признак печати в файл: 0=нет, 1=да.
- PROPPRINT:PrintToName    Имя файла для печати в файл.
- PROPPRINT:Resolution    Разрешающая способность печати в точках на дюйм (Dots Per Inch - DPI). Мнемонические имена соответствия (RESOLUTION:xxx) для стандартных значений перечисляются в файле PRNPROP.CLW.
- PROPPRINT:ToMax    При установки для встроенной переменной PRINTER это свойство устанавливает, что печать должна продолжаться до страницы заданной в поле “To:” диалогового окна PRINTERDIALOG. Указание -1 отменяет печать диапазона страниц.
- PROPPRINT:ToPage    Номер страницы на которой прекращается печать. Указание -1 означает печать до конца документа.
- PROPPRINT:Yresolution    Разрешающая способность печати по вертикали в точках на дюйм (Dots Per Inch - DPI). Мнемонические имена соответствия (RESOLUTION:xxx) для стандартных значений перечисляются в файле PRNPROP.CLW.

**Пример:**

SomeReport

END  
CODE

REPORT

```

PRINTER{PROPPRINT:Device} = 'Epson' !Выбрать 1-й Epson в списке
PRINTER{PROPPRINT:Port} = 'LPT2:' !установить вывод на LPT2
SomeReport{PROPPRINT:Paper} = PAPER:User !Нестандартный размер бумаги
SomeReport{PROPPRINT:PAPERHeight} = 6 * 254 !6 дюймов высотой
SomeReport{PROPPRINT:PAPERWidth} = 3.5 * 254 !и 3.5 дюйма шириной
PRINTER{PROPPRINT:Percent} = 250 !размер печать увеличивается в 2.5 раза
PRINTER{PROPPRINT:Copies} = 3!печатать 3 копии каждой страницы
PRINTER{PROPPRINT:Collate} = False !печатать страницы 1,1,1,2,2,2,3,3,3,...
PRINTER{PROPPRINT:Collate} = True !печатать страницы 1,2,3..., 1,2,3...,
PRINTER{PROPPRINT:PrintToFile} = True !печатать в файл
PRINTER{PROPPRINT:PrintToName} = 'OUTPUT.RPT'!Имя файла, куда печатать
OPEN(SomeReport) !После установки свойств открыть отчет

```

## Необъявленные свойства

Эти свойства не имеют соответствия атрибутам структур данных. К ним можно обратиться только во время выполнения программы:

### **PROP:AcceptAll**

Возвращает 1, если включен режим “AcceptAll” (безостановочный режим), и 0 - если нет. Кроме того, это свойство можно использовать для переключения этого режима. Обычно безостановочный режим включает оператор SELECT без параметров. Это режим редактирования полей, в котором все объекты в окне обрабатываются в последовательности переключения клавишей TAB, путем генерации для каждого объекта события EVENT:Accepted. Это позволяет выполнить процедуры проверки данных для всех полей, включая те, которые пользователь не трогал. Когда встречается одно из следующих условий, безостановочный режим немедленно выключается:

SELECT(?)

Window{PROP:AcceptAll} = 0

Объект с атрибутом REQ имеет значение ноль или пробел.

Оператором SELECT(?) выбирается тот же самый объект, который пользователь редактировал. Обычно это означает, что в этом поле содержатся неверные данные и пользователь должен повторить ввод данных.

Оператор Window{PROP:AcceptAll} = 0 выключает безостановочный режим. Присвоение значения этому свойству можно использовать для включения/выключения режима AcceptAll.

Когда объект с атрибутом REQ имеет значение ноль или пробел, безостановочный режим выключается и выделяется поле, в котором пользователь должен повторить ввод (безо всяких дополнительных нажатий клавиши TAB).

Когда успешно обработаны все поля, для окна генерируется событие EVENT:Completed.

**Пример:**

```

Screen    WINDOW,PRE(Scr)
          ENTRY(@N3),USE(Ctl:Code)
          ENTRY(@S30),USE(Ctl:Name),REQ
          BUTTON('OK'),USE(?OkButton),KEY(EnterKey)
          BUTTON('Cancel'),USE(?CanxButton),KEY(EscKey)
          END
          CODE
          OPEN(Screen)
          ACCEPT
          IF EVENT() = EVENT:Completed THEN BREAK.    !Выключить режим AcceptAll
          CASE ACCEPTED()
          OF ?Ctl:Code
            IF Ctl:Code > 150                        !Если введены неверные данные
              BEEP                                    ! оповестить пользователя и
              SELECT(?)                               ! ввести данные вновь
            END
          OF ?OkButton
            Screen{PROP:AcceptAll} = 1 !Включить режим AcceptAll
          ..                                         !Конец цикла ACCEPT и структуры CASE ACCEPTED

```

**PROP:Active**

Возвращает 1, если данное окно активно, и пробел, если нет. Для того чтобы сделать это верхнее в данном исполняемом процессе окно активным.

**Пример:**

```

CODE
OPEN(ThisWindow)
X# = START(AnotherThread)    !Начать другой процесс
ACCEPT
CASE EVENT()
OF EVENT:LoseFocus           !Когда данное окно теряет фокус
  IF Y# <> X#                 ! проверить: это первое переключение фокуса ?
    ThisWindpw{PROP:Active} = 1 !и вернуть фокус на данный процесс
    Y# = X#                  ! затем пометить, что переключение фокуса было
  ...

```

**PROP:AlwaysDrop**

Когда это свойство установлено в 0, то выпадающая часть окна списка или комбинированного окна с атрибутом DROP раскрывается, только когда пользователь щелкнет на пиктограмме, означающей раскрытие списка, а когда пользователь нажимает клавишу стрелка вниз, высвечиваемые элементы списка прокручиваются не вызывая

раскрытия списка. Если же значение этого свойства отлично от нуля, то выпадающая часть списка раскрывается или по нажатию стрелки вниз, или по щелчку на пиктограмме раскрытия.

### Пример:

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           COMBO(@S20),AT(0,0,20,220),USE(MyCombo),FROM(Que),DROP(5)
           END
           CODE
           OPEN(MDIChild)
           ?MyCombo{PROP:AlwaysDrop} = 0           !Set windows-like drop behavior
```

### **PROP:AppInstance**

Возвращает идентификационный номер экземпляра (HInstance) .EXE файла для того, чтобы использовать в обращениях на низком уровне интерфейса прикладного программирования, где это требуется. Используется только со встроенной переменной SYSTEM (только для чтения)

### Пример:

```
HInstance  PROGRAM
           LONG
           CODE
           OPEN(AppFrame)
           HInstance = SYSTEM{PROP:AppInstance}
           !Получить идентификационный номер экземпляра
           ! .EXE файла для дальнейшего использования
           ACCEPT
           END
```

### **PROP:AutoPaper**

Устанавливает и возвращает состояние механизма выбора наиболее подходящей бумаги. Состояние, используемое по умолчанию, - механизм включен. Это свойство применяется только к переменной SYSTEM.

### Пример:

```
PROGRAM
CODE
OPEN(AppFrame)
SYSTEM{PROP:AutoPaper} = ''
!Выключить механизма выбора наиболее подходящей бумаги
```



```
ACCEPT
END
```

### **PROP:ChoiceFeg**

Возвращает или устанавливает номер выбранного в данный момент поля TAB в структуре SHEET или кнопки RADIO в структуре OPTION.

#### **Пример:**

```
WinView    WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           OPTION('Option 1'),USE(OptVar1),KEY(F10Key),HLP('Option1Help')
           RADIO('Radio 1'),AT(0,0,20,20),USE(?R1)
           RADIO('Radio 2'),AT(20,0,20,20),USE(?R2)
           END
           END
           CODE
           OPEN(WinView)
           ?OptVar1{PROP:ChoiceFeg} = ?R1           !Выбрать первую кнопку
           ACCEPT
           END
```

### **PROP:ClientHandle**

Возвращает идентификационный номер клиента в окне (области окна, которая содержит объекты) для того, чтобы использовать в обращениях на низком уровне интерфейса прикладного программирования, где это требуется. Используется только для чтения.

#### **Пример:**

```
WinView    WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           END
MessageText                                CSTRING('You cannot exit the program from this
window ')
MessageCaption                            CSTRING('No EVENT:CloseDown Allowed ')
TextAddr    LONG
CaptionAddr                                LONG
RetVal    SHORT
           CODE
           OPEN(WinView)
           ACCEPT
           CASE EVENT()
           OF EVENT:CloseDown
               TextAddress = ADDRESS(MessageText)
               CaptionAddress = ADDRESS(MessageCaption)
               RetVal =
MessageBox(WinView{PROP:ClientHandle},TextAddr,CaptionAddr,MB_OK)
```

```

!Обращение к функции интерфейса прикладного программирования
!Windows с использованием идентификационный номер клиента
    CYCLE!Запретить завершение программы из данного окна
END
END

```

### **PROP:ClientWndProc**

Устанавливает или возвращает процедуру сообщения клиентской части окна для того, чтобы использовать в обращениях на низком уровне интерфейса прикладного программирования, где это требуется. Обычно используется с подклассами для отслеживания всех сообщений в среде Windows.

#### **Пример:**

```

PROGRAM
MAP
    main
        SubClassFunc(USHORT,SHORT,SHORT,LONG),LONG,PASCAL
        MODULE('Windows')          !Библиотека Win31 TopSpeed
        CallWindowProc(LONG,USHORT,SHORT,SHORT,LONG),LONG,PASCAL
    END
END
SavedProc LONG
PT        GROUP,PRE(PT)
X         SHORT
Y         SHORT
END
CODE
Main

Main      PROCEDURE
WinView   WINDOW('View'),AT(0,0,320,200),HVSCROLL,MAX,TIMER(1)
    STRING('X Pos'),AT(1,1,,),USE(?String1)
    STRING(@n3),AT(24,1,,),USE(PT:X)
    STRING('Y Pos'),AT(44,1,,),USE(?String2)
    STRING(@n3),AT(68,1,,),USE(PT:Y)
    BUTTON('Close'),AT(240,180,60,20),USE(?Close)
END
CODE
OPEN(WinView)
SavedProc = WinView{PROP:ClientWndProc}      !Запомнить эту процедуру
WinView{PROP:ClientWndProc} = ADDRESS(SubClassFunc)
!Изменить на процедуру подкласса
ACCEPT
    CASE ACCEPTED()
    OF ?Close

```

```

        BREAK
    END
END

```

```

SubClassFunc  FUNCTION(hWnd,wMsg,wParam,lParam)      !Процедура подкласса
WM_MOUSEMOVE  EQUATE(0200H) ! для отслеживания перемещения мыши
    CODE                                           ! в клиентской части окна
    CASE wMsg
    OF WM_MOUSEMOVE
        PT:X = MOUSEX()
        PT:Y = MOUSEY()
    END
    RETURN(CallWindowProc(SavedProc,hWnd,wMsg,wParam,lParam))
                                           !Передать управление
                                           ! запомненной процедуре

```

### **PROP:ClipBits**

Свойство поля IMAGE, которое позволяет записать битовую карту изображения во внутренний буфер обмена (но не из него) Windows, если это свойство установлено в 1. Во внутреннем буфере обмена могут сохраняться в виде битовой карты только изображения типа .BMP, .PCX, и .GIF

### **Пример:**

```

WinView  WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
        IMAGE(),AT(0,0,,),USE(?Image)
        BUTTON('Save Picture'),AT(80,180,60,20),USE(?SavePic)
        BUTTON('New Picture'),AT(160,180,60,20),USE(?NewPic)
    END
FileName  STRING(64)                !Переменная для указания имени файла
    CODE
    OPEN(WinView)
    DISABLE(?LastPic)
    IF NOT FILEDIALOG('Choose File to View',FileName,'BitMap|*.BMP|PCX|*.PCX',0)
    RETURN                            !Выйти, если файл не выбран
    END
    ?Image{PROP:Text} = FileName
    ACCEPT
    CASE ACCEPTED()
    OF ?NewPic
        IF NOT FILEDIALOG('Choose File to
View',FileName,'BitMap|*.BMP|PCX|*.PCX',0)
            BREAK                    ! Выйти, если файл не выбран
        END
        ?Image{PROP:Text} = FileName
        OF ?SavePic

```

```

?Image{PROP:ClipBits} = 1 !Поместить изображение в буфер обмена
ENABLE(?LastPic)         ! активизировать кнопку Last Picture
END
END

```

### **PROP:ConnectionString**

Свойство файла, использующего драйвер ODBC, который возвращает строку соединения (обычно хранящуюся в атрибуте OWNER), которая позволила бы завершить соединение. Если атрибут OWNER содержит только имя источника данных, то появляется окно подключения, чтобы запросить у пользователя остальную необходимую информацию перед выполнением соединения. Это окно подключения раскрывается каждый раз, когда происходит подключение. С помощью этого свойства разработчик может один раз ввести данные для окна соединения, затем установить атрибут OWNER таким образом, чтобы данные вводились из PROP:ConnectionString, исключая явное подключение.

#### **Пример:**

```

OwnerString STRING(20)
Customer  FILE,DRIVER('ODBC'),OWNER(OwnerString)
Record    RECORD
Name      STRING(20)
...
CODE
OwnerString = 'DataSourceName'
OPEN(Customer)
OwnerString = Customer{PROP:ConnectionString} !Взять всю строку подключения
MESSAGE(OwnerString)      !Отобразить ее

```

### **PROP:DDETimeOut**

Свойство встроенной переменной SYSTEM, с помощью которого можно установить или прочесть величину интервала времени, используемую при обмене данными для всех команд. Эта величина исчисляется в сотых долях секунды и по умолчанию равна 500.

#### **Пример:**

```

DDERetVal STRING(20)
WinOne     WINDOW,AT(0,0,160,400)
           ENTRY(@s20),USE(DDERetVal)
           END
MyServer   LONG
           CODE
           OPEN(WinOne)
           SYSTEM{PROP:DDETimeOut} = 1000      !Установить интервал в 10 секунд
           MyServer = DDESERVER('MyApp','DataEntered') !Открыть как сервер DDE
           ACCEPT

```

```

CASE EVENT()
OF EVENT:DDErequest      !Данные запрошены один раз
  DDEWRITE(MyServer,DDE:manual,'DataEntered',DDERetVal)
  !Передать данные один раз
END
END

```

### **PROP:DeferMove**

Свойство встроенной переменной SYSTEM, с помощью которого изменение размеров и/или перемещение экранного объекта откладывается до тех пор, пока не будет завершен цикл ACCEPT или свойство SYSTEM{PROP:DeferMove} не будет установлено в 0. Тем самым отменяется немедленное присвоение значений свойствам, связанным с расположением и размерами и все изменения размеров выполняются библиотечной процедурой за раз (исключая возможные временные наложения объектов друг на друга). Абсолютная величина числа, присваиваемого свойству SYSTEM{PROP:DeferMove} определяет число откладываемых изменений размеров, для которых заранее выделяется память (этот объем автоматически расширяется, но менее эффективно и может привести к аварии). Присвоение положительного числа означает автоматический сброс этого свойства в 0 при выполнении следующего цикла ACCEPT, в то время как присвоенное отрицательное число продолжает действовать до тех пор пока свойство явно не будет установлено в 0.

### **Пример:**

```

WinView  WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
          IMAGE(),AT(0,0,,),USE(?Image)
          BUTTON('New Picture'),AT(160,180,60,20),USE(?NewPic)
          BUTTON('Close'),AT(80,180,60,20),USE(?Close)
          END
FileName  STRING(64)                !Переменная содержащая имя файла
ImageWidth SHORT
ImageHeight SHORT
          CODE
          OPEN(WinView)
          DISABLE(?LastPic)
          IF NOT FILEDIALOG('Choose File to View',FileName,'BitMap|*.BMP|PCX|*.PCX',0)
            RETURN                  !Выйти, если не выбрано никакого файла
          END
          ?Image{PROP:Text} = FileName
          ACCEPT
          CASE ACCEPTED()
          OF ?NewPic
            IF NOT FILEDIALOG('Choose File to
View',FileName,'BitMap|*.BMP|PCX|*.PCX',0)
              BREAK                ! Выйти, если не выбрано никакого файла
            END

```

```

?Image{PROP:Text} = FileName
SYSTEM{PROP:DeferMove} = 4 !Отложить перемещение и изменение размеров
ImageWidth = ?Image{PROP:Width}
ImageHeight = ?Image{PROP:Height}
IF ImageWidth > 320
    ?Image{PROP:Width} = 320
    ?Image{PROP:XPos} = 0
ELSE
    ?Image{PROP:XPos} = (320 - ImageWidth) / 2 !Центрировать по горизонтали
END
IF ImageHeight > 180
    ?Image{PROP:Height} = 180
    ?Image{PROP:YPos} = 0
ELSE
    ?Image{PROP:YPos} = (180 - ImageHeight) / 2 ! Центрировать по вертикали
END
OF ?Close
BREAK
..
!Перемещение и изменение размеров произойдет по окончании цикла ACCEPT

```

### **PROP:DropWidth**

Устанавливает или возвращает ширину выпадающей части списка COMBO или LIST с атрибутом DROP. Ширина измеряется в условных единицах (если не установлено свойство PROP:Pixels).

#### **Пример:**

```

MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
COMBO(@S20),AT(0,0,20,220),USE(MyCombo),FROM(Que),DROP(5)
END
CODE
OPEN(MDIChild)
?MyCombo{PROP:DropWidth} = 48          !Set widthto 48 dialog units

```

### **PROP:Edit**

Задаёт метку соответствия объекта для того, чтобы выполнить редактирование “по месту” в колонке окна списка. Это свойство является массивом, в котором каждый элемент представляет номер колонки, в которой выполняется редактирование. Когда элемент не равен нулю, то поле редактирования не скрыто и способно перемещаться и изменять размеры поверх текущей строки в колонке, что даёт возможность пользователю вводить данные. Чтобы скрыть поле редактирования, присвойте свойству ноль (0).

#### **Пример:**

```

Q QUEUE
f1 STRING(15)
f2 STRING(15)
END
Win1 WINDOW('List Edit In Place'),AT(0,1,308,172),SYSTEM

LIST,AT(6,6,120,90),USE(?List),COLUMN,FORMAT('60L@s15@60L@s15@'),FROM(Q),IMM
END
?EditEntry EQUATE(100)
CODE
OPEN(Win1)
CREATE(?EditEntry,CREATE:Entry)
ACCEPT
CASE FIELD()
OF ?List
CASE EVENT()
OF EVENT:NewSelection
IF ?List{PROP:edit,?List{PROP:column}}
GET(Q,CHOICE())
END
OF EVENT:Accepted
IF KEYCODE() = MouseLeft2
GET(Q,CHOICE())
?EditEntry{PROP:text} = ?List{PROPLIST:picture,?List{PROP:column}}
CASE ?List{PROP:column}
OF 1
?EditEntry{PROP:use} = F1
OF 2
?EditEntry{PROP:use} = F2
END
?List{PROP:edit,?List{PROP:column}} = ?EditEntry
..
OF ?EditEntry
CASE EVENT()
OF EVENT:Selected
?EditEntry{PROP:Touched} = 1
OF EVENT:Accepted
PUT(Q)
?List{PROP:edit,?List{PROP:column}} = 0
...

```

**PROP:Enabled**

Возвращает пустую строку, если объект недоступен или из-за того, что он сам

деактивирован оператором DISABLE, или он входит в состав деактивированного объекта (OPTION, GROUP, MENU, SHEET, или TAB). Это свойство можно только опрашивать.

### Пример:

```
MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
SHEET,AT(0,0,320,175),USE(SelectedTab)
TAB('Tab One'),USE(?TabOne)
PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P1)
ENTRY(@S8),AT(100,140,32,20),USE(E1)
PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P2)
ENTRY(@S8),AT(100,240,32,20),USE(E2)
END
TAB('Tab Two'),USE(?TabTwo)
PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P3)
ENTRY(@S8),AT(100,140,32,20),USE(E3)
PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P4)
ENTRY(@S8),AT(100,240,32,20),USE(E4)
END
END
BUTTON('Ok'),AT(100,180,20,20),USE(?Ok)
BUTTON('Cancel'),AT(200,180,20,20),USE(?Cancel)
END
CODE
OPEN(MDIChild)
ACCEPT
CASE EVENT()
OF EVENT:Completed
BREAK
END
CASE FIELD()
OF ?Ok
CASE EVENT()
OF EVENT:Accepted
SELECT
END
OF ?E3
CASE EVENT()
OF EVENT:Accepted
IF ?E3{PROP:Enabled} AND MDIChild{PROP:AcceptAll}
!Проверить видимость объекта во время безостановочного режима
E3 = UPPER(E3) !Преобразовать введенные данные на верхний регистр
DISPLAY(?E3) ! и вывести данные большими буквами
END
END
OF ?Cancel
CASE EVENT()
```



