Обзор библиотеки ABC

О чем эта часть

Данная часть книги посвящена описанию библиотеки Классов ABC(Application Builder Class).

Здесь приводятся описания каждого класса или соответствующей группы классов. Также вы найдете конкретную информацию об общих свойствах и методах каждого класса с примерами их использования. Здесь приведены файлы с исходным кодом для каждого класса и описаны некоторые отношения между классами.

Библиотека Классов (ABC)

Общие сведения о библиотеках классов

Назначение библиотеки классов в объектно-ориентированной системе заключается в том, чтобы повысить отдачу труда разработчика, дав ему возможность надежного и эффективного многократного использования фрагментов программного кода. Другими словами, библиотека классов должна облегчить программисту задачу написания процедур, решающих конкретные прикладные проблемы, позволить им использовать ранее созданные процедуры, выполняющие общие или повторяющиеся задачи.

Кроме того, библиотека классов может заметно снизить объем программирования при внесении изменений в существующую программу, основанную на классах. Порождая от классов, имеющихся в библиотеке, свои, имеющие расширенную или измененную функциональность, программист может внести существенные изменения в программу, не изменяя базовых классов и/или программ, на них основанных.

Классы ABCи быстрая разработка приложений

Типичные преимущества повторного использования и сопровождения

Библиотека Классов ABC предоставляет все преимущества, которые, в принципе, способны предоставить библиотеки классов. Опирающиеся на ABC шаблоны системы Clarion автоматически генерируют код, использующий высокопроизводительные, гибкие и надежные (предварительно тестированные) объекты, определенные в библиотеке ABC. Более того, шаблоны построены так, что

помогают вам легко создавать собственные классы, порождая их на базе библиотеки ABC.

Разумеется, вам не обязательно использовать шаблоны, чтобы пользоваться Классами Построителя Приложений. Однако, изучив код, сгенерированный шаблонами, можно найти массу примеров использования библиотеки ABC при ручном кодировании. В любом случае, главное для вас - больше мощных программ при меньшем объеме кодирования.

Базы данных и ориентация Windows - приложений

Библиотека Классов ABC имеет несколько специфичную область применения. Это означает, что объекты этой библиотеки созданы для обработки баз данных в среде Windows. Если говорить еще точнее, эти объекты поддерживают всю стандартную функциональность предыдущих версий Clarion for Windows, а также многое другое.

К примеру, в библиотеку включены объекты, которые открывают, читают, распечатывают файлы баз данных, а также осуществляют их сортировку и поиск в них. Еще существует ряд объектов, поддерживающих ссылочную целостность между связанными файлами данных.

Помимо этого имеются классы общего назначения, которые выводят сообщения об ошибках, управляют контекстными меню, выполняют редактирование по месту, обеспечивают работу выпадающих списков, заполненных записями файла данных, осуществляют перевод текста в окнах и на элементах управления, управляют панелями инструментов в многопоточных приложениях, работают с INI-файлами, поддерживают выбор и обработку файлов операционной системы.

В общем, наша библиотека классов поддерживает Windows - программы, работающие с базами данных общего назначения. Она не поддерживает, скажем, управление процессами реального времени при перегонке нефти.

Базовые классы

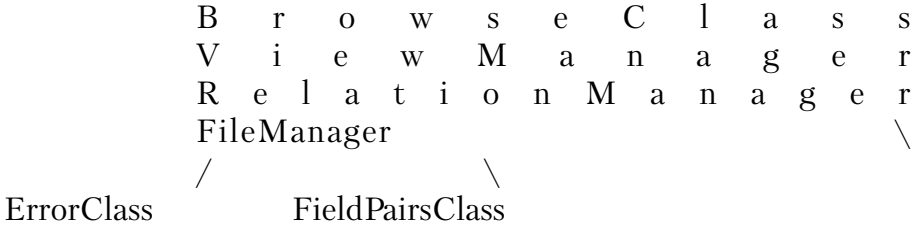
Классы ABC можно логически разделить на “базовые” и “периферийные”. Базовые классы являются ядром библиотеки ABC - все остальное построено на их основе или зависит от них. Если вы намерены изучить Классы ABC, начать следует именно с базовых классов. В дальнейшем доскональное знание этих классов послужит вам прекрасной основой для понимания сгенерированных при помощи

ABC-шаблонов программ и процедур, которые используют эти классы.

Даже если вы хотите оставаться в счастливом неведении о библиотеке ABC, вам стоит помнить пару вещей, относящихся к базовым классам:

- К базовым классам относятся ErrorClass, FieldPairsClass, FileManager, RelationManager, ViewManager, WindowManager, и BrowseClass.
- Базовые классы используются часто, поэтому если вам нужно модифицировать их, старайтесь делать это, не снижая их эффективности.
- Базовые классы присутствуют практически в каждой программе, написанной при помощи шаблонов, поэтому дополнительные ссылки на них незначительно сказываются на размере исполняемых файлов.

Базовые классы имеют следующую иерархию: ErrorClass и FieldPairClass составляют базис, на который опираются FileManager, RelationManager и ViewManager. Венцом иерархической пирамиды базовых классов является BrowseClass, порожденный от ViewManager. Класс WindowManager создан для взаимодействия с базовыми классами и управляет оконными процедурами, которые используют его.