```

OF EVENT:Accepted
BREAK
END
END
END

```

### **PROP:ExeVersion**

Свойство, относящееся к встроенной переменной SYSTEM, которое возвращает номер версии EXE-файла, созданного системой Clarion for Windows. Это номер версии Clarion for Windows, в которой компилировался данный EXE-файла, даже если библиотека исполняющей системы от более нового релиза (см. свойство PROP:LibVersion). Это свойство впервые появилось в Clarion for Windows релиз 1501, поэтому для EXE модулей более старых версий оно возвращает пробел (READ-ONLY).

#### **Пример:**

```
MESSAGE('Compiled in CW release ' & SYSTEM{PROP:ExeVersion})
```

### **PROP:FlushPreview**

Выводит метафайлы, в которых в соответствии с атрибутом PREVIEW временно хранится отчет, на принтер (0 = выключено, иначе - включено, при открытии отчета всегда устанавливается в 0).

#### **Пример:**

SomeReport	PROCEDURE
WMFQue	QUEUE
	!Очередь, содержащая имена метафайлов
	STRING(64)
	END
NextEntry	BYTE(1)
	!Переменная - счетчик элементов очереди
Report	REPORT,PREVIEW(WMFQue)
	!Отчет с атрибутом PREVIEW
DetailOne	DETAIL
	!Объекты отчета
	END
	END
ViewReport	WINDOW('View Report'),AT(0,0,320,200),MDI,MAX,HVSCROLL
	IMAGE(),AT(0,0,320,180),USE(?ImageField)
	BUTTON('View Next Page'),AT(0,180,60,20),USE(?NextPage),DEFAULT
	BUTTON('Print Report'),AT(80,180,60,20),USE(?PrintReport)
	BUTTON('Exit Without Printing'),AT(160,180,60,20),USE(?ExitReport)
	END
	CODE
	OPEN(Report)
	SET(SomeFile)
	!Операторы по созданию отчета
	LOOP

```

NEXT(SomeFile)
IF ERRORCODE() THEN BREAK.
PRINT(DetailOne)
END
ENDPAGE(Report)
OPEN(ViewReport)           !Открыть окно просмотра отчета
GET(WMFQue,NextEntry)      !Get first queue entry
?ImageField{PROP:text} = WMFQue      !Загрузить первую страницу отчета
ACCEPT
CASE ACCEPTED()
OF ?NextPage
    NextEntry += 1          !Нарастить счетчик
    IF NextEntry > RECORDS(WMFQue) THEN CYCLE.
                                !Проверить, не конец ли отчета
    GET(WMFQue,NextEntry)   !Взять следующий элемент очереди
    ?ImageField{PROP:text} = WMFQue      !Загрузить следующую страницу
    DISPLAY                 ! И вывести ее
OF ?PrintReport
    Report{PROP:FlushPreview} = 1 !Вывести файлы на принтер
    BREAK                  ! и выйти из процедуры
OF ?ExitReport
    BREAK                 !Выйти из процедуры
END
END
RETURN                    ! Вернуться в вызвавшую процедуру,
                        ! автоматически закрыв окно и отчет,
                        ! освободив память очереди, и автоматически
                        ! удалив временные метафайлы

```

### **PROP:Follows**

Изменяет порядок выбора полей клавишей TAB в рамках старшего объекта, в котором они находятся. Данный объект следует за объектом, номер которого задается в этом свойстве. Это свойство должно указывать существующий объект в пределах старшего объекта (WINDOW, OPTION, GROUP). Этому свойству можно только устанавливать значение, но не считывать его.

### **Пример:**

```

WinView  WINDOW('View Report'),AT(0,0,320,200),MDI,MAX,HVSCROLL
          BUTTON('View Next Page'),AT(0,180,60,20),USE(?NextPage),DEFAULT
          BUTTON('Print Report'),AT(80,180,60,20),USE(?PrintReport)
          BUTTON('Exit Without Printing'),AT(160,180,60,20),USE(?ExitReport)

END

CODE
OPEN(WinView)
    !Обычно за кнопкой Просмотр отчета следует кнопка Печать отчета

```

```
?PrintReport{PROP:Follows} = ?ExitReport
!Теперь кнопка Печать отчета следует за кнопкой Выход
ACCEPT
END
```

### **PROP:Handle**

Возвращает идентификатор окна или объекта для использования в низкоуровневых обращениях к функциям интерфейса прикладного программирования Windows, которые требуют указания этого идентификатора.

#### **Пример:**

```
WinView    WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           END
MessageText                                CSTRING('You cannot exit the program from this
window ')
MessageCaption                             CSTRING('No EVENT:CloseDown Allowed ')
TextAddressLONG
CaptionAddress                             LONG
RetVal    SHORT
           CODE
           OPEN(WinView)
           ACCEPT
           CASE EVENT()
           OF EVENT:CloseDown
             TextAddress = ADDRESS(MessageText)
             CaptionAddress = ADDRESS(MessageCaption)
             RetVal =
MessageBox(WinView{PROP:Handle},TextAddress,CaptionAddress,MB_OK)
           !В обращении к функции Windows API используется идентификатор окна
           CYCLE    !Запретить завершение программы из этого окна
           END
           END
```

### **PROP:HeaderHeight**

Возвращает высоту заголовка объекта управления LIST или COMBO. Ширина измеряется в условных единицах (если не установлено свойство PROP:Pixels). (READ-ONLY)

#### **Пример:**

```
MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           LIST,AT(0,0,220,220),USE('?L1'),FROM(Que),IMM,FORMAT('60L~Header Text~')
           END
           CODE
```

```

OPEN(MDICHild)
X# = ?L1{PROP:HeaderHeight}
!Получить высоту заголовка в условных единицах

```

### **PROP:HscrollPos**

Возвращает положение “бегунка” на горизонтальной линейке скроллинга (от 0 до 255) в окне, полях IMAGE, TEXT, LIST и COMBO, имеющих атрибут HSCROLL. Установка этого свойства приводит к тому, что происходит прокрутка содержимого окна или объекта в горизонтальном направлении.

### **Пример:**

```

Que      QUEUE
F1       STRING(50)
F2       STRING(50)
F3       STRING(50)
END
WinView  WINDOW('View'),AT(,340,200),SYSTEM,CENTER
          LIST,AT(20,0,300,200),USE(?List),FROM(Que),IMM,HVSCROLL |
          FORMAT('80L#1#80L#2#80L#3#')
END
CODE
OPEN(WinView)
DO BuildListQue
ACCEPT
CASE FIELD()
OF ?List
CASE EVENT()
OF EVENT:ScrollDrag
CASE ?List{PROP:HscrollPos} % 200) + 1
OF 1
?List{PROP:Format} = '80L#1#80L#2#80L#3#'
OF 2
?List{PROP:Format} = '80L#2#80L#3#80L#1#'
OF 3
?List{PROP:Format} = '80L#3#80L#1#80L#2#'
END
DISPLAY
...
FREE(Que)

BuildListQue ROUTINE
LOOP 15 TIMES
Que.F1 = 'F1F1F1F1'
Que.F2 = 'F2F2F2F2'
Que.F3 = 'F3F3F3F3'

```

```

ADD(Que)
END

```

### **PROP:IconList**

Массив, в котором устанавливаются пиктограммы, выводимые в окне списка, отформатированном для вывода пиктограмм (обычно для иерархических структур).

#### **Пример:**

```

PROGRAM
MAP
  RandomAlphaData(*STRING)
END
TreeDemo QUEUE,PREF()      !Очередь, данные из которой выводятся в окне
списка
FName      STRING(20)
ColorNFG   LONG            !Цвет переднего плана для нормального поля
FName
ColorNBG   LONG            ! Цвет фона для нормального поля FName
ColorSFG   LONG            ! Цвет переднего плана для выбранного поля
FName
ColorSBG   LONG            ! Цвет фона для выбранного поля FName
IconField  LONG            !Номер пиктограммы для поля FName
TreeLeve   LONG            !Уровень вложенности
LName      STRING(20)
Init       STRING(4)
END
Win        WINDOW('List Boxes'),AT(0,0,366,181),SYSTEM,DOUBLE
          LIST,AT(0,34,366,146),FROM(TreeDemo),USE(?Show),HVSCROLL, |
          FORMAT('80L*IT~First Name~*80L~Last Name~16C~Initials~')
END
CODE
LOOP 20 TIMES
  RandomAlphaData(FName)
  ColorNFG = COLOR:White    !Назначить цвета для поля FNAME
  ColorNBG = COLOR:Maroon
  ColorSFG = COLOR:Yellow
  ColorSBG = COLOR:Blue
  IconField = ((x#-1) % 4) + 1 !Назначить номер пиктограммы
  TreeLevel = ((x#-1) % 4) + 1 !Назначить уровень вложенности
  RandomAlphaData(LName)
  RandomAlphaData(Init)
  ADD(TD)
END
OPEN(Win)
?Show{PROP:iconlist,1} = ICON:VCRback ! Пиктограмма 1 = <

```



```

END
?Image{PROP:Text} = FileName
OF ?SavePic
  Fil:MyMemo = ?Image{PROP:ImageBits} !Поместить изображение в мемо-поле
  ADD(SomeFile)                      ! и сохранить его в файле на диске
  ENABLE(?LastPic)                   ! активизировать кнопку Last Picture
OF ?LastPic
  ?Image{PROP:ImageBits} = Fil:MyMemo
                                   !Вывести последнее сохраненное мемо-поле

END
END

```

### **PROP:ImageBlob**

Свойство поля IMAGE, который позволяет изображение, отображаемое в поле перемещать в/из поле типа BLOB. Любое отображаемое в экранном поле изображение может храниться поле BLOB.

#### **Пример:**

```

WinView  WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
          IMAGE(),AT(0,0,,),USE(?Image)
          BUTTON('Save Picture'),AT(80,180,60,20),USE(?SavePic)
          BUTTON('New Picture'),AT(160,180,60,20),USE(?NewPic)
          BUTTON('Last Picture'),AT(240,180,60,20),USE(?LastPic)
          END

SomeFile  FILE,DRIVER('TopSpeed'),PRE(Fil)          !Файл с полем BLOB
MyBlob    BLOB,BINARY
Rec       RECORD
F1        LONG

..
FileName  STRING(64)                                !Переменная, в которой хранится имя файла
          CODE
          OPEN(SomeFile)
          OPEN(WinView)
          DISABLE(?LastPic)
          IF NOT FILEDIALOG('Choose File to View',FileName,'BitMap|*.BMP|PCX|*.PCX',0)
            RETURN                                     !Выйти, если не выбрано никакого файла
          END
          ?Image{PROP:Text} = FileName
          ACCEPT
          CASE ACCEPTED()
          OF ?NewPic
            IF NOT FILEDIALOG('Choose File to
View',FileName,'BitMap|*.BMP|PCX|*.PCX',0)
              BREAK                                     ! Выйти, если не выбрано никакого файла

```

```

END
?Image{PROP:Text} = FileName
OF ?SavePic
  Fil:MyBlob{PROP:Handle} = ?Image{PROP:ImageBlob}
!Поместить изображение в BLOB
  ADD(SomeFile)           ! и сохранить его на диске
  ENABLE(?LastPic)       ! активизировать кнопку Last Picture
OF ?LastPic
  ?Image{PROP:ImageBlob} = Fil:MyBlob{PROP:Handle}
!Вывести последнее сохраненное в BLOB изображение
END
END

```

### **PROP:Items**

Возвращает число элементов списка отображаемых в окне списка или комбинированном окне списка, (ТОЛЬКО ДЛЯ ЧТЕНИЯ)

#### **Пример:**

```

Que      QUEUE
          STRING(30)
          END

WinView  WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL,SYSTEM
          LIST,AT(6,6,120,90),USE(?List),FORMAT('120L'),FROM(Q),IMM
          END

CODE
OPEN(WinView)
SET(SomeFile)
LOOP ?List{PROP:Items} TIMES
!Заполнить очередь до предельного числа отображаемых элементов
  NEXT(SomeFile)
  Que = Fil:Record
  ADD(Que)
END
ACCEPT
END

```

### **PROP:LazyDisplay**

Выключает (если установлено в 1) и включает (если установлено в 0, что делается по умолчанию) возможность полной перерисовки окна перед выполнением оператора, следующего за оператором DISPLAY. PROP:LazyDisplay = 1 устанавливает ощутимо более быструю работу с видеоизображением, поскольку перерисовка окна происходит по окончании цикла ACCEPT, если отсутствуют ожидающие сообщения. Эта установка может существенно увеличить производительность некоторых приложений, но может также



отрицательно влиять на его внешний вид.

### Пример:

```
WinView  APPLICATION('MyApp'),AT(0,0,320,200),MAX,HVSCROLL,SYSTEM
          END

          CODE
          OPEN(WinView)
          SYSTEM{PROP:LazyDisplay} = 1  !Запретить дополнительные перерисовки окон
                                         ! на протяжении всего приложения
          ACCEPT
          END
```

### **PROP:LibVersion**

Свойство встроенной переменной SYSTEM, которое возвращает номер версии библиотеки исполняемой системы Clarion for Windows (.DLL) загруженной в данный момент для исполняемого в данный момент EXE. модуля. Это не одно и тоже, что версия Clarion for Windows, в которой компилировался данный EXE. модуль (см. PROP:ExeVersion). Это свойство впервые появилось в Clarion for Windows релиз 1501, поэтому для EXE модулей более старых версий PROP:ExeVersion возвращает пробел (READ-ONLY).

### Пример:

```
MESSAGE('Runtime DLL from release ' & SYSTEM{PROP:LibVersion})
```

### **PROP:Line**

Массив, каждый элемент которого содержат одну строку текста из поля типа TEXT. (Только для чтения)

### **PROP:LineCount**

Возвращает число строк в поле TEXT. (Только для чтения)

### Пример:

```
LineCount  SHORT
MemoLine   STRING(80)

CustRpt    REPORT,AT(1000,1000,6500,9000),THOUS
Detail1     DETAIL,AT(0,0,6500,6000)
            TEXT,AT(0,0,6500,6000),USE(Fil:MemoField)
            END
Detail2     DETAIL,AT(0,0,6500,125)
            STRING(@s80),AT(0,0,6500,125),USE(MemoLine)
```

```

END
END
CODE
OPEN(File)
SET(File)
OPEN(CustRpt)
LOOP
NEXT(File)
  LineCount = CustRpt$?Fil:MemoField{PROP:LineCount}
  LOOP X# = 1 TO LineCount
    MemoLine = CustRpt$?Fil:MemoField{PROP:Line,X#}
    PRINT(Detail2)
  END
END
END

```

### **PROP:LineHeight**

Устанавливает и возвращает высоту строки объекта управления LIST или COMBO. Ширина измеряется в условных единицах (если не установлено свойство PROP:Pixels). Для объекта TEXT оно возвращает высоту знакоместа шрифта для данного объекта (расстояние от верха одной строки текста до верха следующей строки) в установленных на данный момент единицах измерения. Для объектов типа TEXT это свойство READ-ONLY

#### **Пример:**

```

MDIChild  WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
           LIST,AT(0,0,220,220),USE(?L1),FROM(Que),IMM
           END
           CODE
           OPEN(MDIChild)
           ?L1{PROP:LineHeight} = 8      !Установить высоту в 8 условных единиц

```

### **PROP:MaxHeight**

Устанавливает или возвращает максимально возможную высоту окна.

### **PROP:MaxWidth**

Устанавливает или возвращает максимально возможную ширину окна.

### **PROP:MinHeight**

Устанавливает или возвращает минимально возможную высоту окна.

### **PROP:MinWidth**

Устанавливает или возвращает минимально возможную ширину окна.

#### **Пример:**

```

WinView  WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL,SYSTEM,RESIZE
          LIST,AT(6,6,120,90),USE(?List),FORMAT('120L'),FROM(Q),IMM
          END
          CODE
          OPEN(WinView)
WinView{PROPMaxHeight} = 200  !Установить границы, за которые пользователь
не может
WinView{PROPMaxWidth} = 320   !расширять окно
WinView{PROPMinHeight} = 90
WinView{PROPMinWidth} = 120
          ACCEPT
          END

```

**PROP:NoTips**      Выключает (если установлена 1) или включает (если 0) подсказку на панели инструментов (атрибут TIP) для системы, окна или поля в окне.

**Пример:**

```

WinView  APPLICATION('MyApp'),AT(0,0,320,200),MAX,HVSCROLL,SYSTEM
          END
          CODE
          OPEN(WinView)
          SYSTEM{PROP:NoTips} = 1  !Выключить вывод инструментальной подсказки на
протяжении
          ACCEPT                    ! всего приложения
          END

```

**PROP:Pixels**

Переключает единицы измерения линейных размеров между условными единицами и пикселями (не относится к печатным документам). После установки этого свойства все свойства и функции позиционирования (такие как GETPOSITION, SETPOSITION, PROP:Xpos, PROP:Ypos, PROP:Width, and PROP:Height) возвращают и воспринимают координаты в пикселях, а не в условных единицах.

**Пример:**

```

WinView  WINDOW('MyApp'),AT(0,0,320,200),MAX,HVSCROLL,SYSTEM
          END
          CODE
          OPEN(WinView)
WinView{PROP:Pixels} = 1        !Изменить единицы измерения на пиксели
          ACCEPT
          END

```

**PROP:Progress**

Можно непосредственно обновлять выводимый индикатор степени выполнения некого действия (поле типа PROGRESS) приваивая значение (в диапазоне, определенном атрибутом RANGE этому свойству).

**Пример:**

```

BackgroundProcess PROCEDURE      !Фоновая процедура
Win      WINDOW('Batch Processing...'),AT(,400,400),TIMER(1),MDI,CENTER
        PROGRESS,AT(100,140,200,20),USE(?ProgressBar),RANGE(0,200)
        BUTTON('Cancel'),AT(190,300,20,20),STD(STD:Close)
END
CODE
OPEN(Win)
OPEN(File)
?ProgressBar{PROP:rangehigh} = RECORDS(File)
SET(File)                      !Установить пакетную обработку
ACCEPT
CASE EVENT()
OF EVENT:CloseWindow
    BREAK
OF EVENT:Timer                  !Обрабатывать записи когда время позволяет
    ProgressVariable += 3      !Автоматическое обновление 1-го индикатора
    LOOP 3 TIMES
    NEXT(File)
IF ERRORCODE() THEN BREAK.
?ProgressBar{PROP:progress} += 1    !Ручное обновление индикатора
    !выполнение неких пакетных операций
...
CLOSE(File)

```

**PROP:ScreenText**

Возвращает текст выводимый на экране в заданном поле типа ENTRY или ему подобном поле (SPIN/COMBO).

**Пример:**

```

WinView  WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
        SPIN(@n3),AT(0,0,320,180),USE(Fil:Field),RANGE(0,255)
END
CODE
OPEN(WinView)
ACCEPT
CASE FIELD()
OF ?Fil:Field
    CASE EVENT()

```

```

OF EVENT:Rejected
  MESSAGE(?Fil:Field{PROP:ScreenText} & ' is not in the range 0-255')
  SELECT(?)
  CYCLE
END
END
END
END

```

### **PROP:SelStart / PROP:Selected**

Устанавливает или возвращает начальный символ (включительно) помечаемого блока в поле типа ENTRY или TEXT. Позиционирует курсор ввода данных слева от этого символа и устанавливает свойство PROP:SelEnd в ноль, чтобы обозначить, что блок еще не помечен. Оно также идентифицирует выделенный в данный момент элемент объекта LIST или COMBO (обычно записанное как PROP:Selected).

### **PROP:SelEnd**

Устанавливает или возвращает конечный символ (включительно) помечаемого блока в поле типа ENTRY или TEXT.

### **Пример:**

```

WinView  WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
          ENTRY(@S30),AT(0,0,320,180),USE(Fil:Field),ALRT(F10Key)
          LIST,AT(6,6,120,90),USE(?List),FORMAT('120L'),FROM(Q),IMM
          END
          CODE
          OPEN(WinView)
          ACCEPT
          CASE ACCEPTED()
          OF ?List
            GET(Q,?List{PROP:Selected})  !Взять выделенный элемент из очереди
          OF ?Fil:Field
            SETCLIPBOARD(Fil:Field[?Fil:Field{PROP:SelStart} : ?Fil:Field{PROP:SelEnd}])
          !Place highlighted string slice in Windows' clipboard
          END
          END

```

### **PROP:Size**

Возвращает или устанавливает размер поля типа BLOB

### **Пример:**

Names	FILE,DRIVER('TopSpeed')	
NbrKey	KEY(Names:Number)	
Notes	BLOB	!Может быть больше чем 64К