Для изучения базовых классов мы рекомендуем действовать так. Вначале ознакомьтесь с классами, составляющими первооснову - ErrorClass и FieldPairClass. Затем постепенно продвигайтесь к иерархической вершине - BrowsClass.

Файлы исходного кода библиотеки ABC

По умолчанию классы ABC устанавливаются в каталог \CLARION4\LIBSRC. Конкретные классы находятся в соответствующих файлах, перечисленных ниже. Базовые классы выделены полужирным шрифтом.

Объявления классов находятся в соответствующих .INC-файлах, их методы - в .CLW-файлах.

ABASCII.INC	
AsciiFileClass	MODULE('ABASCII.CLW')
AsciiPrintClass	MODULE('ABASCII.CLW')
AsciiSearchClass	MODULE('ABASCII.CLW')
AsciiViewerClass	MODULE('ABASCII.CLW')

ABBROWSE.INC	
StepClass	MODULE('ABBROWSE.CLW')
StepLongClass	MODULE('ABBROWSE.CLW')
StepRealClass	MODULE('ABBROWSE.CLW')
StepStringClass	MODULE('ABBROWSE.CLW')
StepCustomClass	MODULE('ABBROWSE.CLW')
LocatorClass	MODULE('ABBROWSE.CLW')
StepLocatorClass	MODULE('ABBROWSE.CLW')
EntryLocatorClass	MODULE('ABBROWSE.CLW')
IncrementalLocatorClass	MODULE('ABBROWSE.CLW')
ContractingLocatorClass	MODULE('ABBROWSE.CLW')
EditClass	MODULE('ABBROWSE.CLW')
BrowseClass	**MODULE('ABBROWSE.CLW')**

ABDROPS.INC	
FileDropClass	MODULE('ABDROPS.CLW')
FileDropComboClass	MODULE('ABDROPS.CLW')

ABERROR.INC	
ErrorClass	**MODULE('ABERROR.CLW')**

ABFILE.INC	
FileManager	**MODULE('ABFILE.CLW')**
RelationUsage	**MODULE('ABFILE.CLW')**
RelationManager	**MODULE('ABFILE.CLW')**
ViewManager	**MODULE('ABFILE.CLW')**

ABPOPUP.INC	
PopupClass	MODULE('ABPOPUP.CLW')

ABREPORT.INC	
ProcessClass	MODULE('ABREPORT.CLW')
PrintPreviewClass	MODULE('ABREPORT.CLW')

ReportManager	MODULE('ABREPORT.CLW')
ABRESIZE.INC	
WindowResizeClass	MODULE('ABRESIZE.CLW')
ABTOOLBA.INC	
ToolbarTargetClass	MODULE('ABTOOLBA.CLW')
ToolbarListboxClass	MODULE('ABTOOLBA.CLW')
ToolbarReltreeClass	MODULE('ABTOOLBA.CLW')
ToolbarUpdateClass	MODULE('ABTOOLBA.CLW')
ToolbarClass	MODULE('ABTOOLBA.CLW')
ABUTIL.INC	
FieldPairsClass	**MODULE('ABUTIL.CLW')**
BufferedPairsClass	**MODULE('ABUTIL.CLW')**
INIClass	MODULE('ABUTIL.CLW')
DOSFileLookupClass	MODULE('ABUTIL.CLW')
TranslatorClass	MODULE('ABUTIL.CLW')
ABWINDOW.INC	
WindowManager	**MODULE('ABWINDOW.CLW')**

Включение нужных файлов в секции данных вашей программы

Объявления многих классов непосредственно ссылаются на другие классы. Чтобы разрешить эти ссылки, заголовочный файл (.INC) каждого класса включает только заголовки классов, на которые явно ссылается. Это соглашение обеспечивает максимальную инкапсуляцию, минимальное время компиляции, а также гарантирует присутствие всех необходимых для сборки приложения компонентов. Мы рекомендуем вам также следовать этому принципу.

Исходный код Классов ABC структурирован таким образом, что вы можете в своих программах включать как заголовочный файл (.INC), так и файл с полными определениями (.CLW). Если вы включаете заголовок, он ссылается на необходимые определения и наоборот.

Правило хорошего тона - включайте как можно меньше. Компилятор даст вам знать, если вы что-то пропустили.

ABC-библиотека и ABC-шаблоны

ABC-шаблоны тесно связаны с ABC-библиотекой. Однако шаблоны обладают большими возможностями для настройки и устроены так, что дают вам возможность подставить ваши собственные определения классов. Для получения более полной информации о настройке глобального уровня взаимодействия ABC-шаблонов с ABC-библиотекой смотрите *Часть I - Панель "Опции Классов"(Глобальные)*. Локальное (на уровне модулей) взаимодействие рассматривается в *Части I - Панель "Опции Классов"(Локальные)*.

Классы и их объекты, сгенерированные шаблонами

ABC-шаблоны объявляют объекты, являющиеся экземплярами классов из ABC-библиотеки. Стандартные имена объектов, генерируемые шаблонами, обычно похожи на имена соответствующих классов, но несколько отличаются. Сгенерированный код вашего приложения может содержать объявления данных и исполняемые операторы примерно такого вида:

```
GlobalErrors                    ErrorClass
Hide:Access:Customer            CLASS(FileManager)
INIMgr                          INIClass
ThisWindow                      CLASS(ReportManager)
ThisWindow                      CLASS(WindowManager)
ThisReport                      CLASS(ProcessClass)
ThisProcess                     CLASS(ProcessClass)
BRW1                            CLASS(BrowseClass)
EditInPlace::CUS:NAME           EditClass
Resizer                         WindowResizeClass
Toolbar                         ToolbarClass
        CODE
GlobalResponse = ThisWindow.Run()
BRW1.AddSortOrder(BRW1::Sort0:StepClass,ST:StKey)
BRW1.AddToolbarTarget(Toolbar)
GlobalErrors.Throw()
Resizer.AutoTransparent=True
Previewer.AllowUserZoom=True
```

Эти объявления данных описывают экземпляры классов ABC-библиотеки, а исполняемые операторы ссылаются на описанные ранее объекты. Различные классы и их шаблонные экземпляры перечислены ниже, чтобы вы могли идентифицировать ABC-объекты в ваших программах и найти соответствующую документацию по ABC-библиотеке.

Генерируемый объект	Класс ABC
GlobalErrors	ErrorClass
INIMgr	INIClass
Access:*file*	FileManager
Relate:*file*	RelationManager
ThisWindow	WindowManager, ReportManager
BRWn	BrowseClass
BRWn::Sortn:Locator	LocatorClass
BRWn::Sortn:StepClass	StepClass
EditInPlace::*field*	EditClass
Popup	PopupClass
Resizer	WindowResizeClass
Toolbar	ToolbarClass
ToolbarForm	ToolbarUpdateClass
RELn::Toolbar	ToolbarReltreeClass
ThisReport	ProcessClass
Previewer	PrintPreviewClass
ThisProcess	ProcessClass
ProgressMgr	StepClass
FDBn	FileDropClass
FDCBn	FileDropComboClass
ViewerN	ASCIIViewerClass
FileLookupN	SelectFileClass
Translator	TranslatorClass

Соглашения по ABC-кодированию

ABC-библиотека пользуется некоторыми соглашениями по кодированию. Вы можете увидеть примеры этих конструкций в сгенерированном коде ABC-приложений и в самом коде ABC-библиотеки. Мы рекомендуем вам следовать этим соглашениям при написании embed-вставок.

Имена методов

Ряд имен имеют особый смысл в ABC-библиотеке. Эти имена и их смысл приведены ниже.

Add*Элемент* — Объект добавляет *элемент* в свое хранилище данных. Этим элементом может быть поле, ключ, порядок сортировки, диапазон значений ключа, другой объект и т.д. Элементом может служить все, что нужно объекту для работы.

Ask[*Информация*]	Метод взаимодействует с конечным пользователем для получения *информации*.
Fetch	Метод получает данные из файла.
Get*Элемент*	Метод возвращает значение указанного *элемента*.
Init	Метод выполняет некоторые действия, необходимые для инициализации объекта.
Kill	Метод выполняет некоторые действия, необходимые для завершения работы и ликвидации объекта, включая освобождение памяти, захваченной объектом на протяженнии его жизни.
Reset[*что* или *как*]	Метод устанавливает параметры объекта и его элементов управления. Это включает загрузку данных, установку порядков сортировки, перерисовку оконных элементов управления и т.д.
Set*Item*	Метод устанавливает значение указанного *элемента* (Item) или активизирует его для того, чтобы другие методы объекта могли работать с ним.
Take*Item*	Метод "берет" *элемент* (Item) из другого метода или объекта и продолжает его обработку. Элементом может быть оконное событие (Accepted, Rejected, OpenWindow, CloseWindow, Resize и т.д.), запись, ошибочная ситуация и т.д.
Throw[*Item*]	Метод "передает" *элемент* (Item) другому объекту или методу для обработки. Элементом обычно служит ошибочная ситуация.
Try*Action*	Метод делает одну попытку выполнить *действие* (Action), после чего возвращает результат, показывающий, успешно ли завершилось действие. Нулевой результат (0 или Level:Benign) означает успешное завершение, любой другой результат - неудачу.

Где инициализировать и ликвидировать объекты

В общем случае следует иметь в виду два фактора при инициализации и ликвидации объектов:

- Объекты должны жить возможно более короткое время.
- Объекты должны ликвидироваться вызовом метода Kill (чтобы освободить память, захваченную объектом в течение его жизни).