```

Rec      RECORD
Name     STRING(20)
Number   SHORT

..
BlobSize LONG
BlobBuffer1 STRING(65520),STATIC      ! Строка максимального размера
BlobBuffer2 STRING(65520),STATIC      ! Строка максимального размера
WinView  WINDOW('View BLOB Contents'),AT(0,0,320,200),SYSTEM
        TEXT,AT(0,0,320,180),USE(BlobBuffer1),VSCROLL
        TEXT,AT(0,190,320,180),USE(BlobBuffer2),VSCROLL,HIDE
END
CODE
OPEN(Names)
SET(Names)
NEXT(Names)
OPEN(WinView)
BlobSize = Names:Notes{PROP:Size} !Получить длину содержимого поля BLOB
IF BlobSize > 65520
    BlobBuffer1 = Names:Notes[1:65520]
    BlobBuffer2 = Names:Notes[65521:BlobSize]
    WinView{PROP:Height} = 400
    UNHIDE(?BlobBuffer2)
ELSE
    BlobBuffer1 = Names:Notes[1:BlobSize]
END
ACCEPT
END

```

### **PROP:TabRows**

Возвращает число строк объектов TAB в структуре SHEET. (READ-ONLY)

### **PROP:NumTabs**

Возвращает число объектов TAB в структуре SHEET. (READ-ONLY)

### **Пример:**

```

MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
        SHEET,AT(0,0,320,175),USE(SelectedTab)
        TAB('Tab One'),USE(?TabOne)
        OPTION('Option 1'),USE(OptVar1),KEY(F10Key),HLP('Option1Help')
        RADIO('Radio 1'),AT(20,0,20,20),USE(?R1)
        RADIO('Radio 2'),AT(40,0,20,20),USE(?R2)
END
OPTION('Option 2'),USE(OptVar2),MSG('Option 2')
RADIO('Radio 3'),AT(60,0,20,20),USE(?R3)
RADIO('Radio 4'),AT(80,0,20,20),USE(?R4)

```

```

END
END
TAB('Tab Two'),USE(?TabTwo)
PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P1)
ENTRY(@S8),AT(100,140,32,20),USE(E1)
PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P2)
ENTRY(@S8),AT(100,240,32,20),USE(E2)
END
TAB('Tab Three'),USE(?TabThree)
OPTION('Option 3'),USE(OptVar3)
RADIO('Radio 1'),AT(20,0,20,20),USE(?R5)
RADIO('Radio 2'),AT(40,0,20,20),USE(?R6)
END
OPTION('Option 4'),USE(OptVar4)
RADIO('Radio 3'),AT(60,0,20,20),USE(?R7)
RADIO('Radio 4'),AT(80,0,20,20),USE(?R8)
END
END
TAB('Tab Four'),USE(?TabFour)
PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P3)
ENTRY(@S8),AT(100,140,32,20),USE(E3)
PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P4)
ENTRY(@S8),AT(100,240,32,20),USE(E4)
END
END
BUTTON('Ok'),AT(100,180,20,20),USE(?Ok)
BUTTON('Cancel'),AT(200,180,20,20),USE(?Cancel)
END
CODE
OPEN(MDICHild)
MESSAGE('Number of TABs: ' & ?SelectedTab{PROP:NumTabs})
MESSAGE('Number of rows of TABs: ' & ?SelectedTab{PROP:TabRows})
ACCEPT
END

```

### **PROP:Thread**

Возвращает номер исполняемого процесса для окна. Если с помощью оператора SETTARGET установлена встроенная переменная TARGET, то для текущего исполняемого процесса этот номер не требуется.

### **Пример:**

```

WinView    WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL,SYSTEM
           END
ToolboxThread    BYTE
           CODE

```

```

OPEN(WinView)
ToolboxThread = ToolboxWin{PROP:Thread}
! номер исполняемого процесса для окна
ACCEPT
END

```

### **PROP:TipDelay**

Устанавливает длительность паузы перед выводом инструментальной подсказки (заданной атрибутом TIP) для переменной SYSTEM (только 16 разрядов).

### **PROP:TipDisplay**

Устанавливает длительность отображения инструментальной подсказки (заданной атрибутом TIP) для переменной SYSTEM (только 16 разрядов).

### **Пример:**

```

WinView  APPLICATION('MyApp'),AT(0,0,320,200),MAX,HVSCROLL,SYSTEM
          END
          CODE
          OPEN(WinView)
          SYSTEM{PROP:TipDelay} = 50 !Пауза перед выводом в 1/2 секунды
          SYSTEM{PROP:TipDisplay} = 500 !Время показа подсказки 5 секунд
          ACCEPT
          END

```

### **PROP:Touched**

Когда не равно нулю, это свойство означает, что со времени последнего события EVENT:Accepted данные в поле типа ENTRY, TEXT, SPIN, or COMBO были изменены пользователем. При генерации события EVENT:Accepted, свойство автоматически устанавливается в 0.

### **Пример:**

```

WinView  WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
          ENTRY(@S30),AT(0,0,320,180),USE(Fil:Field)
          END
          CODE
          OPEN(WinView)
          ACCEPT
          CASE EVENT()
          OF EVENT:Selected
            ?Fil:Field{PROP:Touched} = 1
          !Принудительно генерировать событие EVENT:Accepted
          OF EVENT:Accepted
            ! Обработать данные или введенные пользователем, или изначально находившиеся в поле

```



```
END
END
```

### **PROP:Type**

Содержит тип экранного объекта управления. Возможные значения в виде мнемонических имен CREATE:xxxx перечисляются в файле EQUATES.CLV. Свойство используется только для чтения.

#### **Пример:**

```
MyField    STRING(1)
?MyField   EQUATE(100)

WinView    WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
END
CODE
OPEN(WinView)
IF UserChoice = 'CheckField'
    CREATE(?MyField,CREATE:Check)
ELSE
    CREATE(?MyField,CREATE:Entry)
END
?MyField{PROP:Use} = MyField
SETPOSITION(?MyField,10,10)
IF ?MyField{PROP:Type} = CREATE:Check !Проверить тип объекта
    ?MyField{PROP:TrueValue} = 'T'
    ?MyField{PROP:FalseValue} = 'F'
END
ACCEPT
END
```

### **PROP:VBXEvent**

Возвращает имя события VBX. (Только для чтения)

### **PROP:VBXEventArg**

Параметры события VBX. Массив.

#### **Пример:**

```
WinView    WINDOW('View'),AT(0,0,320,200),MDI,MAX,HVSCROLL
            CUSTOM,USE(?Graph),CLASS('graph.vbx','graph'),'graphstyle'('2')
END
CODE
OPEN(WinView)
ACCEPT
```

```

CASE EVENT()
OF EVENT:VBXEvent
  IF ?Graph{PROP:VBXEvent} = 'FooEvent'      !Проверить название события
    ProcessFoo(?Graph{PROP:VBXEventArg,1},?Graph{PROP:VBXEventArg,2})
!Взять 1-й и 2-й параметры события и передать для обработки в процедуру
  END
END
END

```

### **PROP:Visible**

Возвращает пустую строку, если объект невидим на экране или из-за того, что он сам скрыт оператором HIDE, или он входит в состав скрытого объекта (OPTION, GROUP, MENU, SHEET, или TAB), или он является неактивной страницей TAB. Это свойство можно только опрашивать.

### **Пример:**

```

MDIChild WINDOW('Child One'),AT(0,0,320,200),MDI,MAX,HVSCROLL
SHEET,AT(0,0,320,175),USE(SelectedTab)
  TAB('Tab One'),USE(?TabOne)
    PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P1)
    ENTRY(@S8),AT(100,140,32,20),USE(E1)
    PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P2)
    ENTRY(@S8),AT(100,240,32,20),USE(E2)
  END
  TAB('Tab Two'),USE(?TabTwo)
    PROMPT('Enter Data:'),AT(100,100,20,20),USE(?P3)
    ENTRY(@S8),AT(100,140,32,20),USE(E3)
    PROMPT('Enter More Data:'),AT(100,200,20,20),USE(?P4)
    ENTRY(@S8),AT(100,240,32,20),USE(E4)
  END
END
BUTTON('Ok'),AT(100,180,20,20),USE(?Ok)
BUTTON('Cancel'),AT(200,180,20,20),USE(?Cancel)
END
CODE
OPEN(MDIChild)
ACCEPT
CASE EVENT()
OF EVENT:Completed
  BREAK
END
CASE FIELD()
OF ?Ok
  CASE EVENT()
OF EVENT:Accepted

```

```

SELECT
END
OF ?E3
CASE EVENT()
OF EVENT:Accepted
  E3 = UPPER(E3)  !Преобразовать введенные данные на верхний регистр
  IF ?E3{PROP:Visible} AND MDIChild{PROP:AcceptAll}
!Проверить видимость объекта во время безостановочного режима
    DISPLAY(?E3)      ! и вывести текст большими буквами
  END
END
OF ?Cancel
CASE EVENT()
OF EVENT:Accepted
  BREAK
END
END
END

```

### **PROP:VscrollPos**

Возвращает положение “бегунка” на вертикальной линейке скроллинга (от 0 до 255) в окне, полях IMAGE, TEXT, LIST и COMBO, имеющих атрибут VSCROLL. Установка этого свойства приводит к тому, что происходит прокрутка содержимого окна или объекта в вертикальном направлении (если окно списка или комбинированное окно списка не имеет атрибута IMM, то перемещается только “бегунок”).

### **Пример:**

```

Que      QUEUE
         STRING(50)
         END
WinView  WINDOW('View'),AT(0,0,320,200),MDI,SYSTEM
         LIST,AT(0,0,320,200),USE(?List),FROM(Que),IMM,VSCROLL
         END
         CODE
         OPEN(WinView)
         Fil:KeyField = 'A' ; DO BuildListQue
         ACCEPT
         CASE FIELD()
         OF ?List
         CASE EVENT()
         OF EVENT:ScrollDrag
           EXECUTE INT(?List{PROP:VscrollPos}/10) + 1
           Fil:KeyField = 'A'           ; Fil:KeyField = 'B'
           Fil:KeyField = 'C'           ; Fil:KeyField = 'D'
           Fil:KeyField = 'E'           ; Fil:KeyField = 'F'

```

```

Fil:KeyField = 'G'           ; Fil:KeyField = 'H'
Fil:KeyField = 'I'           ; Fil:KeyField = 'J'
Fil:KeyField = 'K'           ; Fil:KeyField = 'L'
Fil:KeyField = 'M'           ; Fil:KeyField = 'N'
Fil:KeyField = 'O'           ; Fil:KeyField = 'P'
Fil:KeyField = 'Q'           ; Fil:KeyField = 'R'
Fil:KeyField = 'S'           ; Fil:KeyField = 'T'
Fil:KeyField = 'U'           ; Fil:KeyField = 'V'
Fil:KeyField = 'W'           ; Fil:KeyField = 'X'
Fil:KeyField = 'Y'           ; Fil:KeyField = 'Z'
END
DO BuildListQue

```

```

...
FREE(Que)
BuildListQue ROUTINE
FREE(Queue)
SET(Fil:SomeKey,Fil:SomeKey) !Встать по выбранному значению ключа
LOOP ?List{PROP:Items} TIMES !Обработать нужное число записей
NEXT(SomeFile) ; IF ERRORCODE() THEN BREAK. !Прерваться по концу файла
Que = Fil:KeyField           !Присвоить значения полей файла полям очереди
ADD(Que)                     ! и добавить элемент в очередь
END

```

### **PROP:WndProc**

Установить или получить процедуру сообщения окну (не клиентской части) для того, чтобы использовать в обращениях на низком уровне интерфейса прикладного программирования, где это требуется. Обычно используется с подклассами для отслеживания всех сообщений в среде Windows.

#### **Пример:**

```

PROGRAM
MAP
Main PROCEDURE
SubClassFunc1
FUNCTION(USHORT,SHORT,SHORT,LONG),LONG,PASCAL
SubClassFunc2
FUNCTION(USHORT,SHORT,SHORT,LONG),LONG,PASCAL
MODULE('Windows') !TopSpeed Win31 Library
CallWindowProc
FUNCTION(LONG,USHORT,SHORT,SHORT,LONG),LONG,PASCAL
... !End MAP and MODULE
SavedProc1 LONG
SavedProc2 LONG
WM_MOUSEMOVE EQUATE(0200H)
PT GROUP

```

```

X      SHORT
Y      SHORT
      END
      CODE
      Main

Main   PROCEDURE
WinView WINDOW('View'),AT(0,0,320,200),HVSCROLL,MAX,TIMER(1),STATUS
      STRING('X Pos'),AT(1,1,,),USE(?String1)
      STRING(@n3),AT(24,1,,),USE(PT:X)
      STRING('Y Pos'),AT(44,1,,),USE(?String2)
      STRING(@n3),AT(68,1,,),USE(PT:Y)
      BUTTON('Close'),AT(240,180,60,20),USE(?Close)
      END
      CODE
      OPEN(WinView)
      SavedProc1 = WinView{PROP:WndProc}  !Запомнить эту процедуру
      WinView{PROP:WndProc} = ADDRESS(SubClassFunc1)  !Заменить на процедуру
подкласса
      SavedProc2 = WinView{PROP:ClientWndProc}  ! Запомнить эту процедуру
WinView{PROP:ClientWndProc} = ADDRESS(SubClassFunc2)  ! Заменить на
процедуру подкласса
      ACCEPT
      CASE ACCEPTED()
      OF ?Close
      BREAK
      ..
SubClassFunc1  FUNCTION(hWnd,wMsg,wParam,lParam)
!Процедура подкласса
      CODE  ! для отслеживания перемещений мыши только в
      IF wMsg = WM_MOUSEMOVE ! в строке состояния окна
      PT:X = MOUSEX()
      PT:Y = MOUSEY()
      END
      RETURN(CallWindowProc(SavedProc1,hWnd,wMsg,wParam,lParam))

SubClassFunc2  FUNCTION(hWnd,wMsg,wParam,lParam)  ! Процедура подкласса
      CODE  ! для отслеживания перемещений мыши только в
      IF wMsg = WM_MOUSEMOVE ! клиентской части окна
      PT:X = MOUSEX()
      PT:Y = MOUSEY()
      END
      RETURN(CallWindowProc(SavedProc2,hWnd,wMsg,wParam,lParam))
!Передать управление обратно
! в запомненную процедуру

```

## **Свойства структур *VIEW* и *FILE***

### **Свойства - атрибуты**

---

#### **PROP:Filter**

Устанавливает атрибут FILTER структуры VIEW.

#### **Пример:**

```
BRW1::View:Browse                                VIEW(Members)
      PROJECT(Mem:MemberCode,Mem:LastName,Mem:FirstName)
      END
KeyValue  STRING(20)

      CODE
      KeyValue = 'Smith'
      BIND('KeyValue',KeyValue)
      BIND('Mem:LastName',Mem:LastName)
      Mem:LastName = KeyValue
      SET(Mem:LastNameKey,Mem:LastNameKey)
      BRW1::View:Browse{PROP:Filter} = 'Mem:LastName = KeyValue'
      OPEN(BRW1::View:Browse)
```

#### **PROP:Order**

Устанавливает атрибут ORDER структуры VIEW.

#### **Пример:**

```
BRW1::View:Browse                                VIEW(Members)
      PROJECT(Mem:MemberCode,Mem:LastName,Mem:FirstName)
      END
KeyValue  STRING(20)

      CODE
      BIND(Mem:Record)
      BRW1::View:Browse{PROP:Order} = 'Mem:LastName,MEM:FirstName'
      OPEN(BRW1::View:Browse)
      SET(BRW1::View:Browse)
```

## Необъявляемые свойства структур VIEW и FILE

### PROP:LoginTimeOut

Свойство файла, использующего драйвер ODBC, позволяющее вам установить перерыв в подключении. Значение задается в секундах.

#### Пример:

```
OwnerString  STRING(20)
Customer  FILE,DRIVER('ODBC'),OWNER(OwnerString)
Record    RECORD
Name      STRING(20)
..
CODE
OwnerString = 'DataSourceName'
Customer{PROP:LoginTimeOut} = 30      !Установить в 30 секунд
OPEN(Customer)
OwnerString = Customer{PROP:ConnectString} !Получить полную строку
подключения
MESSAGE(OwnerString)
```

### PROP:JoinExpression

Свойство структуры VIEW, использующей вторую форму JOIN, которой позволяет звдвть связи между файлами. Это свойство представляет собой массив, у которого индекс соответствует порядковому номеру JOIN в структуре VIEW, выражение которого вы хотите опросить или изменить.

#### Пример:

```
ViewOrd  VIEW(Customer)      !Объявить структуру VIEW
JOIN(Header,'Cus:AcctNumber = Hea:AcctNumber AND ' & |
' (Hea:Discount + Hea:Total) * .1 > Hea:Discount')
PROJECT(Hea:AcctNumber,Hea:OrderNumber)
JOIN(Dtl:OrderKey,Hea:AcctNumber,Hea:OrderNumber)
PROJECT(Dtl:ItemDtl:Quantity)
JOIN(Pro:ItemKey,Dtl:Item)
PROJECT(Pro:Description,Pro:Price)
END
END
END
END
CODE
OPEN(ViewOrd)
ViewOrd{PROP:JoinExpression,1} = 'Cus:AcctNumber = Hea:AcctNumber AND ' & |
```

‘ (Hea:Discount + Hea:Total) \* .15 > Hea:Discount’)

!Изменить выражение первого JOIN на предмет проверки 15% скидки вместо 10%

### **PROP:OrderAllTables**

Свойство структуры VIEW, которое может понадобиться, если вы используете VIEW, которая объединяет несколько SQL таблиц. Это может быть только, когда используется SQL драйвер (такой как ODBC, AS/400, или Oracle ).

По умолчанию SQL файловые драйверы (ODBC, Oracle, ... ) включают компоненты ключа только первичного файла в предложение ORDER BY, посылаемое SQL серверу. Устанавливая свойство PROP:OrderAllTables в TRUE, мы говорим драйверу использовать в предложении ORDER BY наряду с компонентами ключа первичного файла, связывающие поля и компоненты ключа вторичных файлов.

При доступе к структуре VIEW файловый драйвер генерирует предложение SELECT подобное примеру, приведенному ниже.

```
SELECT CustNo,Name,Zip,OrderNo,OrderDate FROM Customer,Ord
      WHERE (Customer.CustNo = Ord.CustNo)
      ORDER BY CustNo,OrderNo
```

### **Пример:**

```
BRW1::View:Browse                                VIEW(Customer)
  PROJECT(CUST:CustNo)
  PROJECT(CUST:Name)
  PROJECT(CUST:Zip)
  PROJECT(CUST:CustNo)
  JOIN(ORD:ByCustomer,CUST:CustNo)
    PROJECT(ORD:OrderNo)
    PROJECT(ORD:OrderDate)
  END
END
CODE
?BRW1::View:Browse{PROP:OrderAllTables} = TRUE
```

## **Встроенный SQL**

---

### **PROP:SQL**

Синтаксис свойств Clarion можно использовать для выполнения в тексте вашей программы предложений языка SQL, указывая PROP:SQL с именем файла как назначение в присваивании. Это справедливо только для использования драйверов, поддерживающих SQL (таких как ODBC, AS/400, или Oracle).



Можно использовать любые предложения языка SQL, поддерживаемые SQL сервером. Если выполнено предложение, результатом которого является набор записей (такое как предложение SELECT), то для выборки записей в буфер (по одной за раз) нужно использовать оператор NEXT(файл). Функции FILEERRORCODE() и FILEERROR() будут возвращать коды ошибочных ситуаций и строки сообщений, установленные SQL сервером.

Можно также опросить содержимое свойства PROP:SQL, для того получить последнее переданное драйверу предложение языка SQL.

### Пример:

```
SQLFile{PROP:SQL} = 'SELECT field1,field2 FROM table1'      |
                   & 'WHERE field1 > (SELECT max(field1)'   |
                   & 'FROM table2'                         |
                   !Возвращает набор записей, которые получаете
                   ! по одной, используя
                   ! NEXT(SQLFile)
SQLFile{PROP:SQL} = 'CALL GetRowsBetween(2,8)'             !Вызвать запомненную процедуру
SQLFile{PROP:SQL} = 'CREATE INDEX ON table1 (field1, field2 DESC)'
                   !нет набора выбранных записей
SQLString = SQLFile{PROP:SQL}
                   !Получить последнее предложение SQL, выполненной драйвером
```

### PROP:SQLFilter

Можно использовать свойство PROP:SQLFilter для того, чтобы установить фильтр записей для структуры VIEW, используя естественный синтаксис SQL, а не операторы Clarion. Это допустимо только при использовании SQL драйверов (таких как ODBC, AS/400, или Oracle drivers).

Когда используется свойство PROP:SQLFilter, то фильтр на языке SQL передается непосредственно серверу. А поэтому фильтр не может содержать имен переменных и функции недоступные серверу.

### Пример:

```
View{PROP:SQLFilter} = 'DateField = TO_DATE(''01-MAY-1995'', ''DD-MON-YYYY'')'
View{PROP:SQLFilter} = 'StrField LIKE 'AD%'''
```



## Приложение Д Сообщения об ошибках

### Ошибки времени выполнения

Ошибки времени выполнения, отслеживаемые в программе

В прикладной программе с помощью функций `ERRORCODE` и `ERROR` можно отследить возникновение приведенных далее ошибочных ситуаций. Для каждой ошибки есть числовой код (возвращаемый функцией `ERRORCODE`) и соответствующее текстовое сообщение (возвращаемый функцией `ERROR`), поясняющее что случилось.

2	File Not Found	В заданном каталоге нет указанного файла.
3	Path Not Found	Каталог, указанный как часть пути в файловой системе, не существует.
4	Too Many Open Files	Использованы все доступные идентификационные номера файлов. Проверьте значение, установленное в файле <code>CONFIG.SYS</code> в строке <code>FILES=</code> , или в установках сетевой операционной среды число одновременно открытых файлов.
5	Access Denied	Файл уже открыт другим пользователем в режиме исключительного доступа, оставлен в заблокированном состоянии или у вас нет прав в сети на открытие данного файла. Кроме того, эта ошибка может возникать, когда недостаточно свободного места на диске.
7	Memory Corrupted	По неизвестной причине произошло некое разрушение данных в памяти.
8	Insufficient Memory	Для выполнения операции не хватает свободной оперативной памяти. Закрыв другие приложения, можно освободить требуемое количество памяти. В случае использования драйвера <code>Btrieve</code> , эта ошибка означает, что в вашем распоряжении недостаточно памяти в реальном режиме для загрузки модуля <code>BTR32.EXE</code> . В Windows 95 этой проблемы можно избежать, загрузив в <code>WINSTART.BAT</code> модуль <code>WBTR32.EXE</code> .
30	Entry Not Found	Ошибка при выполнении оператора <code>GET</code> для <code>QUEUE</code> . В форме <code>GET(очередь,ключ)</code> не найдено соответствующее значение ключа, а для формы <code>GET(очередь,указатель)</code> , указатель выходит за границы допустимого диапазона.
32	File Is Already Locked	Попытка заблокировать файл не удалась, потому что его уже заблокировал другой пользователь.
33	Record Not Available	Обычно возникает при попытке считать запись после последней (или перед первой) записи файла оператором

		NEXT или PREVIOUS. Может также выдаваться при выполнении операторов PUT или DELETE, если запись предварительно не была прочитана.
35	Record Not Found	При выполнении оператора в форме GET(файл,ключ), не найдена запись с соответствующим значением ключевого поля.
36	Invalid Data File	Возникло нарушение структуры файла данных, природа которого неясна.
37	File Not Open	Попытка выполнить некую операцию, требующую предварительного открытия файла, завершилась аварийно вследствие того, что файл не открыт.
38	Invalid Key File	Возникло нарушение структуры файла ключей, природа которого неясна.
40	Creates Duplicate Key	Попытка внести в файл оператором ADD или PUT запись, в которой значение ключевого поля совпадает со значением этого поля в другой, уже находящейся в файле записи, причем для файла определено, что дублирование ключевых полей недопустимо.
43	Record Is Already Held	Попытка выполнить оператор HOLD применительно к некоторой записи не удалась из-за того, что другой пользователь уже заблокировал эту запись.
45	Invalid Filename	Имя файла не соответствует правилам образования имен файлов в DOS.
46	Key File Must Be Rebuilt	Возникло нарушение структуры файла ключей, природа которого неясна, и требуется повторное построение ключевого файла оператором BUILD.
47	Invalid Record Declaration	Структура файла данных на диске не соответствует структуре, объявленной в программе. Обычно, такая ситуация возникает из-за того, что поменяли описание файла в Словаре Данных и еще не преобразовали сам файл данных в новый формат.
48	Unable To Log Transaction	Файл регистрации транзакции или pre-image файл нельзя записать на диск. Обычно возникает вследствие недостатка свободного места на диске или у пользователя нет соответствующих прав в сетевой среде.
52	File Already Open	Попытка открыть файл, который данный пользователь уже открыл.
53	Invalid Clarion File	Означает, что файл имеет неправильный заголовый dBase. Эта ошибка возникает только с драйверами семейства xBase.
54	No Create Attribute	Попытка выполнить процедуру CREATE применительно к файлу, в объявлении которого отсутствует атрибут CREATE.

- 56 LOGOUT Already Active Попытка выполнить второй оператор LOGOUT, тогда как транзакция еще не закончилась.
- 57 Invalid Memo File Возникло нарушение структуры файла мемо-полей, природа которого неясна. Для файлов данных Clarion причиной может быть нарушение сигнатуры .MEM файла или указателей на мемо-поле в файле данных, которое свидетельствует о “рассогласовании” (обычно из-за копирования файлов с одного места в другое и копирования несоответствующего .MEM файла).
- 63 Exclusive Access Required Попытка выполнить оператор BUILD(файл), BUILD(ключ), EMPTY(файл) или PACK(файл), в то время как файл не открыт в режиме исключительного доступа.
- 64 Sharing Violation Попытка выполнить некое действие с файлом, для которого требуется, чтобы файл был открыт в режиме совместного использования.
- 65 Unable To ROLLBACK Попытка выполнить оператор ROLLBACK закончилась по Transaction неизвестной причине неудачно.
- 73 Memo File Missing Попытка открыть файл, в объявлении которого имеется мемо-поле, а файл, содержащий данные мемо-полей отсутствует.
- 75 Invalid Field Type Descriptor Или описатель типа поля испорчен, в операторе GET(очередь,имя) употреблено имя несуществующей переменной, или описание файла не соответствует файловому драйверу. Например, попытка определить поле LONG в файле xBase не имеющего соответствующего поля MEMO.
- 76 Invalid Index String Строка, указанная в операторе BUILD(Динамический индекс,строка) имеет неправильный формат.
- 77 Unable To Access Index Попытка прочитать запись на основе динамического индекса не удалась из-за того, не найден нужный динамический индекс.
- 78 Invalid Number Of Parameters В операторе EVALUATE в процедуру или функцию передается неверное число параметров.
- 79 Unsupported Data Type In File Файловый драйвер обнаружил в файле поле, тип которого не поддерживается файловой системой данного драйвера.
- 80 Unsupported File Driver Function Файловый драйвер обнаружил недопустимый оператор доступа к файлу. Обычно это неподдерживаемая форма оператора (отличающаяся параметрами), который вообще-то поддерживается.
- 81 Unknown Error Posted Файловый драйвер обнаружил некую ошибку в файловой системе, о которой он не может получить более детальной информации.

88	Invalid Key Length	Попытка создать ключ или индекс по полю длиннее 245 символов для файла с драйвером Clarion.
89	Record Changed By Another Station	При выполнении в сетевой операционной среде оператора WATCH обнаружено, что запись в файле на диске, которую нужно изменить в сетевой операционной среде, не соответствует первоначальной версии записи.
90	File Driver Error	Файловый драйвер обнаружил некую ошибку при работе с файловой системой. Для уточнения природы этой ошибки можно использовать функции FILEERRORCODE и FILEERROR.
800	Illegal Expression	Функция EVALUATE обнаружила синтаксическую ошибку в выражении.
801	Variable Not Found	Функция EVALUATE не нашла переменную, использованную в выражении. Сначала нужно выполнить оператор BIND, чтобы указать все переменные, видимые EVALUATE.

### **Ошибки времени выполнения, не обнаруживаемые в программе**

Приведенные далее ошибки, возникающие во время выполнения программы, нельзя обнаружить с помощью функций ERRORCODE и ERROR.

ACCEPT loop requires a window	Для цикла ACCEPT нет соответствующего ему окна.
ENDPAGE must only be called for reports	Попытка выполнить оператор ENDPAGE, еогда нет активной структуры REPORT.
Event posted to a report control	Попытка сгенерировать событие применительно к объекту в структуре REPORT.
Metafile record too large in report	Метафайл .WMF слишком велик, чтобы его можно было напечатать в отчете.
Mismatch with CWVBX.DLL detected	Версия первого встреченного в пути поиска в файловой системе файла CWVBX.DLL отличается от версии, которая использовалась при создании .EXE - файла.
PRINT must only be called for reports	Попытка напечатать структуру, которая не является частью структуры REPORT.
Report is already open	Попытка открыть отчет, который уже открыт и еще не закрыт.
Too many keystrokes PRESSed	Параметр оператора PRESS содержит слишком много символов.
Unable to complete operation (system is MODAL)	Попытка выполнить недопустимое действие в программе, которая уже раскрыла модальное окно.
Unable to create control (system is MODAL)	Попытка создать объект оператором CREATE в программе, открывшей окно с атрибутом

## MODAL.

Unable to open APPLICATION (APPLICATION already active)	Попытка открыть окно APPLICATION в программе, в которой уже открыто окно MDI.
Unable to open APPLICATION (system is MODAL)	Попытка открыть окно APPLICATION в программе, в которой уже открыто окно с атрибутом MODAL.
Unable to open APPLICATION	Неудачная попытка открыть окно APPLICATION
Unable to open MDI window (No APPLICATION active)	Попытка открыть окно MDI в программе, которая еще не открыла окно APPLICATION с атрибутом MDI.
Unable to open MDI window (system is MODAL)	Попытка открыть окно MDI в программе, которая уже открыла окно с атрибутом MODAL.
Unable to open MDI window on APPLICATION's thread	Попытка открыть окно MDI в том же самом процессе, в котором открыто окно APPLICATION с атрибутом MDI.
Unable to open MDI WINDOW	Неудачная попытка открыть окно с атрибутом MDI
Unable to open WINDOW	Неудачная попытка открыть окно

## Ошибки компиляции

Компилятор выдает сообщение об ошибке точно в том месте исходного текста программы, в котором обнаружил нарушение синтаксиса. Поэтому, неправильная конструкция находится или справа от этого места, или где-то а предшествующем тексте. Для большинства сообщений об ошибках неправильная конструкция находится справа от места обнаружения, но некоторые сообщения относятся к ошибкам, причина которых лежит гораздо раньше того момента, когда ошибка будет обнаружена компилятором. Для обнаружения таких ошибок наряду с пониманием того, что пытался сообщить вам компилятор, нужна еще некоторая “исследовательская” работа

Одна (относительно небольшая) ошибка может породить “лавинный эффект”; - длинный список сообщений об ошибках, которые все имеют одну причину. Как правило в этом случае и возникает ситуация, когда в одном модуле исходного текста появляется длинный список сообщений об ошибках. Чтобы устранить такую ситуацию, нужно исправить только первую ошибку, а затем прекомпилировать и посмотреть, сколько ошибок останется (довольно часто бывает, что не одной). Если же имеется только пара ошибок, далеко друг от друга отстоящих в тексте программы, то похоже, что это независимые ошибки и придется все их найти и исправить перед повторной компиляцией.

## Особенные ошибки

Приведенные далее сообщения выдаются, когда компилятор обнаружил особую синтаксическую ошибку и пытается точно указать ее причину, чтобы вы могли ее исправить.

В некоторых из приведенных далее сообщений содержится шаблон “%V”. Вместо него компилятор подставляет точное имя, указывая, что произошло с этим именем тем самым уточняя и место возникновения и причину ошибки.

! introduces a comment      Это типичная ошибка программистов на С. Если написать IF A != 1 THEN, то и получите такое сообщение.

Actual value parameter cannot be array      Передаваемый параметр не должен быть массивом.

ADDRESS parameter ambiguous ADDRESS(метка), где метка представляет собой имя и процедуры и имя элемента данных.

All fields must be declared before JOINS      В структуре VIEW любой оператор JOIN должен идти после всех операторов PROJECT.

Ambiguous label      Неоднозначность в синтаксисе уточнения имени.

Например:

G      GROUP

S:T      SHORT      !Обращаются как к G:S:T  
END

G:S      GROUP

T      SHORT      ! Обращаются как к G:S:T  
END

CODE

G:S:T = 7      !Что здесь имеется ввиду?

Array too big      В 16-ти разрядных приложениях размер массивов ограничен 64 К.

Attribute parameter must be QUEUE, QUEUE field or constant string

Параметр должен быть меткой объявленной ранее структуры QUEUE, поля в структуре QUEUE или строковой константой.

Attribute requires more parameters

Атрибуту нужно передавать все необходимые параметры.

Attribute string must be constant

Параметр должен быть строковой константой, а не меткой переменной.



Attribute variable must be global

Данный параметр должен быть переменной, объявленной в программном модуле, как глобальная переменная.

Attribute variable must have string type

Данный параметр должен быть переменной типа `STRING`, `CSTRING`, или `PSTRING`.

BREAK structure must enclose DETAIL

Внутри вложенной структуры `BREAK` (на самом нижнем уровне) должна быть объявлена хотя бы одна структура `DETAIL`.

Calling function as procedure

Предупреждение о том, что к функции обращаются как процедуре и возвращаемое значение будет потеряно.

Cannot call procedure as function

К функции можно обращаться как процедуре, но процедуру нельзя использовать как функцию.

Cannot declare KEY in a VIEW

В структуре нельзя объявлять ключ.

Cannot EXIT from here

Оператор `EXIT` может содержаться только внутри локальной подпрограммы (`ROUTINE`).

Cannot GOTO into ROUTINE

Метка перехода в операторе `GOTO` должна быть меткой исполняемого оператора в той же процедуре или локальной подпрограмме, где находится оператор `GOTO` и не может быть именем локальной подпрограммы.

Cannot have default parameter here

Для целочисленного параметра, передаваемого значением, можно передавать значение по умолчанию, но совсем опускать значение нельзя.

Cannot have initial values with OVER

Переменная, объявленная с атрибутом `OVER` не может иметь к тому же начальное значение.

Cannot have statement here

Это сообщение выдается, когда компилятор видит, что вы попытались определить исполняемый оператор внутри раздела глобальных данных.

Cannot initialize variable reference

Указанная переменная не может иметь начального значения.

Cannot return CSTRING from CLARION function

Для функций, написанных на языке Clarion, CSTRING является недопустимым типом возвращаемого значения (значения такого типа могут возвращать только функции, написанные на других языках).

Cannot RETURN value from procedure

Оператор RETURN с параметром возвращаемое значение может содержаться только в функции.

CLARION function cannot use RAW or NAME

Для функций и процедур, написанных на языке Clarion, такие атрибуты недопустимы (только функции, написанные на других языках могут сопровождаться такими атрибутами).