Баланс этих двух, иногда противоречащих факторов говорит, что объекты, инициализированные при обработке события EVENT:OpenWindow, обычно ликвидируются при обработке события EVENT:CloseWindow. Объекты, инициализированные в ThisWindow.Init, обычно ликвидируются в ThisWindow.Kill

Возвращаемые значения

Многие ABC-методы возвращают результат, значение которого показывает, завершилась операция успешно или окончилась неудачно. Нулевое значение (0 или Level:Benign) означает успех. Любое другое значение означает проблему, серьезность которой может быть различной. Другие возвращаемые значения и соответствующие им константы "серьезности", принятые в ABC-библиотеке (Level:User, Level:Cancel, Level:Notify, Level:Fatal, Level:Program), документированы в главе *ErrorClass* и в разделах, посвященных конкретным методам.

В результате этого соглашения, код имеет следующий вид:

```
IF ABCObject.Method()                    !обработка неудачи/ошибки
ELSE                                     !нормальное продолжение
END

IF ~ABCObject.Method()                   !нормальное продолжение
END
```

Значения, возвращаемые методами обработки событий

Некоторые ABC-методы обрабатывают события из ACCEPT-цикла. Имена этих методов начинаются с "Take" и обычно указывают тип событий, которые они обрабатывают. Эти методы выполняются в рамках ACCEPT-цикла (как реализовано в методе WindowManager.Ask) и возвращают результат, указывающий, как должен вести себя ACCEPT-цикл.

Значение, соответствующее Level:Benign указывает, что обработка данного события должна протекать нормальным образом. Значение Level:Notify указывает, что обработка завершена, и ACCEPT- цикл должен выполнить операцию CYCLE. Значение Level:Fatal говорит о том, что событие не может быть обработано, и должен быть осуществлен выход из ACCEPT-цикла (операция BREAK).

Если вы (или ABC-шаблоны) порождаете класс, имеющий такие методы, вы должны выполнять это соглашение о возвращаемых значениях применительно к управлению ACCEPT-циклом.

Ниже приводится фрагмент кода метода WindowManager.Ask, который реализует это соглашение.

```
ACCEPT
        CASE SELF.TakeEvent()
        OF Level:Fatal
```

```
            BREAK
         OF Level:Notify
            CYCLE
         END
END
```

Завершение процедуры

У вас может возникнуть необходимость немедленного завершения процедуры (т.е. ожидание события EVENT:CloseWindow невозможно или это событие не подходит).

В ряде случаев простой RETURN не завершит процедуру (поскольку RETURN, вставленный в некоторый метод, завершает этот метод, но не вызвавшую его процедуру), а даже если и завершит, то использование его исключено по другим причинам (например, процедура динамически выделяла память или запускала другие задачи, завершение которых должно быть проконтролировано.)

Существует несколько способов инициировать нормальное завершение процедуры в зависимости от того, в каком месте процедуры вы вставляете свой код. Они приведены ниже.

```
RETURN(Level:Fatal)                 !нормальное завершение в порожденном от
                                    !ABC методе
ReturnValue = Level:Fatal           !нормальное завершение в конце
                                    !порожденного от ABC метода
ThisWindow.Kill                     !нормальное завершение из подпрограммы в
                                    !процедуре
ThisWindow.Kill;RETURN              !нормальное завершение из подпрограммы в
                                    !процедуре, вызванной из ACCEPT-цикла
```

Внутренние (PRIVATE) недокументированные элементы

Некоторые свойства и методы в ABC-библиотеке имеют атрибут PRIVATE. Эти элементы не документированы. Они являются внутренними, поскольку будут изменены или ликвидированы в будущих выпусках библиотеки. Объявление ряда элементов как внутренних дает фирме TopSpeed возможность изменить и усовершенствовать эти области, не оказывая влияния на приложения, разработанные на базе библиотеки ABC. Мы настоятельно рекомендуем не снимать атрибут PRIVATE с элементов библиотеки ABC.

ErrorClass - обработчик ошибок

Обзор

ErrorClass объявляет обработчик ошибок, который полно и гибко обрабатывает любые ошибки. Это означает, что для данной программы вы можете определить все возможные ошибки, указав их номера, уровни требуемой обработки и тексты сообщений. Затем, при возникновении ошибки или в других случаях, требующих обработки, вы просто передаете в качестве параметра соответствующий код обработчику ошибок, а тот, в свою очередь, обрабатывает ее в соответствии со степенью ее серьезности.

Определенные таким образом "ошибки" в действительности могут включать в себя помимо реальных ошибок вопросы, предупреждения, замечания, сообщения и т.д. В стандартной поставке уже определено около сорока общих ошибок, связанных с обработкой баз данных. Вы можете расширить этот список, добавив ошибки общего характера, специфические ошибки прикладного уровня, или даже ошибки проверки корректности отдельных данных. Расширение списка обрабатываемых ошибок может носить "перманентный" характер, либо осуществляться динамически во время выполнения программы.

Исходные файлы ErrorClass

Исходные файлы ErrorClass по умолчанию устанавливаются в каталог \CLARION4\LIBSRC. Ниже перечислены файлы ErrorClass и их содержание:

ABERROR.INC Объявления ErrorClass

ABERROR.CLW Определения методов ErrorClass.

ABERROR.TRN Стандартный список ошибок для ErrorClass.