DECIMAL has too many places

Переменные DECIMAL и PDECIMAL могут содержать только 30 десятичных разрядов и дробная часть не может превышать общую длину.

DECIMAL too long

Переменные DECIMAL и PDECIMAL могут иметь длину максимум 31.

Declaration not valid in FILE structure

В структуре FILE нельзя объявлять такие данные.

Declaration too big

В 16-ти разрядном приложении компилятор обнаружил PSTRING > 255 или MEMO > 64K.

DLL attribute requires EXTERNAL attribute

Атрибут DLL предполагает наличие атрибута EXTERNAL.

Dynamic INDEX must be empty

Попытка использовать форму с двумя параметрами оператора BUILD применительно к ключу или индексу, объявленному с полями-компонентами.

Embedded OVER must name field in same structure

Параметр атрибута OVER должен быть меткой ранее объявленной переменной в той же самой структуре.

ENCRYPT attribute requires OWNER

Атрибуты ENCRYPT и OWNER работают совместно.

Entity-parameter cannot be an array

Нельзя передавать массив параметров-объектов (FILE, QUEUE, и т.д.).

Expected a PROJECT statement

В структуре VIEW должен быть по крайней мере один оператор PROJECT.

Expected: %V

Это наиболее общая ошибка. Компилятор предполагает, что далее будет идти некий синтаксический элемент (один из элементов подставляемых в сообщение вместо шаблона %V), но вместо этого обнаруживает в исходном тексте нечто, не соответствующее синтаксическому контексту.

Expression cannot be picture

Попытка использовать мнемоническую метку соответствия шаблона там, где применение шаблона недопустимо.

Expression cannot have conditional type

Выражение не дает числового значения. Например, MyValue = A > B неверно.

Expression must be constant

В данном выражении использование переменной недопустимо.

Field equate label not defined: %V

Указанное мнемоническое имя соответствия экранного поля не было предварительно объявлено.

Field not found

При использовании синтаксиса уточнения имен присутствует ссылка на поле, не являющееся частью порождающего объекта. Например, указание MyGroup.SomeField, в котором SomeField не объявлено в MyGroup.

Field not found in parent FILE

В операторе JOIN должны объявляться все поля, связывающие главный и вторичный файлы.

Field requires (more) subscripts

Ссылка на элемент многомерного массива должна содержать индекс для каждого измерения.

FILE must have DRIVER attribute

Для того, чтобы определить файловую систему, для которой объявляется файл, в объявлении файла обязателен атрибут DRIVER.

FILE must have RECORD structure

Недопустимо объявлять структуру FILE, в которой нет структуры RECORD.

FILEs must have same DRIVER attribute

Все файлы, упомянутые в операторе LOGOUT должны использовать одну и ту же файловую систему.

Function did not return a result

Предупреждение о том, что реализация функции не возвращает результат.

FUNCTION must have return type

Если в структуре MAP объявлен прототип с возвращением значения, то ему должна соответствовать функция.

Function result is not of correct type

Оператор RETURN должен возвращать значение согласующееся с типом возвращаемого значения, указанным в прототипе функции в структуре MAP.

Group too big

В 16-ти разрядном приложении размер групп ограничен 64K.

Ignoring EQUATE redefinition: %V

Сообщение о том, что указанное мнемоническое имя соответствия игнорируется. Это действительная ошибка переопределения имени, при этом основное определение остается в силе.

Illegal array assignment

При присвоении значения элементу массива и должен указываться отдельный элемент, а не целый массив.

Illegal character

Недопустимый лексический элемент. Например код ASCII 255 встреченный в исходном тексте.

Illegal data type: %V

Указанный тип данных не соответствует структуре в которой он находится.

Illegal key component

В ключе имеется компонент недопустимого типа.

Illegal nesting of window controls

В структуру OPTION помещен экранный объект, не являющийся кнопкой RADIO или в структуру SHEET помещен объект, не являющийся объектом типа TAB.

**Illegal parameter for LIKE**

Неправильный параметр в объявлении LIKE. Например, LIKE(7).

**Illegal parameter type for STRING**

Неправильный параметр в объявлении STRING. Например, STRING(MyVar).

**Illegal reference assignment**

Значение переменной-указателя можно присвоить только другой переменной-указателю такого же типа.

**Illegal return type or attribute**

Прототип содержит в качестве возвращаемого недопустимый тип данных (такой как \*CSTRING).

**Illegal target for DO**

Параметром оператора DO должно быть имя локальной подпрограммы (ROUTINE).

**Illegal target for GOTO**

Метка перехода в операторе GOTO должна быть меткой исполняемого оператора в той же процедуре или локальной подпрограмме, где находится оператор GOTO и не может быть именем локальной подпрограммы.

**INCLUDE invalid, expected: %V**

Параметр оператора INCLUDE должен быть правилам сформированной строкой. В частности, недопустимо преобразование типа, таким образом неправилен оператор INCLUDE('MyFile'&MyValue).

**INCLUDE misplaced**

Оператор INCLUDE должен следовать за разделителем строк или точкой с запятой (за которой могут идти несколько пробелов).

**INCLUDE nested too deep**

Вложенность операторов INCLUDE не может превышать 3-х. Другими словами можно вставить оператором INCLUDE файл, в который вставить оператором INCLUDE файл, в который вставить оператором INCLUDE файл, Но в последний уже вставлять нельзя.

**Incompatible assignment types**

Попытка присвоения данных несовместимых типов.

**Incorrect procedure profile**

Попытка в качестве параметра-процедуры передать процедуру, не соответствующую прототипу.

Indices must be constant

Попытка сделать USE-переменную, которая представляет собой массив с переменными индексами.

Indistinguishable new prototype: %V

Имеется прототип, который компилятор не в состоянии отличить от предыдущего прототипа, используя правила перегрузки функций.

Integer expression expected

Выражение должно давать в результате целочисленное значение.

Invalid BREAK statement

Оператор BREAK, который пытается прервать несуществующий цикл или находится вне структуры LOOP или ACCEPT.

Invalid CYCLE statement

Оператор CYCLE который пытается начать новую итерацию несуществующего цикла или находится вне структуры LOOP или ACCEPT.

Invalid data declaration attribute

Атрибут, который не соответствует данному объявлению данных.

Invalid data type for value parameter

Тип данных указанные в прототипе в структуре MAP не может передаваться по значению и должен передаваться “адресом”. Например, чтобы передать параметр типа CSTRING в процедуру на языке Clarion он должен прототипироваться только как \*CSTRING.

Invalid FILE attribute

Атрибут, который не соответствует объявлению файла.

Invalid first parameter of ADD

Первый параметр оператора ADD неверен.

Invalid first parameter of FREE

Первый параметр оператора FREE неверен.

Invalid first parameter of NEXT

Первый параметр оператора NEXT неверен.

Invalid first parameter of PUT

Первый параметр оператора PUT неверен.

Invalid GROUP/QUEUE/RECORD attribute

Атрибут не соответствует объявлению GROUP, QUEUE, или RECORD.

Invalid KEY/INDEX attribute

Атрибут не соответствует объявлению KEY или INDEX.

Invalid label

Метка, в которой содержатся символы отличные от букв, цифр, знака подчеркивания (\_), и двоеточия (:), или начинающаяся не с буквы или символа подчеркивания.

Invalid LOOP variable

Попытка использовать недопустимый тип данных (DATE, TIME, STRING, и т.д.) в качестве переменной цикла.

Invalid MEMBER statement

Параметр оператора MEMBER не является строковой константой или не указывает на программный модуль текущего проекта.

Invalid method invocation syntax

Попытка использовать синтаксис {} при обращении к методу с применением объекта BLOB или FILE.

Invalid number

Требуется число, например в качестве коэффициента повторения в строковой константе ({}).

Invalid OMIT expression

Параметр оператора OMIT неверен.

Invalid parameters for attribute

Атрибуту, который предполагает наличие параметра, нужно передавать правильный параметр.

Invalid picture token

Шаблон содержит несоответствующие символы.

Invalid printer control token

Оператор PRINT содержит символы управления принтером отличные от @CR, @LF, и @FF.

Invalid QUEUE/RECORD attribute

Атрибут, который не соответствует объявлению QUEUE или RECORD.

Invalid SIZE parameter

SIZE(ерунда+еще\_что-то)

Invalid string (misused <...> or {...} )

В строковой константе содержится дна открывающая скобка (< или {) и нет соответствующей закрывающей (> или }). Если они являются частью строки, то такие символы должны представляться удвоением (<< или {{).

Invalid structure as first parameter

Первый параметр оператора неверен.

Invalid structure within property syntax

Структура, которая недопустима в операторе присвоения значения свойству.

Invalid USE attribute parameter

Параметр не соответствует атрибуту USE.

Invalid use of PRIVATE data

Попытка обращения к личным данным вовне класса..

Invalid use of PRIVATE procedure

Попытка обращения к личному методу вовне класса.

Invalid variable data parameter type

При передаче параметров “адресом”, нужно передавать данные того типа, который указан в прототипе в структуре MAP.

Invalid WINDOW control

Объект, которые не допустим в структуре WINDOW.

ISL error: %V

Свяжитесь со Службой технической поддержки и сообщите подробности, связанные с появлением этого сообщения.

KEY must have components

Нельзя объявить ключ, не указав поля-компоненты, которые устанавливают порядок сортировки.

Label duplicated, second used: %V

Указанная mnemonic метка соответствия объявлялась несколько раз в пределах одного и того же модуля и только последняя использована в списке mnemonic меток соответствия, допустимых для использования в исполняемых операторах в данном модуле. Исправляется с помощью третьего параметра атрибута USE.

Label in prototype not defined: %V

Использование прототипа, в котором один из типов данных еще не определен.



Label not defined: %V

Указанная метка не была предварительно объявлена.

Mis-placed string slice operator

Часть строки которая не является последним измерением массива. Например, `MyStringArray[3:4,5]`.

Missing procedure definition: %V

Указанная процедура не представлена прототипом в структуре MAP.

Missing virtual function

Ошибка компилятора.

Must be dimensioned variable

Здесь должен указываться массив.

Must be field of a FILE or VIEW

Здесь должно быть поле из реального или виртуального файла. Например, `NULL(LocalVariable)` даст такое сообщение об ошибке.

Must be FILE or KEY

Параметр оператора JOIN не является меткой структуры FILE или KEY.

Must be reference variable

В DISPOSE можно использовать только переменную-указатель.

Must be variable

Здесь должна быть метка предварительно объявленной переменной.

Must have constant string parameter

Данный параметр должен быть строковой константой, но не переменной.

Must specify DECIMAL size

В объявлении переменной типа DECIMAL или PDECIMAL должно быть указано максимальное число хранимых цифр.

Must specify identifier

Требуется, но отсутствует идентификатор.

Must specify print-structure

Оператором PRINT может печататься только структура входящая в структуру REPORT.

No matching prototype available

Попытка определить процедуру, для которой нет соответствующего прототипа в структуре MAP или CLASS.

Not valid inside structure

В структуре имеются данные несоответствующего типа.

OMIT cannot be nested

Оператор OMIT не может быть вложенным.

OMIT misplaced

Перед оператором OMIT может быть разделитель строк или точка с запятой (после которой возможно некоторое количество пробелов).

OMIT not terminated: %V

Указанный в операторе OMIT параметр не найден до конца исходного модуля.

Order is MENUBAR, TOOLBAR, Controls

Структура MENUBAR должна предшествовать структуре TOOLBAR, а структура TOOLBAR должна предшествовать объявлениям объектов в WINDOW или APPLICATION.

OVER must name variable

Параметр атрибута OVER должен быть меткой ранее объявленной переменной.

OVER must not be larger than target variable

Параметр атрибута OVER должен быть меткой ранее объявленной переменной, которая больше или равна по длине переменной, которая объявляется поверх нее.

OVER not allowed with STATIC or THREAD

Переменная, объявленная с атрибутом OVER не может иметь атрибут STATIC или THREAD (их должна иметь базовая переменная).

Parameter cannot be omitted

При обращении в процедуру или функцию должны передаваться все параметры, для которых в прототипе не указано, что они могут опускаться.

Parameter kind does not match

При передаче параметра “адресом” должны передаваться данные того типа, который был указан в прототипе в структуре MAP.

Parameter must be picture

Здесь должен быть шаблон отображения.

Parameter must be procedure label

Здесь должно быть имя процедуры.

Parameter must be report DETAIL label

Оператором PRINT может печататься только структура входящая в структуру REPORT

Parameters must have labels

Попытка определить процедуру, не используя имен параметров

Parameter type label ambiguous (CODE or DATA)

У данных и процедуры может совпадать имя, но в этом случае его нельзя использовать в прототипе процедуры.

PROCEDURE cannot have return type

Если в структуре MAP в прототипе отсутствует возвращаемое значение, то этому прототипу должна соответствовать процедура.

Procedure doesn't belong to module: %V

Попытка объявить процедуру, прототип которой определен как принадлежащий другому модулю.

Procedure in parent CLASS has VIRTUAL mismatch

Для виртуального метода требуется атрибут VIRTUAL в прототипах и в родительском классе и в порожденном классе.

Prototype is: %V

Попытка определить процедуру с неправильным прототипом.

QUEUE/RECORD not valid in GROUP

Структура GROUP не может содержать структуры QUEUE или RECORD.

Redefining system intrinsic: %V

Предупреждение о том, что указанная процедура (часть исходного кода) имеет имя совпадающее с именем библиотечной процедуры или функции в библиотеке исполняющей системы Clarion, и что вместо встроенной процедуры или функции будет происходить обращение к вашей процедуре или функции.

Routine label duplicated

Метка оператора ROUTINE уже использовалась ранее в другом операторе.

Routine not defined: %V

Обозначенная локальная подпрограмма (ROUTINE) не существует.

SECTION duplicated: %V

Указанная секция имеется во включаемом оператором INCLUDE файле в двух экземплярах.

SECTION not found: %V

Указанная секция во включаемом оператором INCLUDE файле отсутствует.

Statement label duplicated

Две строки исполняемых операторов имеют одинаковую метку.

Statement must have label

Оператор (такой как ROUTINE или PROCEDURE) должен иметь метку.

String not terminated

У строковой константы отсутствует заключительная кавычка (').

Subscript out of range

Попытка указать элемент массива, один или несколько индексов которого выходят за допустимые границы.

Too few indices

Это ссылка на элемент массива, который имеет несколько измерений и для него нужно указывать индексы по всем измерениям.

Too few parameters

При обращении в процедуру или функцию должны передаваться все параметры, для которых в прототипе не указано, что они могут опускаться.

Too many indices

В ссылке на элемент массива указано индексов больше, чем количество измерений массива.

Too many parameters

При обращении к процедуре или функции не может передаваться параметров больше чем указано в прототипе.

Unable to verify validity of OVER attribute

Предупреждение о том, что вы объявляете переменную поверх принимаемого параметра и во время выполнения типы данных могут не соответствовать.

Unknown attribute: %V

Указанный атрибут не принят в языке Clarion.

Unknown function label

Функция не представлена прототипом в структуре MAP.

Unknown identifier

Имя предварительно не объявлено.

Unknown identifier: %V

Указанный идентификатор предварительно не объявлен.

Unknown key component: %V

Указанный компонент ключа не существует в структуре FILE.

Unknown procedure label

Процедура не представлена прототипом в структуре MAP.

UNTIL/WHILE illegal here

Попытка использовать UNTIL или WHILE для завершения уже заверщенного цикла.

Value-parameter cannot be an array

Нельзя передавать массив как параметр-значение.

Value requires (more) subscripts

Это ссылка на массив, который имеет несколько измерений, поэтому нужно указать индексы по всем измерениям.

Variable expected

Здесь должна быть метка предварительно объявленной переменной.

Variable-size must be constant

В объявлении переменной в качестве параметра, определяющего длину, должно указываться константное выражение.

VIRTUAL illegal outside of CLASS structure

Атрибут VIRTUAL можно использовать только в прототипах в структуре CLASS, но не в MAP.

Wrong number of parameters

При обращении в процедуру или функцию должны передаваться все параметры, для которых в прототипе не указано, что они могут опускаться.

Wrong number of subscripts

Попытка обратиться к многомерному массиву, не указав индексы по каждому измерению.

Например:

```
MyShort SHORT,DIM(8,2)
```

```
CODE
```

```
MyValue = MyShort[7]
```

```
!Неправильное число индексов
```

## Ошибки неопределенного характера

---

Существуют ошибки которые никогда не должны возникать, и ключ к причинам которых компилятор пытается дать разработчикам компилятора. Сообщите о проблеме в фирму TopSpeed и передайте файл исходного текста, при компиляции которого возникла ошибочная ситуация.

Inconsistent scanner initialization

Unknown operator

Unknown expression type

Unknown expression kind

Unknown variable context

Unknown parameter kind

Unknown assignment operator

Unknown variable type

Unknown case type

Unknown equate type

Unknown string kind

Unknown picture type

Unknown descriptor type

Unknown initializer type

Unknown designator kind

Unknown structure field

Unknown formal entity

Type descriptor not static

Unknown clear type

Unkonwn simple formal type

Out of attribute space

Unknown label/routine

Unknown special identifier

Value not static

Unknown static label

Unknown screen structure kind

Corrupt pragma string

Old symbol non-NIL

Not implemented yet

String not CCST

## Приложение Е. Мнемонические имена событий

### События

В программах на языке Clarion для Windows большинство сообщений от Windows автоматически обрабатываются внутри обработчика событий АССЕРТ. Существуют события общего характера, которые обрабатываются библиотечными функциями исполняющей системы (перерисовка экрана и т.п.). И только те события, которые действительно могут потребовать реакции программы, передаются обработчиком событий АССЕРТ в программу. Результатом такого подхода явилось облегчение прикладного программирования из-за устранения из программы низкоуровневого кодирования рутинных операций, что позволяет вместо этого сконцентрировать внимание на аспектах прикладного уровня. Конечно, можно также обрабатывать низкоуровневые сообщения Windows и самому, но делать это стоит только в абсолютно необходимых случаях. Если потребуется более подробная информация о программировании для среды Windows, обратитесь к книге Чарльза Петцольда *Programming Windows*, опубликованной Microsoft Press.

В программу обработчиком событий АССЕРТ передаются события двух видов: события связанные с полем и события не связанные с полем. Ниже приведен список мнемонических имен соответствия событий, который содержится в файле EQUATES.CLW.

### События не связанные с полем

---

События не связанные с полем не относятся ни к одному из экранных объектов, однако требуют от программы некоторой реакции (например, закрыть окно или переключить исполняемый процесс). Большинство из этих событий помещают систему в модальное состояние на время обработки этого события, поскольку прежде чем продолжить выполнение программы требуется получить ее реакцию.

**EVENT:AlertKey** Пользователь нажал горячую клавишу, определенную атрибутом ALRT для окна. Это событие, при обработке которого и выполняются действия, запрошенные пользователем нажатием горячей клавиши.

**EVENT:CloseDown** Приложение закрывается. Выдача этого события закрывает приложение. Это событие, при обработке которого выполняются некоторые завершающие действия.

**EVENT:CloseWindow** Окно закрывается. Выдача этого события закрывает окно. Это событие, при обработке которого выполняются некоторые завершающие действия.

EVENT:Completed	Закончена обработка всех экранных полей в безостановочном режиме (AssertAll). Это событие, по которому выполняются процедуры проверки для всех введенных данных во всех полях окна и данные можно спокойно записывать на диск.
EVENT:DDEadvise	Программа-клиент в процессе DDE обмена запросила постоянное обновление данных, передаваемых программой-сервером, написанным на языке Clarion. Это событие, по которому вы выполняете оператор DDEWRITE, чтобы передавать клиенту данные каждый раз, когда они изменяются.
EVENT:DDEclosed	DDE сервер разорвал канал DDE для данного приложения-клиента, написанного на языке Clarion.
EVENT:DDEdata	DDE сервер передал обновленный элемент данных этой конкретной программе-клиенту, написанной на языке Clarion.
EVENT:DDEexecute	Программа-клиент прислала команду данному DDE серверу (если клиент написан на языке DDE, то он выполнил оператор DEEXECUTE). Это событие по которому вы определяете действие, запрошенное клиентом, и выполняете его, затем выполняете оператор CYCLE, чтобы подтвердить выполнение клиенту, который прислал команду.
EVENT:DDEpoke	Клиент прислал не запрошенные данные этому серверу, написанному на языке Clarion. Это событие, при обработке которого вы определяете, что прислал клиент и куда его поместить, затем выполняете оператор CYCLE, чтобы подтвердить получение клиенту, который прислал данные.
EVENT:DDErequest	Программа-клиент в процессе DDE обмена запросила данные от программы-сервера, написанной на языке Clarion. Это событие, по которому вы выполняете оператор DDEWRITE, чтобы один раз передать клиенту данные.
EVENT:GainFocus	Фокус ввода переключается на данное окно из другого исполняемого процесса. Это событие, при обработке которого выполняется восстановление данных, запомненных при обработке события EVENT:LoseFocus.
EVENT:Iconize	Пользователь сворачивает окно до пиктограммы. Если среди операторов, осуществляющих обработку данного события,



встретится оператор CYCLE, то событие EVENT:Iconized не генерируется и действие прерывается. С помощью этого события можно не дать пользователю свернуть окно до пиктограммы.

**EVENT:Iconized** Пользователь свернул окно до пиктограммы. Это событие, по которому, вы снова выравниваете все элементы, которые зависят от размеров окна.

**EVENT:LoseFocus** Фокус ввода переключается с данного окна на другой исполняемый процесс. Это событие, при обработке которого выполняется сохранение данных, в отношении которых есть риск изменения их в другом исполняемом процессе.

**EVENT:Maximize** Пользователь устанавливает максимальные размеры окна. Если среди операторов, осуществляющих обработку данного события, встретится оператор CYCLE, то событие EVENT:Maximized не генерируется и действие прерывается. С помощью этого события можно не дать пользователю установить максимальные размеры окна.

**EVENT:Maximized** Пользователь установил максимальные размеры окна. Это событие, по которому, вы снова выравниваете все элементы, которые зависят от размеров окна.

**EVENT:Move** Пользователь перемещает окно. Если среди операторов, осуществляющих обработку данного события, встретится оператор CYCLE, то событие EVENT:Moved не генерируется и действие прерывается. С помощью этого события можно не дать пользователю изменить расположение окна.

**EVENT:Moved** Пользователь переместил окно. Это событие, по которому, вы снова выравниваете все элементы в окне, которые зависят от расположения окна.

**EVENT:OpenWindow** Окно открывается. Это событие, при обработке которого выполняются некоторые действия по инициализации работы с окном.

**EVENT:PreAlertKey** Пользователь нажал горячую клавишу, определенную атрибутом ALRT для окна. Если в процессе обработки этого события встретится оператор CYCLE, то событие EVENT:AlertKey не генерируется и действия по его обработке не выполняются.

EVENT:Restore	Пользователь восстанавливает предыдущие размеры окна. Если среди операторов, осуществляющих обработку данного события, встретится оператор CYCLE, то событие EVENT:Restored не генерируется и действие прерывается. С помощью этого события можно не дать пользователю восстановить окно.
EVENT:Restored	Пользователь восстановил предыдущие размеры окна. Это событие, по которому, вы снова выравниваете все элементы, которые зависят от размеров окна.
EVENT:Resume	Фокус ввода остается на данном окне, и ему снова передается управление после остановки по событию EVENT:Suspend.
EVENT:Size	Пользователь изменяет размеры окна. Если среди операторов, осуществляющих обработку данного события, встретится оператор CYCLE, то событие EVENT:Sized не генерируется и действие прерывается. С помощью этого события можно не дать пользователю изменить расположение окна.
EVENT:Sized	Пользователь изменил размеры окна. Это событие, по которому, вы снова выравниваете все элементы, которые зависят от размеров окна.
EVENT:Suspend	Фокус ввода остается на данном окне, но управление передается другому исполняемому процессу для обработки таймерных событий.
EVENT:Timer	Переключен атрибут TIMER. Это событие по которому выполняются какие-либо периодические действия, такие как отображение времени, или фоновая обработка записей для подготовки отчета, или их пакетная обработка.

## События связанные с полем

---

Связанные с полем события возникают, когда пользователь нажимает клавишу, по которой от программы может потребоваться выполнение особенных действий, связанных с данным экранным объектом.

**EVENT:Accepted** Пользователь ввел данные или сделал выбор и затем нажал клавишу TAB или щелкнул мышью, чтобы переключиться на другое поле. Это событие по которому следует выполнить какие-либо действия по проверке введенных данных.

**EVENT:AlertKey** Пользователь нажал горячую клавишу, заданную атрибутом ALRT экранного поля. Это событие, по которому выполняются запрошенные пользователем действия.

**EVENT:Contracted** Для окна-списка, у которого в строке форматирования присутствует шаблон “Т”, пользователь щелкнул на квадратике свертывания.

**EVENT:Contracting** Для окна-списка, у которого в строке форматирования присутствует шаблон “Т”, пользователь щелкнул на квадратике свертывания. Если среди операторов, осуществляющих обработку данного события, встретится оператор CYCLE, то событие EVENT:Contracted не генерируется, а свертывание дерева отменяется.

**EVENT:Drag** Пользователь отпустил кнопку мыши над объектом - допустимым для завершения операции перетаскивания. Это событие генерируется для объекта, из которого пользователь “перетаскивает” данные. Это событие по которому программно осуществляется передача перетаскиваемых данных в принимающий объект.

**EVENT:Dragging** Пользователь выполняет операцию “перетаскивания” из объекта, имеющего атрибут DRAGID, и в данный момент курсор находится в пределах объекта, который потенциально может служить объектом-назначением в который производится перенос. Это событие генерируется для объекта, из которого пользователь “перетаскивает” данные. Это событие по которому можно изменить форму курсора мыши, чтобы обозначить возможность “отпускания”

EVENT:Drop	Пользователь отпустил кнопку мыши над объектом - допустимым для завершения операции перетаскивания. Это событие генерируется для объекта, в который пользователь “перетаскивает” данные. Это событие по которому программно осуществляется прием перетаскиваемых данных в принимающий объект.
EVENT:DroppedDown	В окне списка или комбинированном окне списка с атрибутом DROP список раскрыт. Это событие, по которому вы можете скрыть другие поля, которые перекрывает выпадающий список, чтобы предотвратить беспорядок на экране, который отвлекает пользователя.
EVENT:DroppingDown	В окне списка или комбинированном окне списка с атрибутом DROP пользователь нажал кнопку со стрелкой вниз. Это событие, по которому считываются записи при запросе на раскрытие списка.
EVENT:Expanded	Для окна-списка, у которого в строке форматирования присутствует шаблон “Т”, пользователь щелкнул на квадратике расширения.
EVENT:Expanding	Для окна-списка, у которого в строке форматирования присутствует шаблон “Т”, пользователь щелкнул на квадратике расширения. Если среди операторов, осуществляющих обработку данного события, встретится оператор CYCLE, то событие EVENT:Expanded не генерируется, а раскрытие дерева отменяется.
EVENT:Locate	В окне списка с атрибутом VCR пользователь нажал кнопку VCR “поиск”. Это событие, по которому можно открыть вводное поле локатора, если оно было скрыто.
EVENT:MouseIn	Курсор мыши вошел в пределы поля типа REGION, имеющее атрибут IMM.
EVENT:MouseMove	Курсор мыши перемещается в пределах поля типа REGION, имеющего атрибут IMM.
EVENT:MouseOut	Курсор мыши покинул пределы поля типа REGION, имеющее атрибут IMM.

EVENT:NewSelection	Изменился выбор, сделанный пользователем в окне списка или комбинированном окне списка (выделенная полоса-курсор переместилась вверх или вниз). Это событие по которому выполняются некоторые внутренние действия по синхронизации данных в других экранных объектах с выбранной в данный момент записью в списке.
EVENT:PageDown	В окне списка или комбинированном окне списка пользователь нажал клавишу PgDn. Это событие, по которому вы получаете следующую страницу при перелистывании списка.
EVENT:PageUp	В окне списка или комбинированном окне списка пользователь нажал клавишу PgUp. Это событие, по которому вы получаете предыдущую страницу при перелистывании списка.
EVENT:PreAlertKey	Пользователь нажал горячую клавишу, заданную атрибутом ALRT экранного поля. Если среди операторов, осуществляющих обработку данного события, встретится оператор CYCLE, то событие EVENT:AlertKey не генерируется и действие прерывается.
EVENT:Rejected	Пользователь ввел данные, не соответствующие шаблону поля или значение в поле SPIN, выходящее за границы допустимого диапазона. Функция REJECTCODE возвращает причину, по которой отвергаются введенные пользователем данные, а для того, чтобы прочитав с экрана данные можно использовать свойство PROP:ScreenText. Это событие, по которому вы оповещаете пользователя о сути ошибки, сделанной им при вводе.
EVENT:ScrollBottom	В окне списка или комбинированном окне списка с атрибутом IMM пользователь нажал сочетание клавиш CTRL+PGDN. Это событие, по которому вы получаете последнюю страницу при перелистывании списка.
EVENT:ScrollDown	В окне списка или комбинированном окне списка с атрибутом IMM пользователь попытался переместить выделенную ниже последнего на экране элемента списка. Это событие, по которому вы получаете следующую запись при перелистывании списка.

EVENT:ScrollDrag	В объекте LIST или COMBO с атрибутом IMM пользователь перемещает “бегунок” линейки скроллинга, и только отпустил кнопку мыши. Это событие, по которому вы динамически прокручиваете выводимые записи, основываясь на текущем значении свойства PROP:VScrollPos
EVENT:ScrollTop	В окне списка или комбинированном окне списка с атрибутом IMM пользователь нажал сочетание клавиш CTRL+PGUP. Это событие, по которому вы получаете первую страницу при перелистывании списка.
EVENT:ScrollTrack	В объекте LIST или COMBO с атрибутом IMM пользователь перемещает “бегунок” линейки скроллинга. Это событие, по которому вы динамически прокручиваете выводимые записи, основываясь на текущем значении свойства PROP:VScrollPos
EVENT:ScrollUp	В окне списка или комбинированном окне списка с атрибутом IMM пользователь попытался переместить выделенную полосу-курсор выше верхнего элемента списка. Это событие, по которому вы получаете предыдущую запись при перелистывании списка.
EVENT:Selected	Данный экранный объект получил фокус ввода. Это событие по которому следует выполнить какие-либо действия по инициализации данных.
EVENT:TabChanging	В объекте типа SHEET фокус переключается на другой лист TAB. Это событие, по которому выполняется все необходимые внутренние действия.
EVENT:VBXevent	Для объекта типа CUSTOM возникло характерное для VBX событие. Это событие, по которому вы считываете свойства PROP:VBXEvent и PROP:VBXEventArg, чтобы определить, что произошло и параметры этого события.

# Содержание

## **Предисловие - История создания языка Clarion ..... 23**

Выбор стиля.....	24
Объявление данных .....	26
Типы данных без проблем .....	29
Промежуточные значения .....	31
Управляющие структуры .....	32
Покорение пользовательского интерфейса .....	33
Раскрытие окон .....	35
Проектирование базы данных .....	37
Новая структура VIEW.....	39
Первый компилятор .....	41
Новый партнер .....	42
Что теперь .....	43

## **Глава 1 Введение ..... 45**

Справочное руководство по языку.....	45
Построение книги .....	45
Соглашения и обозначения .....	47
Формат описания элементов языка .....	47
КЛЮЧЕВОЕ_СЛОВО (краткое описание назначения) .....	48

## **Глава 2. Формат исходного текста программы ..... 51**

### **Формат оператора ..... 51**

Имена переменных и метки операторов .....	51
Структуры .....	52
Уточнение имени переменной .....	52
Зарезервированные слова .....	54
Специальные символы .....	54

### **Формат программы ..... 56**

PROGRAM (объявить программу).....	56
MEMBER (идентифицировать дополнительный исходный файл).....	58
MAP (объявить прототипы процедур и/или функций) .....	60
MODULE (указать исходный файл member-модуля) .....	61
PROCEDURE (определить процедуру) .....	62
FUNCTION (определить функцию) .....	64
CODE- (начало исполняемых операторов) .....	66
ROUTINE (объявить локальную подпрограмму) .....	66

<i>END</i> (закончить структуру) .....	67
Последовательность выполнения операторов .....	68
Обращение к процедурам и функциям .....	69
<b>Прототипы процедур и функций</b> .....	<b>71</b>
Синтаксис прототипов .....	71
Списки параметров в прототипах .....	74
Типы значений, возвращаемых функциями .....	82
<i>C</i> , <i>PASCAL</i> (соглашения о связях) .....	83
<i>RAW</i> (передавать только адрес) .....	84
<i>NAME</i> (установить внешнее имя для прототипируемой процедуры) .....	84
<i>TYPE</i> (Задать определение типа процедуры или функции) .....	85
<i>DLL</i> (процедура определена внешне, в библиотеке <i>DLL</i> ) .....	85
<i>PROC</i> (нет предупреждения, что функция вызывается как процедура) .....	86
<i>PRIVATE</i> (использование процедуры ограничено классом или модулем) .....	86
<i>VIRTUAL</i> (установить, что метод виртуальный) .....	87
<b>Перегрузка функций</b> .....	<b>88</b>
Правила перегрузки функций .....	89
Образование имен и совместимость с <i>C++</i> .....	90
<b>Директивы компилятора</b> .....	<b>91</b>
<i>BEGIN</i> (определить операторную структуру) .....	91
<i>COMPILE</i> (указать исходный текст для компиляции) .....	92
<i>INCLUDE</i> (компилировать текст другого файла) .....	93
<i>OMIT</i> (указать текст, который не должен компилироваться) .....	93
<i>SECTION</i> (указать секцию исходного текста) .....	94
<b>Глава 3 Объявление переменных</b> .....	<b>97</b>
<b>Операторы объявления</b> .....	<b>97</b>
<i>BYTE</i> (целочисленная переменная без знака длиной в один байт) .....	97
<i>SHORT</i> (целочисленная переменная со знаком длиной в два байта) .....	98
<i>USHORT</i> (целочисленная переменная без знака длиной в два байта) .....	99
<i>LONG</i> (целочисленная переменная длиной четыре байта со знаком) .....	100
<i>ULONG</i> (целочисленная переменная без знака длиной четыре байта) .....	101
<i>SREAL</i> (переменная с плавающей точкой длиной четыре байта со знаком) .....	102
<i>REAL</i> (переменная с плавающей точкой длиной восемь байт со знаком) .....	103
<i>BFLOAT4</i> (с плавающей точкой длиной четыре байта со знаком) .....	104
<i>BFLOAT8</i> (с плавающей точкой длиной восемь байт со знаком) .....	105
<i>DECIMAL</i> (упакованная десятичная переменная длины со знаком) .....	106
<i>PDECIMAL</i> (упакованная десятичная переменная длины со знаком) .....	107
<i>STRING</i> (строка фиксированной длины) .....	108
<i>CSTRING</i> (строка, заканчивающаяся двоичным нулем) .....	110
<i>PSTRING</i> (строка, включающая байт длины) .....	112
<i>DATE</i> (дата длиной четыре байта) .....	114



TIME (переменная для времени длиной четыре байта) .....	115
GROUP (составная структура данных) .....	116
CLASS (объявление объекта) .....	119
LIKE (наследуемый тип данных) .....	124
Неявно объявляемые переменные .....	126
Переменные-указатели .....	127
<b>Атрибуты переменных .....</b>	<b>129</b>
PRE (объявить префикс) .....	129
DIM (установить размерность массива) .....	130
EXTERNAL (переменная определена во внешней библиотеке) .....	131
DLL (переменная определена в библиотеке .DLL) .....	132
NAME (указать внешнее имя) .....	133
OVER (совместное использование памяти) .....	134
STATIC (статическая локальная переменная) .....	135
THREAD (статическая переменная зависит от исполняемого процесса) .....	136
BINDABLE (переменная, используемая в динамических выражениях) .....	136
AUTO (локальная переменная без начального значения) .....	137
TYPE (определение типа группы или класса) .....	137
PRIVATE (переменные класса доступны только в пределах модуля) .....	138
MODULE (указать модуль исходного текста для методов класса) .....	138
<b>Объявление данных и распределение памяти .....</b>	<b>139</b>
Глобальные, локальные, статические и динамические данные .....	139
Разделы объявления данных .....	139
NEW (выделить память из кучи) .....	140
DISPOSE (освободить память в куче) .....	141
<b>Шаблоны .....</b>	<b>142</b>
Числовые и денежные шаблоны .....	142
Шаблоны научной записи чисел .....	144
Шаблоны дат .....	145
Шаблоны времени .....	147
Шаблоны пользователя .....	148
Шаблоны редактирования строк .....	149
Строковые шаблоны .....	150