Различные уровни обработки ошибок и их настройка

Шесть уровней обработки

По умолчанию обработчик ошибок распознает шесть различных уровней серьезности ошибок. Стандартные действия для этих уровней варьируются от отсутствия обработки для легких ошибок до остановки программы для ошибок фатальных. Обработчик ошибок также поддерживает промежуточные действия, как простое оповещение пользователя или предложение выбрать меду продолжением и завершением программы.

Настройка обработки

Вышеупомянутые уровни обработки реализованы в виде виртуальных методов, что делает легкой их настройку. Обработчик ошибок вызывает различные виртуальные методы для каждого уровня, поэтому вы можете заменить стандартную обработку ошибок на специфичную для вашего приложения. Для примера рассмотрите разнообразные *Take*-методы.

Распознаваемые константы серьезности ошибок объявлены в файле ABERROR.INC. Ниже перечислены эти уровни и соответствующие стандартные действия:

Level:Benign	нет действий, возвращается Level:Benign
Level:User	выводится сообщение, возвращается Level:Benign или Level:Cancel
Level:Notify	выводится сообщение, возвращается Level:Benign
Level:Fatal	выводится сообщение, программа завершается
Level:Program	обрабатывается как Level:Fatal

любое другое значение обрабатывается как Level:Program

Вы можете определить ваши собственные дополнительные уровни серьезности и соответствующие им действия.

Предопределенные ошибки окон и баз данных

Список общих ошибок баз данных определен в файле ABERROR.TRN и может быть использован вами и ABC-шаблонами. Эти список “ошибок” наряду с настоящими ошибками включает в себя вопросы, предупреждения, сообщения, замечания и т.д.

Вы можете отредактировать определения этих ошибок в соответствии с вашими потребностями. То есть, вы можете добавить определения новых ошибок, изменить содержание сообщений об ошибках, или перевести английский текст на другой язык.

Примечание: Если вы используете ABC-шаблоны, вам не следует удалять стандартные определения ошибок или изменять их номера.

Динамичиское расширение ошибок

Вы можете добавлять новые определения ошибок, заменять стандартные определения и модифицировать стандартные определения ошибок на этапе выполнения программы при помощи методов, предназначенных для этих целей:

AddErrors	Добавляет новые ошибки, заменяет их, или делает то и другое вместе.
RemoveErrors	Удаляет ошибки, восстанавливает ранее замененные или делает то и другое вместе.
SetFatality	Изменяет уровень скрьезности ошибки.

Реализация ABC-шаблонов

ABC-шаблон объявляет глобальный обработчик ошибок, называемый GlobalErrors. Все шаблоны распознают ошибки, определенные при старте программы, и далее почти каждая генерируемая процедура опирается на обработчик ошиибок для обработки известных ошибочных ситуаций.

Отношение к другим Классам

ErrorClass используется всеми классами, имеющими дело с файлами: ASCIIFileClass, ASCIIViewerClass, FileManager, RelationManager, ViewManager и BrowseClass. Таким образом, если ваша программа содержит объекты, указанных классов, она должна содержать и объект ErrorClass.

Макрорасширения

Следующие методы ErrorClass позволяют настраивать текст сообщений об ошибках при помощи макроподстановок:

SetField	Указывает поле, которое привело к ошибке.
SetFile	Указывает файл, который привел к ошибке.

ThrowFile	Указывает файл, который привел к ошибке, затем обрабатывает ошибку.
ThrowMessage	Модифицирует текст ошибки, затем обрабатывает ошибку.

Каждая ошибка имеет ассоциированный текст сообщения. Текст сообщения может содержать макропеременные, известные обработчику ошибок. Перед тем, как вывести на экран сообщение, он заменяет эти макропеременные на их текущие значения. Ниже приводится список поддерживаемых макропеременных и их расшифровка времени выполнения:

%File	Свойство ErrorClass.FileName
%Field	Свойство ErrorClass.FieldName
%Message	Свойство ErrorClass.MessageText
%Error	Значение функции ERROR()
%ErrorCode	Значение функции ERRORCODE()
%FileError	Значение функции FILEERROR()
%FileErrorCode	Значение функции FILEERRORCODE()
%ErrorText	%Error(%ErrorCode) или %FileError(%FileErrorCode)
%Previous	Текст от ранее определенной ошибки с этим же кодом.

Макропеременная %ErrorText использует %FileError(%FileErrorCode) - более точную информацию об ошибке, если она доступна, в противном случае используется %Error(%ErrorCode).

Описанные возможности макрорасширения являются свойством ErrorClass, а не языка Clarion.

Совет: Вам не обязательно указывать два символа процента (%%) чтобы вывести знак процента (%) в своем сообщении.

Многоязыковая поддержка

Поскольку все тексты сообщений об ошибках определены в одном месте (ABERROR.TRN), весьма просто реализовать не английские сообщения об ошибках. Для статического перевода просто переведите английский текст в ABERROR.TRN на нужный вам язык. Для динамического перевода следует добавить в файл ABERROR.TRN блок определений ошибок для каждого поддерживаемого языка. Например, объявите в ABERROR.TRN:

```
DefaultErrors  GROUP                    !Английские сообщения
               END
GermanErrorsGROUP                       !Немецкие сообщения
               ND
```

Затем во время выполнения программы инициализируйте обработчик ошибок подходящим блоком определений. Например, вы можете переопределить метод Init примерно таким образом:

```
INCLUDE('ABERROR.INC')                  !объвление ErrorClass
MyErrorClass CLASS(ErrorClass)          !объявление  порожденного класса
Init           PROCEDURE(BYTE PreferredLanguage)
               END

GlobalErrors   MyErrorClass             !объявление
                                        !объекта GlobalErrors
Language  BYTE                          !Флаг языка
Language:English   EQUATE(0)            !Английский
Language:German  EQUATE(1)              !Немецкий

               CODE
Language = GETINI('Preferences','Language',0)  !Прочесть необходимый язык
GlobalErrors.Init(Language)                    !Инициализация GlobalErrors
                                               !необходимым языком
               .
               .
               .
MyErrorClass.Init PROCEDURE(BYTE PreferredLanguage)        !Новый метод Init
               CODE
               SELF.Errors &= NEW ErrorEntry       !Выделить новый список ошибок
               CASE PreferredLanguage              !Какой язык выбран
               OF Language:German                  !Немецкий
               SELF.AddErrors(GermanErrors)        !Добавим в список немецкие
                                                   !ошибки
               ELSE                                !иначе
               SELF.AddErrors(DefaultErrors)       !Добавим стандартные (английские)
                                                   !ошибки
               END
```

Кроме того, вы можете вызвать метод AddErrors чтобы определить дополнительные ошибки для выбранного языка. Этот подход показан в приведенном ниже примере.

Концептуальный пример

Приведенный ниже пример демонстрирует типичную последовательность операторов, объявляющих, инициализирующих использующих и завершающих объект ErrorClass.

```
PROGRAM
INCLUDE('ABERROR.INC')                  !включить объявление ErrorClass
AppErrors  GROUP                        !Объявить специфичные для
                                        !приложения ошибки
Number USHORT(2)                        !Количество ошибок в группе
USHORT(Msg:DuplicateKey)                !Код первой ошибки
 BYTE(Level:Notify)                     !Уровень серьезности
 PSTRING('Duplicate Key')               !Заголовок окна
 PSTRING('%File key is invalid.')       !Текст сообщения с макропеременной
    USHORT(Msg:FieldOutOfRange)         !Код второй ошибки
    BYTE(Level:Notify)                  !Уровень серьезности
    PSTRING('Range Error')              !Заголовок окна
PSTRING('%Field must be between %Message.') !Текст сообщения
            END

GlobalErrors ErrorClass                 !Объявим объект GlobalErrors
CODE
GlobalErrors.Init                       !Инициализация (добавим
                                        !стандартные ошибки)
GlobalErrors.AddErrors(AppErrors)       !Добавим специфические ошибки
GlobalErrors.SetFatality(Msg:DuplicateKey,Level:Fatal)
                                        !Модифицируем серьезность ошибки
.                                       !
.                                           !Код программы
.                                       !
!Пользователь пытается ввести неверный месяц
GlobalErrors.SetField('Month')          !Установить значение
                                        !макропеременной %Field
GlobalErrors.ThrowMessage(Msg:FieldOutOfRange,'1 and 12')
                                        !Передать ошибку обработчику
.                                       !
                                        !Пользователь пытается ввести
                                        !повторяющийся ключ
GlobalErrors.SetFile('Customer')        !Установить значение
                                        !макропеременной %File

GlobalErrors.Throw(Msg:DuplicateKey)!Передать ошибку обработчику
```

```
                                !program code
.                               !
GlobalErrors.Kill               !Завершить работу объекта GlobalErrors
```

Свойства ErrorClass

Существует два типа свойств ErrorClass - список ошибок и значения макроподстановок. Наиболее важным свойством является список ошибок, распознаваемых объектами ErrorClass. Определенные “ошибки” наряду с настоящими ошибками могут быть вопросами, предупреждениями, замечаниями и т.д. Этот список устанавливается методом инициализации ErrorClass.Init. Впоследствии список может быть модифицирован при помощи методов, служащих этой цели, что позволяет использовать специфичные ошибки приложения (такие, как специфические сообщения о неверных данных).

Другие три свойства ErrorClass поддерживают распознаваемые обработчиком ошибок макроопределения для текстов сообщений. Обработчик ошибок заменяет эти макросиволы их текущими значениями времени выплонения перед тем, как вывести на экран сообщение.

Errors (распознаваемые определения ошибок)

Errors &ErrorEntry, PROTECTED

Свойство **Errors** является ссылкой на структуру данных, содержащую все ошибки, распознаваемые объектами ErrorClass. Определенные “ошибки” наряду с настоящими ошибками могут быть вопросами, предупреждениями, замечаниями и т.д.

Данное свойство имеет атрибут PROTECTED, следовательно, к нему могут обращаться только методы ErrorClass или методы порожденных от него классов.

Стандартные ошибки определены в файле ABERROR.TRN, который вы можете отредактировать в целях настройки списка стандартных ошибок под свои нужды. Метод Init во время исполнения программы добавляет эти стандартные ошибки в свойство Errors. Вы можете использовать методы SetFatality, AddErrors и RemoveErrors для динамической настройки свойства Errors на этапе выполнения.