<b>Директивы компилятора .....</b>	<b>151</b>
EQUATE (присвоить метку) .....	151
SIZE (размер памяти в байтах) .....	152
<b>Глава 4 Выражения и операторы присваивания .....</b>	<b>153</b>
<b>Выражения .....</b>	<b>153</b>
Вычисление выражений .....	153
Арифметические операции .....	153
Логические операции .....	154

Числовые константы .....	155
Числовые выражения .....	156
Строковые константы .....	156
Операция конкатенации .....	157
Строковые выражения .....	157
Неявно объявленные строковые массивы и части строк .....	158
Логические выражения .....	159
<b>Строки динамических выражений .....</b>	<b>160</b>
BIND (объявить переменную для динамического выражения) .....	161
EVALUATE (получить результат динамического выражения) .....	163
POPBIND (восстановить пространство имен динамических выражений) .....	164
PUSHBIND (очистить пространство имен динамических выражений) .....	165
UNBIND (освободить логическое имя) .....	166
<b>Операторы присваивания .....</b>	<b>167</b>
Простые операторы присваивания .....	167
Вычисляющие операторы присваивания .....	168
Операторы множественного присваивания .....	168
Операторы присваивания указателей .....	170
CLEAR (очистить переменную) .....	171
<b>Правила преобразования типов данных .....</b>	<b>171</b>
Базовые типы .....	171
Двоично-десятичные операции и функции .....	173
Преобразование типов и промежуточные результаты .....	174
Преобразования типов данных при простых присваиваниях .....	175
<b>Глава 5 Управляющие структуры и операторы .....</b>	<b>183</b>
<b>Управляющие структуры .....</b>	<b>183</b>
CASE (структура условного выполнения) .....	183
EXECUTE (структура выбора оператора) .....	184
IF (структура условного выполнения) .....	186
LOOP (структура повторения) .....	187
<b>Управляющие операторы .....</b>	<b>189</b>
BREAK (прервать выполнение цикла) .....	189
CHAIN (выполнить другую программу) .....	190
CYCLE (переход в начало цикла) .....	191
DO (выполнить локальную подпрограмму) .....	192
EXIT (прекратить выполнение локальной подпрограммы) .....	192
GOTO (безусловный переход) .....	193
HALT (вернуться в DOS) .....	193
IDLE (включить периодически исполняемую процедуру) .....	194
RETURN (возврат в вызвавшую процедуру или функцию) .....	195
RUN (выполнить команду) .....	196

<i>SHUTDOWN</i> (включить процедуру завершения) .....	197
<i>STOP</i> (приостановить выполнение программы) .....	197

## **Глава 6 Окна и меню ..... 201**

### **Окна в Clarion .....201**

Обзор окон .....	201
Управляющие поля и “фокус ввода” .....	202
Метки соответствия полей .....	203

### **Структуры, описывающие окно .....204**

<i>APPLICATION</i> (объявить MDI окно) .....	204
<i>WINDOW</i> (объявить диалоговое окно) .....	209

### **Атрибуты структур *WINDOW* и *APPLICATION* .....215**

<i>ALRT</i> (установить “горячие клавиши”) .....	215
<i>AT</i> (установить положение и размеры окна) .....	215
<i>AUTO</i> (автоматическое обновление на экране значения <i>USE-переменной</i> ) .....	217
<i>CENTER</i> (центрировать окно) .....	217
<i>COLOR</i> (установить цвет окна) .....	218
<i>CURSOR</i> (установить форму курсора мыши) .....	219
<i>DOUBLE, NOFRAME, RESIZE</i> (установить для окна тип рамки) .....	220
<i>FONT</i> (установить для окна шрифт) .....	220
<i>GRAY</i> (установить фон для объемных полей) .....	221
<i>HLP</i> (установить для окна идентификатор диалоговой справки) .....	222
<i>ICON</i> (установить для окна пиктограмму) .....	223
<i>ICONIZE</i> (установить, что в момент раскрытия выводится пиктограмма) .....	224
<i>IMM</i> (немедленная генерация события при изменении размера окна) .....	225
<i>MASK</i> (установить режим ввода данных по шаблону) .....	226
<i>MAX</i> (установить наличие кнопки максимизации окна) .....	227
<i>MAXIMIZE</i> (установить раскрытие окна максимального размера) .....	227
<i>MDI</i> (установить для окна тип MDI) .....	228
<i>MODAL</i> (установить системную модальность окна) .....	228
<i>MSG</i> (установить сообщение на линейке состояния) .....	229
<i>PALETTE</i> (установить аппаратное количество цветов) .....	230
<i>STATUS</i> (установить наличие линейки состояния) .....	230
<i>SYSTEM</i> (установить наличие системного меню) .....	231
<i>TOOLBOX</i> (поведение инструментальной панели) .....	232
<i>TIMER</i> (установить генерацию периодических событий) .....	233

### **Структуры *MENUBAR* и *TOOLBAR* .....235**

<i>MENUBAR</i> (объявить структуру спускающегося меню) .....	235
<i>TOOLBAR</i> (объявить панель инструментов) .....	238

### **Атрибуты структур *MENUBAR* и *TOOLBAR* .....241**

<i>COLOR</i> (установить цвета для панели инструментов) .....	241
<i>CURSOR</i> (установить форму курсора мыши) .....	242

FONT (установить шрифт для панели инструментов .....	243
NOMERGE (установить возможность или невозможность слияния) .....	244
<b>Управляющие поля структуры MENUBAR .....</b>	<b>245</b>
MENU (объявить окно меню) .....	245
ITEM (объявить пункт меню) .....	247
<b>Атрибуты объектов в меню .....</b>	<b>249</b>
CHECK (объявить пункт меню - переключатель) .....	249
DISABLE (установить, что при открытии окна объект затушеван) .....	249
FIRST, LAST (установить положение окна-меню или пункта) .....	250
HLP (установить идентификатор справочной системы) .....	250
KEY (клавиша быстрого выполнения меню или пункта) .....	251
MSG (установить сообщение для меню или пункта) .....	252
RIGHT (установить положение меню) .....	253
SEPARATOR (установить в меню разделительную линию) .....	253
STD (установить стандартное действие для меню или пункта) .....	253
USE (задать для меню или пункта метку соответствия или переменную) .....	254
<b>Глава 7 Управляющие объекты в окнах .....</b>	<b>257</b>
<b>Объекты структур TOOLBAR и WINDOW .....</b>	<b>257</b>
BOX (объявить прямоугольную область внутри окна) .....	257
BUTTON (объявить кнопку) .....	258
CHECK (объявить кнопку с независимой фиксацией) .....	261
COMBO (объявить комбинированное окно списка) .....	263
ELLIPSE (объявить экранный объект эллиптической формы) .....	267
ENTRY (объявить поле для ввода данных) .....	269
GROUP (объявить группу экранных объектов) .....	272
IMAGE (объявить экранное поле, содержащее изображение) .....	274
LINE (экранный объект - прямая линия) .....	275
LIST (объявить окно списка) .....	276
OLE (объявить объект, содержащий объект OLE or .OCX) .....	281
OPTION (объявить группу кнопок с зависимой фиксацией) .....	284
PANEL (объявить область-панель в окне) .....	287
PROMPT (объявить поле - подсказку) .....	287
PROGRESS (объявить индикатор степени выполнения) .....	289
RADIO (объявить кнопку с зависимой фиксацией) .....	291
REGION (объявить объект - область в окне) .....	294
SHEET (объявить группу листов-закладок) .....	296
SPIN (объявить вращающийся список) .....	299
STRING (объявить экранное строковое поле) .....	302
TAB (объявить лист в структуре SHEET) .....	304
TEXT (объявить поле из нескольких строк для ввода данных) .....	306
VBX (объявить окно пользовательского объекта .VBX) .....	308

<b>Атрибуты экранных объектов .....</b>	<b>311</b>
ALRT (установить горячие клавиши для объекта) .....	311
ANGLE (установить угол под которым выводится объект) .....	312
AT (установить положение и размеры поля в окне) .....	312
AUTOSIZE (установить изменение размеров объекта OLE) .....	313
BEVEL (установить эффект объемности границ объекта) .....	313
BOXED (установить рамку вокруг группы экранных объектов) .....	314
CAP, UPR (установить регистр букв) .....	315
CLASS (установить класс объекта .VBX) .....	315
CLIP (установить усечение OLE-объекта) .....	315
COLOR (установить цвет экранного объекта) .....	316
COLUMN (установить полосу-курсор в окне списка) .....	317
COMPATIBILITY (установить режим совместимости объекта OLE) .....	317
CREATE (создать объект элемента управления OLE) .....	317
CURSOR (установить форму курсора мыши для объекта) .....	318
DEFAULT (установить экранную кнопку для клавиши ENTER) .....	319
DISABLE (при открытии окна элемент управления не действует) .....	319
DOCUMENT (создать объект из файла для элемента OLE) .....	319
DROP (установить поведение окна списка) .....	320
DRAGID (установить идентификатор источника) .....	320
DROPID (установить идентификатор поля-приемника данных) .....	321
FILL (установить цвет внутренней части объекта) .....	323
FONT (установить шрифт для поля) .....	323
FORMAT (установить структуру окна LIST или COMBO) .....	324
FROM (установить источник данных для окна списка) .....	328
FULL (установить расширение поля на весь экран) .....	329
GRID (установить цвет разделительных линий сетки в списке) .....	329
HIDE (установить, что поле не выводится при раскрытии окна) .....	330
HLP (установить идентификатор справочной системы) .....	330
HSCROLL, VSCROLL, HVSCROLL (наличие ленток скроллинга) .....	331
ICON (установить для поля пиктограмму) .....	331
IMM (установить немедленное генерирование события) .....	332
INS, OVR (установить режим ввода) .....	333
JOIN (установить объединение кнопок прокрутки на листе) .....	333
KEY (установить клавишу выполнения элемента) .....	333
LEFT, RIGHT, ABOVE, BELOW (установить положение листов) .....	334
LEFT, RIGHT, CENTER, DECIMAL (установить выравнивание) .....	335
LINEWIDTH (установить толщину линий экранных элементов) .....	337
LINK (создать элемент управления OLE - связь с объектом в файле) .....	337
MARK (установить режим множественного выбора) .....	338
MSG (установить сообщение для поля) .....	339
NOBAR (отсутствие полосы - курсора) .....	339
NOSHEET (задать "плавающий" лист) .....	339

<i>OPEN</i> (открыть из файла объект для поля OLE ) .....	339
<i>PASSWORD</i> (установить неотображаемое поле) .....	340
<i>RANGE</i> (установить границы диапазона) .....	340
<i>READONLY</i> (установить поле только для вывода данных) .....	341
<i>REQ</i> (установить обязательное поле) .....	341
<i>ROUND</i> (установить скругление углов у поля BOX) .....	341
<i>SCROLL</i> (установить прокручивающееся поле) .....	341
<i>SINGLE</i> (поле TEXT для ввода в одну строку) .....	342
<i>SKIP</i> (установить пропуск поля при выборе клавишей Tab) .....	342
<i>SPREAD</i> (установить равномерные промежутки между листами) .....	342
<i>STD</i> (установить стандартное действие) .....	343
<i>STEP</i> (установить приращение значения в поле SPIN) .....	344
<i>STRETCH</i> (растягивание объекта OLE) .....	344
<i>TIP</i> (установить текст “возникающей” подсказки) .....	344
<i>TRN</i> (установить вывод элемента на прозрачном фоне) .....	345
<i>UP, DOWN</i> (установить ориентацию ярлычков листов) .....	345
<i>USE</i> (задать для поля метку соответствия или переменную) .....	346
<i>VALUE</i> (значение, присваиваемое USE-переменной для RADIO или CHECK) .....	347
<i>VCR</i> (установить кнопки управления как у видеомаягнитофона) .....	348
<i>WIZARD</i> (листы не переключаются клавишей Tab) .....	349
<i>ZOOM</i> (пропорциональное изменение размеров объекта OLE) .....	349

## **Глава 8 Обработка событий ..... 351**

### ***Создание программ, выполняемых под управлением событий .....351***

<i>ACCEPT</i> (обработчик событий) .....	352
<i>ALERT</i> (установка клавиши, порождающей событие) .....	354
<i>EVENT</i> (возвратить номер события) .....	355
<i>POST</i> (послать событие определенное пользователем) .....	356
<i>YIELD</i> (разрешить обработку событий) .....	357

### ***Прикладные программы с несколькими процессами .....360***

<i>Процессы и многозадачность</i> .....	360
<i>Процессы и MDI</i> .....	360
<i>START</i> (возвращает новый процесс) .....	361
<i>THREAD</i> (выдать номер текущего процесса) .....	362
<i>UNLOCKTHREAD</i> (разблокировать обработку сообщений) .....	363
<i>LOCKTHREAD</i> (вновь заблокировать внешнюю обработку сообщений) .....	364
<i>THREADLOCKED</i> (текущее состояние внешней обработки сообщений) .....	365

### ***Процедуры работы с окнами .....366***

<i>CHANGE</i> (изменить значение поля) .....	366
<i>CLOSE</i> (закрывать окно) .....	366
<i>CREATE</i> (создать новое поле) .....	367
<i>DESTROY</i> (удалить экранный объект) .....	369

<i>DISABLE</i> (блокирует поле) .....	370
<i>DISPLAY</i> (отобразить USE-переменные на экран) .....	370
<i>ENABLE</i> (разрешает доступ к полю) .....	371
<i>ERASE</i> (очистить поля и USE-переменные) .....	372
<i>GETFONT</i> (получить информацию о шрифте) .....	372
<i>GETPOSITION</i> (получить информацию о расположении поля) .....	373
<i>HELP</i> (доступ к окну справки) .....	374
<i>HIDE</i> ("спрятать" поле) .....	375
<i>OPEN</i> (открыть окно для исполнения) .....	376
<i>SELECT</i> (выбор поля для последующей обработки) .....	376
<i>SET3DLOOK</i> (установить объемное изображение окна) .....	378
<i>SETCURSOR</i> (временно изменить курсор мыши) .....	379
<i>SETFONT</i> (установить шрифт) .....	381
<i>SETPOSITION</i> (определить новое положение поля) .....	382
<i>SETTARGET</i> (сделать окно или отчет текущим) .....	383
<i>UNHIDE</i> (изобразить спрятанное поле) .....	384
<i>UPDATE</i> (запись с экрана в USE-переменную) .....	384
<i>UPDATE</i> Обновляет все поля экрана .....	385
<b>Функции работы с Окнами</b> .....	<b>386</b>
<i>ACCEPTED</i> (указать выполненное поле) .....	386
<i>CHOICE</i> (указать относительное положение элемента) .....	386
<i>CONTENTS</i> (вернуть значение USE-переменной) .....	387
<i>FIELD</i> (указать поле, владеющее фокусом) .....	388
<i>FIRSTFIELD</i> (указать первое поле окна) .....	389
<i>FOCUS</i> (указать поле, владеющее фокусом) .....	389
<i>INCOMPLETE</i> (указать пустое REQ-поле) .....	390
<i>LASTFIELD</i> (указать последнее поле окна) .....	391
<i>MESSAGE</i> (изобразить окно сообщений) .....	391
<i>MOUSEX</i> (получить положение мыши по горизонтали) .....	393
<i>MOUSEY</i> (получить положение мыши по вертикали) .....	393
<i>POPUP</i> (получить выбор пользователя в спускающемся меню) .....	393
<i>SELECTED</i> (указать поле, владеющее фокусом ввода) .....	395
<b>Процедуры работы с клавиатурой</b> .....	<b>396</b>
<i>ALIAS</i> (перезаопределить код клавиши) .....	396
<i>ASK</i> (ждать нажатие клавиши) .....	396
<i>PRESS</i> (поместить в буфер строку символов) .....	397
<i>PRESSKEY</i> (поместить в буфер код нажатия клавиши) .....	397
<i>SETKEYCODE</i> (назначить код клавиши) .....	398
<b>Функции работы с клавиатурой</b> .....	<b>399</b>
<i>KEYBOARD</i> (возвратить код первый в буфере) .....	399
<i>KEYCHAR</i> (возвратить ASCII код) .....	399
<i>KEYCODE</i> (возвратить код последней клавиши) .....	400
<i>KEYSTATE</i> (возвратить состояние клавиатуры) .....	400

<b>Функции поддержки окон стандарта Windows .....</b>	<b>402</b>
COLRDIALOG (выбор цвета) .....	402
FILEDIALOG (выбор файла) .....	402
FONTDIALOG (выбор шрифта) .....	405
PRINTERDIALOG (выбор шрифта) .....	406
<b>Процессы Drag and Drop .....</b>	<b>407</b>
CLIPBOARD (получить содержимое буфера обмена Windows) .....	407
DRAGID (возвратить совпавший “потащил-отпустил” ярлык) .....	408
DROPID (возвратить “потащил-отпустил”-строку) .....	409
SETCLIPBOARD (положить информацию в Windows-буфер обмена) .....	411
SETDROPID (задать строку, возвращаемую DROPID) .....	412
<b>Поддержка INI-файлов .....</b>	<b>413</b>
GETINI (возвратить элемент INI-файла) .....	413
PUTINI (поместить элемент в INI-файл) .....	414
<b><u>Глава 9. Создание печатных документов .....</u></b>	<b><u>417</u></b>
<b>Документы в Windows .....</b>	<b>417</b>
Переход на новую страницу .....	417
<b>Процедуры работы с документом .....</b>	<b>418</b>
REPORT (объявить структуру документа) .....	418
AT (назначение области печати тела документа) .....	421
FONT (назначение для документа шрифта по умолчанию) .....	422
PRE (назначение префикса для меток структур документа) .....	423
PREVIEW (направить вывод документа в метафайлы) .....	424
PAPER (установить размер бумаги для отчета) .....	425
LANDSCAPE (выбор ориентации страницы) .....	426
COLOR (установить цвет фона отчета) .....	427
THOUS, MM, POINTS (выбор единицы измерений координат документа) .....	427
<b>Структуры, образующие документ .....</b>	<b>428</b>
BREAK (объявление структуры раздела документа) .....	428
DETAIL (объявить структуру документа) .....	429
FOOTER (структура нижнего колонтитула страницы или раздела) .....	431
FORM (структура трафарета страницы) .....	432
HEADER (верхний колонтитул страницы или раздела) .....	433
<b>Атрибуты Структур Документа .....</b>	<b>436</b>
ABSOLUTE (печатать с фиксированной позиции) .....	436
ALONE (печатать с фиксированной позиции) .....	436
AT (указание области страницы для печати структуры) .....	437
COLOR (установить цвет фона) .....	438
FONT (назначение для документа шрифта по умолчанию) .....	439
PAGEAFTER (затем - переход на новую страницу) .....	440
PAGEBEFORE (сначала - переход на новую страницу) .....	441



<i>USE (определить метку соответствия структуры)</i> .....	442
<i>WITHNEXT (предотвратить отрыв от последующих)</i> .....	443
<i>WITHPRIOR (предотвратить отрыв от предыдущих)</i> .....	444
<b>Управляющие поля в структуре REPORT</b> .....	<b>446</b>
<i>BOX (объявить поле окна в структуре документа)</i> .....	446
<i>CHECK (объявить поле флагов в структуре документа)</i> .....	447
<i>VBX (объявить в документе импортируемое.VBX-поле)</i> .....	448
<i>ELLIPSE (объявить эллипс-поле в документа)</i> .....	449
<i>GROUP (объявить группу полей документа)</i> .....	450
<i>IMAGE (объявить в документе графическое поле)</i> .....	451
<i>LINE (объявить в документа поле линии)</i> .....	451
<i>LIST (объявить в документе поле списка)</i> .....	452
<i>OPTION (объявить группу RADIO-полей документа)</i> .....	454
<i>RADIO (объявить поле radio-кнопки в документе)</i> .....	455
<i>STRING (объявить в документе поле строки)</i> .....	456
<i>TEXT (объявить многострочное текстовое поле)</i> .....	458
<b>Атрибуты элементов в отчетах</b> .....	<b>460</b>
<i>ANGLE (установить угол печати элемента)</i> .....	460
<i>AT (установить местоположение и размер поля в документе)</i> .....	460
<i>AVE (установить итоговое среднее)</i> .....	461
<i>BOXED (установить рамку для группы полей документа)</i> .....	463
<i>CAP,UPR (установить регистр вывода символа на печать)</i> .....	463
<i>CNT (установить подсчет итога)</i> .....	463
<i>COLOR (установить цвет печати)</i> .....	464
<i>FILL (установить при выводе на печать цвет заполнения)</i> .....	465
<i>FONT (установить шрифт по умолчанию)</i> .....	465
<i>FORMAT (установить формат печати поля LIST)</i> .....	466
<i>FROM (установить источник данных поля списка)</i> .....	469
<i>HIDE (запретить вывод поля на печать)</i> .....	469
<i>LEFT, RIGHT, CENTER, DECIMAL (установить выравнивание)</i> .....	470
<i>LINEWIDTH (установить толщину линии элемента)</i> .....	471
<i>MAX (установить итоговый максимум)</i> .....	471
<i>META (печатать .VBX как .WMF)</i> .....	472
<i>MIN (установить итоговый минимум)</i> .....	472
<i>PAGE (установить сброс страничных итогов)</i> .....	473
<i>PAGENO (установить печать номера страницы)</i> .....	474
<i>RESET (установить сброс итогов)</i> .....	474
<i>RESIZE (переменная высота элемента TEXT)</i> .....	474
<i>SKIP (условная печать поля STRING или TEXT)</i> .....	475
<i>SUM (установить итоговое значение)</i> .....	475
<i>TALLY (установить общее число вычислений)</i> .....	476
<i>TRN (установить "прозрачность" элемента документа)</i> .....	477
<i>USE (задать имя для ссылки из программы)</i> .....	478

<b>Процедуры отчета .....</b>	<b>480</b>
CLOSE (закрыть структуру REPORT) .....	480
ENDPAGE (форсировать переход на новую страницу) .....	480
OPEN (открыть структуру REPORT для работы) .....	482
PRINT (напечатать структуру) .....	482

## **Глава 10 Графические команды ..... 483**

<b>Предисловие .....</b>	<b>483</b>
Текущий объект .....	483
Графика и система координат .....	483
<b>Графические процедуры .....</b>	<b>484</b>
ARC (нарисовать дугу эллипса) .....	484
BLANK (стереть графические изображения) .....	485
BOX (нарисовать прямоугольник) .....	485
CHORD (нарисовать сектор эллипса) .....	486
ELLIPSE (нарисовать эллипс) .....	487
IMAGE (нарисовать эллипс) .....	488
LINE (изобразить прямую линию) .....	489
PIE (изобразить секторную диаграмму) .....	490
POLYGON (изобразить многозвенную фигуру) .....	491
ROUNDBOX (нарисовать прямоугольник с округлыми углами) .....	492
SETPENCOLOR (установить цвет линий) .....	493
SETPENSTYLE (установить тип линий) .....	494
SETPENWIDTH (установить толщину линий) .....	495
SHOW (отобразить на экран) .....	496
TYPE (вывести строку на экран) .....	496
<b>Графические функции .....</b>	<b>497</b>
PENCOLOR (возвратить цвет линии) .....	497
PENSTYLE (возвратить тип линии) .....	498
PENWIDTH (возвратить толщину линии) .....	499

## **Глава 11 Файлы данных ..... 501**

<b>Структуры для работы с файлами данных .....</b>	<b>501</b>
FILE (объявить структуру файла данных) .....	501
CREATE (разрешить создание файла) .....	503
DRIVER (указать тип файловой системы) .....	503
NAME (задать имя файла) .....	504
ENCRYPT (шифрование файла данных) .....	505
OWNER (объявить пароль для шифрования данных) .....	505
RECLAIM (использовать пространство удаленных записей) .....	506
PRE (задать префикс для переменных структуры) .....	506
BINDABLE (переменная, используемая в динамических выражениях) .....	507

<i>THREAD</i> (отдельный буфер записи для каждого исполняемого процесса) .....	508
<i>EXTERNAL</i> (файл объявлен во внешнем модуле) .....	509
<i>DLL</i> (процедура определена внешне, в библиотеке <i>DLL</i> ) .....	511
<i>OEM</i> (установить поддержку международной кодировки) .....	512
<b>Операторы, описывающие структуру файла .....</b>	<b>514</b>
<i>INDEX</i> (объявить статический ключ доступа к записям файла) .....	514
<i>KEY</i> (объявить динамический ключ доступа к записям файла) .....	515
<i>MEMO</i> (объявить текстовое поле) .....	516
<i>BLOB</i> (объявить тето-поле переменной длины) .....	517
<i>RECORD</i> (объявить структуру записи файла) .....	519
<b>Атрибуты операторов <i>INDEX</i>, <i>KEY</i> и <i>MEMO</i> .....</b>	<b>520</b>
<i>BINARY</i> (тето-поле содержит двоичные данные) .....	520
<i>DUP</i> (допускается повторение значений ключа) .....	520
<i>NOCASE</i> (независимость ключа или индекса от регистра букв) .....	521
<i>OPT</i> (исключить нулевые значения ключа или индекса) .....	521
<i>PRIMARY</i> (установить первичный ключ) .....	522
<i>NAME</i> (задать внешнее имя) .....	522
<b>Команды для работы с файлами .....</b>	<b>524</b>
<i>BUILD</i> (построить ключ или индекс) .....	524
<i>CLOSE</i> (закрыть файл данных) .....	525
<i>COPY</i> (скопировать файл данных) .....	526
<i>CREATE</i> (создать пустой файл данных) .....	527
<i>EMPTY</i> (очистить файл данных) .....	527
<i>FLUSH</i> (записать на диск буферы <i>DOS</i> ) .....	528
<i>LOCK</i> (заблокировать файл) .....	528
<i>OPEN</i> (открыть файл данных) .....	529
<i>PACK</i> (исключить удаленные записи) .....	530
<i>REMOVE</i> (удалить файл данных) .....	531
<i>RENAME</i> (изменить имя файла и/или каталог) .....	532
<i>SHARE</i> (открыть файл данных) .....	532
<i>STATUS</i> (получить состояние файла) .....	534
<i>STREAM</i> (включить буферизацию) .....	534
<i>UNLOCK</i> (разблокировать файл данных) .....	535
<b>Команды для работы с записями .....</b>	<b>536</b>
<i>ADD</i> (добавить новую запись) .....	536
<i>APPEND</i> (добавить новую запись) .....	536
<i>DELETE</i> (удалить запись) .....	537
<i>GET</i> (прочитать запись с помощью прямого доступа) .....	538
<i>HOLD</i> (исключительный доступ к записи) .....	540
<i>NEXT</i> (прочитать следующую запись) .....	541
<i>NOMEMO</i> (включить режим выборки без <i>MEMO</i> -полей) .....	542
<i>PREVIOUS</i> (прочитать предыдущую запись) .....	542

PUT (записать данные обратно в файл) .....	543
RELEASE (освободить захваченную запись) .....	544
REGEX (повторно прочитать запись файла) .....	545
RESET (восстановить положение в последовательности записей) .....	546
SET (начать последовательную обработку файла) .....	547
SKIP (пропустить запись при последовательной обработке) .....	549
WATCH (автоматическая проверка совместного использования) .....	550
<b>Функции для работы с файлами .....</b>	<b>551</b>
BOF (сигнализировать о достижении начала файла) .....	551
BYTES (получить размер файла в байтах) .....	551
DUPLICATE (проверить на повторение значения ключа) .....	552
EOF (сигнализировать о достижении конца файла) .....	553
NAME (получить имя файла в DOS или имя устройства) .....	554
POINTER (получить относительное положение записи) .....	554
POSITION (получить положение в последовательности обработки файла) .....	555
RECORDS (получить число записей или значений ключа) .....	556
SEND (послать сообщение файловому драйверу) .....	557
<b>Обработка транзакций .....</b>	<b>558</b>
Определение транзакции .....	558
Границы транзакции .....	559
LOGOUT, COMMIT, ROLLBACK .....	560
Вопросы многопользовательской работы .....	560
COMMIT (завершить успешную транзакцию) .....	561
LOGOUT (начать транзакцию) .....	562
ROLLBACK (завершить неуспешную транзакцию) .....	563
Обработка фиктивных данных .....	564
NULL (проверить значение поля на фиктивное) .....	565
SETNULL (установить для поля фиктивное значение) .....	566
SETNONULL (установить нефиктивное значение поля) .....	568
<b>Настройка специфики, связанной со страной .....</b>	<b>570</b>
Файлы переменных среды .....	570
CONVERTANSITOOEM (преобразовать код ANSI в ASCII) .....	573
CONVERTOEMTOANSI (преобразовать код ASCII в ANSI) .....	574
ISALPHA (проверить символ на букву) .....	575
ISLOWER (проверить символ на строчную букву) .....	576
ISUPPER (проверить символ на прописную букву) .....	576
LOCALE (загрузить файл с переменными среды) .....	577
<b><u>Глава 12 Виртуальные файлы .....</u></b>	<b><u>579</u></b>
<b>Структуры для организации виртуального файла .....</b>	<b>579</b>
VIEW (объявить “виртуальный” файл) .....	579
FILTER (установить ограничивающее выражение) .....	582

ORDER (выражение, определяющее порядок сортировки) .....	583
PROJECT (задать поля виртуального файла) .....	584
JOIN (объявить операцию “соединения”) .....	585
<b>Операторы для работы с виртуальным файлом .....</b>	<b>588</b>
CLOSE (закрыть виртуальный файл) .....	588
OPEN (открыть виртуальный файл) .....	589
DELETE (удалить запись первичного в структуре VIEW файла) .....	590
HOLD (исключительный доступ к записи виртуального файла) .....	591
NEXT (прочитать следующую запись виртуального файла) .....	593
NOMEMO (читать записи, не считывая MEMO-поля) .....	594
PREVIOUS (прочитать предыдущую запись виртуального файла) .....	595
PUT (занести запись обратно в первичный файл) .....	596
REGET (повторно прочитать запись) .....	598
RELEASE (освободить запись виртуального файла) .....	600
RESET (восстановить положение в последовательности записей) .....	601
SET (последовательная обработка виртуального файл) .....	602
SKIP (пропустить записи в виртуальном файле) .....	604
WATCH (автоматическая проверка совместного использования) .....	605
<b>Функции для работы с виртуальными файлами .....</b>	<b>607</b>
POSITION (получить идентификатор положения в виртуальном файле) .....	607
<b>Глава 13 Очереди в памяти. ....</b>	<b>609</b>
<b>Структура QUEUE .....</b>	<b>609</b>
QUEUE (объявить структуру QUEUE) .....	609
PRE (задать префикс для переменных структуры) .....	611
STATIC (статическая локальная очередь) .....	612
THREAD (отдельный буфер очереди для каждого процесса) .....	612
NAME (задать внешнее имя для переменной в очереди) .....	613
TYPE (определение типа для очереди) .....	613
BINDABLE (использование в динамических выражениях) .....	614
EXTERNAL (очередь объявлена вне данной программы) .....	615
DLL (очередь объявлена внешне, в библиотеке DLL) .....	616
<b>Операторы работы с очередью .....</b>	<b>618</b>
ADD (добавить элемент в очередь) .....	618
DELETE (удалить элемент очереди) .....	619
FREE (удалить все элементы очереди) .....	620
GET (прочитать элемент очереди) .....	620
PUT (записать элемент в очередь) .....	622
SORT (упорядочить очередь) .....	623
<b>Функции для работы с очередью .....</b>	<b>625</b>
POINTER (получить относительный номер) .....	625
RECORDS (получить число элементов в очереди) .....	625

<b>Глава 14 Прочие операторы и функции .....</b>	<b>627</b>
<b>Математические функции .....</b>	<b>627</b>
ABS (получить абсолютную величину) .....	627
INRANGE (проверить попадание в диапазон) .....	627
INT (взять целую часть) .....	628
LOGE (вычислить натуральный логарифм) .....	628
LOG10 (вычислить десятичный логарифм) .....	629
RANDOM (получить случайное число) .....	629
ROUND (округлить число) .....	630
SQRT (квадратный корень) .....	630
<b>Тригонометрические функции .....</b>	<b>631</b>
SIN (синус) .....	631
COS (косинус) .....	632
TAN (тангенс) .....	632
ASIN (арксинус) .....	633
ACOS (арккосинус) .....	633
ATAN (арктангенс) .....	634
<b>Строковые функции .....</b>	<b>634</b>
ALL (повторение символов) .....	634
CENTER (центрировать строку) .....	635
CHR (получить символ ASCII) .....	635
CLIP (отсечь пробелы в конце строки) .....	636
DEFORMAT (исключить форматирование из числовой строки) .....	636
FORMAT (форматировать число по шаблону) .....	637
INLIST (искать элемент в списке) .....	637
INSTRING (искать вхождение строки) .....	638
LEFT (выровнять строку влево) .....	639
LEN (получить длину строки) .....	639
LOWER (преобразовать в строчные буквы) .....	640
NUMERIC (проверить числовую строку) .....	640
RIGHT (выровнять строку вправо) .....	641
SUB (получить подстроку) .....	641
UPPER (преобразовать в прописные буквы) .....	642
VAL (получить код символа ASCII) .....	642
<b>Функции манипуляций с битами .....</b>	<b>643</b>
BAND (поразрядное И) .....	643
BOR (поразрядное ИЛИ) .....	644
BXOR (поразрядное исключающее ИЛИ) .....	645
BSHIFT (поразрядный сдвиг) .....	646
<b>Операторы и функции даты и времени .....</b>	<b>647</b>
Стандартная дата .....	647

Стандартное время .....	647
TODAY () (получить системную дату) .....	648
SETTODAY (установить системную дату) .....	648
CLOCK (получить системное время) .....	648
SETCLOCK (установить системное время) .....	649
DATE (получить стандартное время) .....	649
DAY (получить день месяца) .....	650
MONTH (получить месяц) .....	650
YEAR (получить год) .....	651
AGE (получить возраст в заданный день) .....	651
<b>Операторы и функции обращения к DOS .....</b>	<b>652</b>
COMMAND (получить параметр командной строки) .....	652
DIRECTORY (получить список файлов в каталоге) .....	653
PATH (получить текущий каталог DOS) .....	655
RUNCODE (получить код завершения из DOS) .....	655
SETCOMMAND (задать параметры командной строки) .....	656
SETPATH (изменить текущий диск и каталог) .....	657
<b>Функции обработки ошибочных ситуаций .....</b>	<b>658</b>
ERROR (получить текст сообщения об ошибке) .....	658
ERRORCODE (получить код ошибки) .....	658
ERRORFILE (получить имя файла, вызвавшего ошибку) .....	658
FILEERROR (получить сообщение об ошибке от файлового драйвера) .....	659
FILEERRORCODE (получить код ошибки от файлового драйвера) .....	659
REJECTCODE (получить код причины события EVENT:Rejected) .....	660
<b>Другие операторы и функции .....</b>	<b>661</b>
ADDRESS (получить адрес памяти) .....	661
BEEP (подать звуковой сигнал) .....	662
CALL (обратиться к процедуре из DLL) .....	662
CHOOSE (получить выбранное значение) .....	663
MAXIMUM (получить максимальное значение индекса) .....	664
OMITTED (проверить не опущены ли параметры) .....	665
PEEK (прочитать данные из памяти) .....	666
POKE (записать данные в память) .....	667
<b>Приложение А. Библиотека DDE, OLE, и .OCX .....</b>	<b>669</b>
<b>Динамический Обмен Данными .....</b>	<b>669</b>
Введение в DDE .....	669
DDE События .....	670
<b>DDE Функции .....</b>	<b>671</b>
DDESERVER (возвратить номер канала DDE- сервера) .....	671
DDECLIENT (возвратить номер канала DDE-клиента) .....	672
DDEQUERY (просмотр зарегистрированных DDE-серверов) .....	673

<i>DDECHANNEL (получить номер DDE-канала)</i> .....	674
<i>DDEAPP (получить имя сервер-приложения)</i> .....	675
<i>DDEITEM (получить имя элемента данных сервера)</i> .....	676
<i>DDETOPIC (получить имя раздела сервера)</i> .....	677
<i>DDEVALUE (получить значение данных, посланных серверу)</i> .....	678
<b>DDE Процедуры</b> .....	<b>679</b>
<i>DDERead (получить данные от DDE-сервера)</i> .....	679
<i>DDEWRITE (предоставить данные DDE-клиенту)</i> .....	680
<i>DDEEXECUTE (послать команду DDE серверу)</i> .....	682
<i>DDEPOKE (послать неиспрашиваемые данные DDE-серверу)</i> .....	684
<i>DDECLOSE (завершить диалог с DDE-сервером)</i> .....	685
<b>Связывание и внедрение объектов</b> .....	<b>686</b>
<i>Введение в OLE</i> .....	686
<i>Свойства-атрибуты</i> .....	687
<i>Необъявленные свойства</i> .....	688
<i>OLEDIRECTORY (получить список установленных OLE/OCX)</i> .....	694
<b>Пользовательские объекты OLE (.OCX)</b> .....	<b>696</b>
<i>Введение</i> .....	696
<i>Свойства объектов .OCX</i> .....	696
<i>Оконные функции</i> .....	697
<i>Оконная функция - обработчик событий OCX</i> .....	698
<b>Библиотечные функции OCX</b> .....	<b>703</b>
<i>OCXREGISTERPROPEDIT (установить контроллер свойств)</i> .....	703
<i>OCXREGISTERPROPCHANGE (установить функцию - редактор)</i> .....	703
<i>OCXREGISTEREVENTPROC (установить процедуру обработки событий)</i> .....	703
<i>OCXUNREGISTERPROPEDIT (отменить функцию - контроллер свойств)</i> .....	704
<i>OCXUNREGISTERPROPCHANGE (отменить редактор свойств)</i> .....	704
<i>OCXUNREGISTEREVENTPROC (отменить процедуру обработки событий)</i> .....	705
<i>OCXGETPARAMCOUNT (получить число параметров для события)</i> .....	705
<i>OCXGETPARAM (получить строку параметра)</i> .....	706
<i>OCXSETPARAM (установить строку параметра)</i> .....	707
<i>OCXLOADIMAGE (получить графический объект)</i> .....	708
<b>Приложение В Коды клавиш</b> .....	<b>709</b>
<b>Коды клавиш в Clarion</b> .....	<b>709</b>
<i>Формат кода клавиш в Windows</i> .....	709
<i>KEYCODES.CLW</i> .....	709
<b>Приложение С Присвоение значений свойствам</b> .....	<b>725</b>
<b>Присвоение значений свойствам структур данных</b> .....	<b>725</b>
<i>Встроенные переменные</i> .....	725
<i>Имена свойств</i> .....	726



Мнемонические имена свойств атрибутов .....	728
Свойства строки форматирования окна списка .....	733
<b>Прочие свойства .....</b>	<b>736</b>
Свойства, связанные с действиями мышью в окне списка .....	736
Свойства, относящиеся к печати .....	737
Необъявленные свойства .....	740
<b>Свойства структур VIEW и FILE .....</b>	<b>772</b>
Свойства - атрибуты .....	772
Необъявляемые свойства структур VIEW и FILE .....	773
Встроенный SQL .....	774
<b><u>Приложение Д. Сообщения об ошибках .....</u></b>	<b><u>777</u></b>
<b>Ошибки времени выполнения .....</b>	<b>777</b>
Ошибки времени выполнения, не обнаруживаемые в программе .....	780
<b>Ошибки компиляции .....</b>	<b>781</b>
Особенные ошибки .....	782
Ошибки неопределенного характера .....	796
<b><u>Приложение Е. Мнемонические имена событий .....</u></b>	<b><u>797</u></b>
<b>События .....</b>	<b>797</b>
События не связанные с полем .....	797
События связанные с полем .....	801

# Индекс

## A

ABS	605
ABOVE	312
ABSOLUTE	414
ACCEPT	330
ACCEPTED	364
ACOS	611
ADD	514, 596
ADDRESS	639
AGE	629
ALERT	332
ALIAS	374
ALL	612
ALONE	414
ALRT	193, 289
ANGLE	290, 438
APPEND	514
APPLICATION	182
ARC	462
ASIN	611
ASK	374
AT	193, 290, 399, 415, 438
ATAN	612
AUTO	115, 195
AUTOSIZE	291
AVE	439

## B

BAND	621
BEEP	640
BEGIN	69
BELOW	312
BEVEL	291
BFLOAT4	82
BFLOAT8	83
BINARY	498
BIND	139
BINDABLE	114, 485, 592
BLANK	463
BLOB	495
BOF	529

BOR	622
BOX	235, 424, 463
BOXED	292, 441
BREAK	167, 406
BSHIFT	624
BUILD	502
BUTTON	236
BXOR	623
BYTE	75
BYTES	529

## C

CALL	640
CAP	293, 441
CASE	161
CENTER	195, 313, 448, 613
CHAIN	168
CHANGE	344
CHECK	227, 239, 425
CHOICE	364
CHOOSE	641
CHORD	464
CHR	613
CLASS	97, 293
CLEAR	149
CLIP	293, 614
CLIPBOARD	385
CLOCK	626
CLOSE	344, 458, 503, 566
CNT	441
CODE	44
COLOR	196, 219, 294, 405, 416, 442
COLRDIALOG	380
COLUMN	295
COMBO	241
COMMAND	630
COMMIT	538, 539
COMPATIBILITY	295
COMPILE	70
CONTENTS	366
CONVERTANSITO OEM	551

CONVERTOEMTOANSI 552  
 COPY 504  
 COS 610  
 CREATE 295, 345, 481, 505  
 CSTRING 88  
 CURSOR 197, 220, 296  
 CYCLE 169

**D**

DEFORMAT 614  
 DATE 627  
 DATE 92  
 DAY 628  
 DDE 647  
 DDEAPP 653  
 DDECHANNEL 652  
 DDECLIENT 650  
 DDECLOSE 663  
 DDEEXECUTE 660  
 DDEITEM 654  
 DDEPOKE 662  
 DDEQUERY 651  
 DDEREAD 657  
 DDESERVER 649  
 DDETOPIC 655  
 DDEVALUE 656  
 DDEWRITE 658  
 DECIMAL 84, 313, 448  
 DEFAULT 297  
 DELETE 515, 568, 597  
 DESTROY 347  
 DETAIL 407  
 DIM 108  
 DIRECTORY 631  
 DISABLE 227, 297, 348  
 DISPLAY 348  
 DISPOSE 119  
 DLL 64, 110, 489, 594  
 DO 170  
 DOCUMENT 297  
 DOUBLE 198  
 DOWN 323  
 DRAGID 298, 386  
 DRIVER 481  
 DROP 298  
 DROPID 299, 387  
 DUP 498

DUPLICATE 530

**E**

ELLIPSE 245, 427, 465  
 EMPTY 505  
 ENABLE 349  
 ENCRYPT 483  
 END 45  
 ENDPAGE 458  
 ENTRY 247  
 EOF 531  
 EQUATE 129  
 ERASE 350  
 ERROR 636  
 ERRORCODE 636  
 ERRORFILE 636  
 EVALUATE 141  
 EVENT 333, 775  
 EXECUTE 162  
 EXIT 170  
 EXTERNAL 109, 487, 593

**F**

FIELD 366  
 FILE 479  
 FILEDIALOG 380  
 FILEERROR 637  
 FILEERRORCODE 637  
 FILL 301, 443  
 FILTER 560  
 FORMAT 615  
 FIRST 228  
 FIRSTFIELD 367  
 FLUSH 506  
 FOCUS 367  
 FONT 198, 221, 301, 400, 417, 443  
 FONTDIALOG 383  
 FOOTER 409  
 FORM 410  
 FORMAT 302, 444  
 FREE 598  
 FROM 306, 447  
 FULL 307  
 FUNCTION 42, 49

**G**

GET 516, 598  
 GETFONT 350  
 GETINI 391  
 GETPOSITION 351  
 GOTO 171  
 GRAY 199  
 GRID 307  
 GROUP 94, 250, 428

**H**

HALT 171  
 HEADER 411  
 HELP 352  
 HIDE 308, 353, 447  
 HLP 200, 228, 308  
 HOLD 518, 569  
 HSCROLL 309  
 HVSCROLL 309

**I**

ICON 201, 309  
 ICONIZE 202  
 IDLE 172  
 IF 164  
 IMAGE 252, 429, 466  
 IMM 203, 310  
 INCLUDE 71  
 INCOMPLETE 368  
 INDEX 492  
 INLIST 615  
 INT 606  
 INRANGE 605  
 INS 311  
 INSTRING 616  
 ISALPHA 553  
 ISLOWER 554  
 ISUPPER 554  
 ITEM 225

**J**

JOIN 311, 563

**K**

KEY 229, 311, 493

KEYBOARD 377  
 KEYCHAR 377  
 KEYCODE 378  
 KEYCODES.CLW 687  
 KEYSTATE 378

**L**

LEFT 617  
 LEN 617  
 LANDSCAPE 404  
 LAST 228  
 LASTFIELD 369  
 LEFT 312, 313, 448  
 LOG10 607  
 LOGE 606  
 LIKE 102  
 LINE 253, 429, 467  
 LINEWIDTH 315, 449  
 LONG 78  
 LINK 315  
 LOOP 165  
 LIST 254, 430  
 LOWER 618  
 LOCALE 555  
 LOCK 506  
 LOCKTHREAD 342  
 LOGOUT 538, 540

**M**

MAP 38  
 MARK 316  
 MASK 204  
 MAX 205, 449  
 MAXIMIZE 205  
 MAXIMUM 642  
 MDI 206  
 MEMBER 36  
 MEMO 494  
 MENU 223  
 MENUBAR 213  
 MESSAGE 369  
 META 450  
 MIN 450  
 MM 405  
 MODAL 206  
 MODULE 39, 116  
 MONTH 628

MOUSEX 371  
 MOUSEY 371  
 MSG 207, 230, 317

**N**

NAME 62, 111, 482, 500, 532, 591  
 NEW 118  
 NEXT 519, 571  
 NOBAR 317  
 NOCASE 499  
 NOFRAME 198  
 NOMEMO 520, 572  
 NOMERGE 222  
 NOSHEET 317  
 NUMERIC 618  
 NULL 543

**O**

OCX 674  
 OCXGETPARAM 684  
 OCXGETPARAMCOUNT 683  
 OCXLOADIMAGE 686  
 OCXREGISTEREVENTPROC 681  
 OCXREGISTERPROPCHANGE 681  
 OCXREGISTERPROPEDIT 681  
 OCXSETPARAM 685  
 OCXUNREGISTEREVENTPROC 683  
 OCXUNREGISTERPROPCHANGE 682  
 OCXUNREGISTERPROPEDIT 682  
 OEM 490  
 OLE 259, 664  
 OLEDIRECTORY 672  
 OMIT 71  
 OMITTED 643  
 OPEN 317, 354, 460, 507, 567  
 OPT 499  
 OPTION 262, 432  
 ORDER 561  
 OVER 112  
 OVR 311  
 OWNER 483

**P**

PACK 508  
 PAGE 451  
 PAGEAFTER 418

PAGEBEFORE 419  
 PAGENO 452  
 PALETTE 208  
 PANEL 265  
 PAPER 403  
 PASSWORD 318  
 PATH 633  
 PDECIMAL 85  
 PEEK 644  
 PENCOLOR 475  
 PENSTYLE 476  
 PENWIDTH 477  
 PIE 468  
 POINTER 532, 603  
 POINTS 405  
 POKE 645  
 POLYGON 469  
 POPBIND 142  
 POPUP 371  
 POSITION 533, 585  
 POST 334  
 PRE 107, 401, 484, 589  
 PRESS 375  
 PRESSKEY 375  
 PREVIEW 402  
 PREVIOUS 520, 573  
 PRIMARY 500  
 PRINT 460  
 PRINTERDIALOG 384  
 PRIVATE 64, 116  
 PROC 64  
 PROCEDURE 40, 49  
 PROGRAM 34  
 PROGRESS 267  
 PROJECT 562  
 PSTRING 90  
 PUSHBIND 143  
 PUT 521, 574, 600  
 PUTINI 392

**Q**

QUEUE 587

**R**

RADIO 269, 434  
 RANDOM 607  
 RANGE 318

RAW 62  
 READONLY 319  
 REAL 81  
 RECLAIM 484  
 RECORD 497  
 RECORDS 534, 603  
 REGET 523, 576  
 REGION 272  
 REJECTCODE 638  
 RELEASE 522, 578  
 REMOVE 509  
 RENAME 510  
 REPORT 396  
 REQ 319  
 RESET 452, 524, 579  
 RESIZE 198, 452  
 RETURN 173  
 RIGHT 231, 312, 313, 448  
 RIGHT 619  
 ROUND 608  
 ROLLBACK 538, 541  
 ROUND 319  
 ROUNDBOX 470  
 ROUTINE 44  
 RUN 174  
 RUNCODE 633

## S

SETCLOCK 627  
 SETTODAY 626  
 SCROLL 319  
 SECTION 72  
 SELECT 354  
 SELECTED 373  
 SEND 535  
 SEPARATOR 231  
 SET 525, 580  
 SET3DLOOK 356  
 SETCLIPBOARD 389  
 SETCOMMAND 634  
 SETCURSOR 357  
 SETDROPID 390  
 SETFONT 359  
 SETKEYCODE 376  
 SETNONULL 546  
 SETNULL 544

SETPATH 635  
 SETPENCOLOR 471  
 SETPENSTYLE 472  
 SETPENWIDTH 473  
 SETPOSITION 360  
 SETTARGET 361  
 SHARE 510  
 SHEET 274  
 SHORT 76  
 SHOW 474  
 SHUTDOWN 175  
 SIN 609  
 SINGLE 320  
 SIZE 130  
 SKIP 320, 453, 527, 582  
 SORT 601  
 SPIN 277  
 SPREAD 320  
 SQRT 608  
 SREAL 80  
 START 339  
 STATIC 113, 590  
 STATUS 208, 512  
 STD 231, 321  
 STEP 322  
 STOP 175  
 STREAM 512  
 STRETCH 322  
 STRING 86, 280, 434  
 SUB 619  
 SUM 453  
 SYSTEM 209

## T

TAB 282  
 TALLY 454  
 TAN 610  
 TEXT 284, 436  
 THOUS 405  
 THREAD 114, 340, 486, 590  
 THREADLOCKED 343  
 TIME 93  
 TIMER 211  
 TIP 322  
 TODAY () 626  
 TOOLBAR 216  
 TOOLBOX 210

TRN	323, 455
TYPE	63, 115, 474, 591

**U**

UPPER	620
ULONG	79
UNBIND	144
UNHIDE	362
UNLOCK	513
UNLOCKTHREAD	341
UP	323
UPDATE	362, 363
UPR	293, 441
USE	232, 324, 420, 456
USHORT	77

**V**

VAL	620
VALUE	325
VBX	286, 426
VCR	326
VIEW	557
VIRTUAL	65
VSCROLL	309

**W**

WATCH	528, 583
WINDOW	187
WITHNEXT	421
WITHPRIOR	422
WIZARD	327

**Y**

YEAR	629
YIELD	335

**Z**

ZOOM	327
------	-----