Метод SetFatality изменяет уровень серьезности указанной ошибки.
Метод AddErrors позволяет вам добавить новые определения ошибок, заменить

существующие определения, или выполнить обе эти функции сразу. Свойство Errors может иметь несколько ошибок с одинаковыми кодами. Определения ошибок, добавленные позже, "переопределяют" любые ранее добавленные ошибки с тем же кодом. "Переопределенные" определения сохраняются для подстановки вместо макропеременной %Previous.

Метод RemoveErrors позволяет вам удалить определения ошибок, восстановить переопределенные ранее, или выполнить оба эти действия сразу.

Сообщения об ошибках могут содержать макроопределения, известные обработчику ошибок, который заменяет эти макросимволы их текущими значениями времени выполнения перед тем, как вывести на экран сообщение. Более подробную информацию смотрите в разделе *Макрорасширения*.

Реализация: Errors представляет собой ссылку на очередь, объявленную в ABERROR.INC; объявление приведено ниже. Для каждой распознаваемой ошибки структура Errors включает код, текст сообщения, текст заголовка окна и индикатор степени серьезности.

```
ErrorEntry  QUEUE,TYPE          !Список всех определений ошибок
Id          SHORT               !Код ошибки
Message     STRING              !Текст сообщения
Title       STRING              !Текст заголовка окна
Fatality    BYTE                !Серьезность ошибки
            END
```

См. также: AddErrors, Init, RemoveErrors, SetFatality

FieldName (поле, вызвавшее ошибку)

FieldName CSTRING(MessageMaxlen), PROTECTED

Свойство **FieldName** содержит имя поля, которое вызвало ошибку. Метод SetField устанавливает значение свойства FieldName. Значение FieldName заменяет макропеременную %Field в тексте сообщения.

MessageMaxlen - константа, объявленная в файле ABERROR.INC.

Данное свойство имеет атрибут PROTECTED, следовательно, к нему могут обращаться только методы ErrorClass или методы порожденных от него классов.

См. также: SetField

FileName (файл, который вызвал ошибку)

FileName CSTRING(MessageMaxlen), PROTECTED

Свойство **FileName** содержит имя файла, который вызвал ошибку. Методы SetFile и ThrowFile устанавливают значение свойства FileName. Значение FileName заменяет макропеременную %File в тексте сообщения.

MessageMaxlen - константа, объявленная в файле ABERROR.INC.

Данное свойство имеет атрибут PROTECTED, следовательно, к нему могут обращаться только методы ErrorClass или методы порожденных от него классов.

См. Также: SetFile, ThrowFile

MessageText (текст сообщения об ошибке)

MessageText CSTRING(MessageMaxlen), PROTECTED

Свойство **MessageText** содержит текст, подставляемый вместо макропеременную %Message в тексте сообщения об ошибке. Метод ThrowMessage устанавливает значение свойства MessageText. Значение MessageText заменяет макропеременную %File в тексте сообщения.

MessageMaxlen - константа, объявленная в файле ABERROR.INC.

Данное свойство имеет атрибут PROTECTED, следовательно к нему могут обращаться только методы ErrorClass или методы порожденных от него классов.

См. Также: ThrowMessage

Методы ErrorClass

Функциональная организация - ожидаемое использование

В качестве помощи для понимания ErrorClass, можно разделить многообразные методы этого класса на две большие категории в соответствии с их ожидаемым использованием - первичный интерфейс и виртуальные методы. Такое структурирование отражает то, что мы понимает под типичным использованием методов ErrorClass.

Методы первичного интерфейса

Методы первичного интерфейса, которые вы, скорее всего, явно вызываете в своих программах, могут в свою очередь быть разделены на три категории:

Организационного (однократного) применения:

Init	инициализирует объект ErrorClass
AddErrors	добавляет или переопределяет ошибки SetFatality изменяет уровень серьезности конкретной ошибки
Kill	завершает работу объекта ErrorClass

Основного применения:

Throw	обрабатывает ошибку
ThrowFile	устанавливает значение макропеременной %File и обрабатывает ошибку ThrowMessage устанавливает значение макропеременной %Message и обрабатывает ошибку Message выводит сообщение из списка ошибок

Эпизодического применения:

SetField	устанавливает значение макропеременной %Field
SetFile	устанавливает значение макропеременной %File
SetErrors	сохраняет текущее состояние ошибок SetId делает выбранную ошибку текущей
RemoveErrors	удаляет (и /или) восстанавливает определения ошибки remove
TakeError	обрабатывает ошибку, предполагая, что SetErrors был вызван

Виртуальные методы

Как правило вы не делаете непосредственных вызовов этих методов - первичный интерфейс вызывает их. Мы предполагаем, что вам захочется переопределить эти методы, и, поскольку они являются виртуальными, сделать это несложно. Однако, и их стандартное поведение представляется разумным. Эти методы приведены в порядке функциональности.

TakeBenign	обрабатывает ошибки уровня Level:Benign
TakeNotify	обрабатывает ошибки уровня Level:Notify
TakeUser	обрабатывает ошибки уровня Level:User
TakeFatal	обрабатывает ошибки уровня Level: Fatal
TakeProgram	обрабатывает ошибки уровня Level:Program
TakeOther	обрабатывает прочие ошибки

AddErrors (добавляет или переопределяет ошибки)

AddErrors(*блок ошибок* **), VIRTUAL**

AddErrors	Добавляет элементы в свойство Errors из *блока ошибок*, переданного в качестве параметра.
блок ошибок	Структура GROUP, первое поле которой имеет тип USHORT, и его значение задает количество определений ошибок в этой структуре. Последующие поля определяют элементы списка ошибок.

Метод **AddErrors** принимает элементы списка ошибок и добавляет их в существующее свойство Errors. Добавленные позже определения ошибок "переопределяют" любые ранее добавленные определения с теми же кодами. "Переопределенные" ошибки сохраняются для подстановки вместо макропеременной %Previous и могут быть полностью восстановлены удалением переопределяющих элементов при помощи метода RemoveErrors.

Рассматриваемый метод является виртуальным, следовательно другие методы базового класса (ErrorClass) могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: AddErrors работает в предположении, что свойство Errors уже корректно инициализировано.

Каждый элемент *блока ошибок* состоит из поля типа USORT, в котором содержится код ошибки, поля типа BYTE, хранящего уровень серьезности, поля типа PSTRING, содержащего заголовок окна с сообщением об ошибке, и еще одного поля типа PSTRING, содержащего собственно текст сообщения.

Пример:

```
AppErrors  GROUP
Number     USHORT(2)                    !Количество ошибок в группе
USHORT(Msg:RebuildKey)                  !Код первой ошибки
BYTE(Level:Notify)                      !Уровень серьезности
PSTRING('Invalid Key')                  !Заголовок окна
```

```
PSTRING('%File key is invalid.')        !Текст сообщения
USHORT(Msg:RebuildFailed)               !Код второй ошибки
BYTE(Level:Fatal)                       !Уровень серьезности
PSTRING('Key was not built')            !Заголовок окна
PSTRING('Repairing key for %File.')     !Текст сообщения
        END
GlobalErrors  ErrorClass                !объявление GlobalErrors
CODE
GlobalErrors.Init                       !инициализация GlobalErrors
GlobalErrors.AddErrors(AppErrors)       !добавить специфические
                                        !ошибки приложения
Main                                    !Вызов главной процедуры
GlobalErrors.Kill                       !завершение GlobalErrors
```

См. Также: Init, Errors, RemoveErrors

Init (инициализация объекта ErrorClass)

Init

Метод **Init** инициализирует объект класса ErrorClass и добавляет стандартные ошибки.

Реализация: Создает свойство Errors и вызывает метод AddErrors для его инициализации стандартными ошибками, определенными в файле ABERROR.TRN. Стандартные константы кодов ошибок определены в файле ABERROR.INC.

Стандартные шаблоны заводят глобальный объект и делают единственный вызов метода Init. Однако вы можете иметь объект ErrorClass с отдельным списком ошибок для каждого из базовых классов либо для каждой логической сущности (например, ошибки платежей в той части своей программы, что занимается платежами).

Пример:

```
GlobalErrors  ErrorClass                !объявление GlobalErrors
        CODE
        GlobalErrors.Init               !инициализация GlobalErrors
        Main                            !Вызов главной процедуры
        GlobalErrors.Kill               !завершение GlobalErrors
```

См. Также: AddErrors, Errors, Kill

Kill (выполняет необходимый код завершения)

Kill

Метод **Kill** освобождает всю память, выделенную для работы объекта на протяжении его жизни, и выполняет все необходимые действия для завершения работы объекта.

Реализация: Очищает очередь Errors, созданную методом Init.

Пример:

```
GlobalErrors ErrorClass                    !объявление GlobalErrors
        CODE
        GlobalErrors.Init                  !инициализация GlobalErrors
        Main
                                           !Вызов главной процедуры
        GlobalErrors.Kill                  !завершение GlobalErrors
```

См. Также: Init

Message (выводит на экран сообщение об ошибке)

Message(*код ошибки, кнопки, кнопка по умолчанию* **)**

Message Выводит окно с сообщением об ошибке, возвращает кнопку, нажатую пользователем.

код ошибки	Целочисленная константа, переменная или выражение, указывающая, для какого элемента очереди ErrorClass.Errors следует вывести сообщение.
кнопки	Целочисленная константа, переменная или выражение, указывающее, какие из стандартных кнопок Windows следует поместить в окне.
кнопка по умолчанию	Целочисленная константа, переменная или выражение, указывающая кнопку по умолчанию.

Метод **Message** выводит на экран стандартное окно сообщений Windows, содержащее текст сообщения об ошибке из свойства Errors, и возвращает номер копки, нажатой пользователем. Этот метод предоставляет простой, последовательный, поддающийся централизованному сопровождению способ вывода сообщений.

Реализация: Использует оператор MESSAGE, который выводит модальное в рамках приложения окно с пиктограммой в виде вопросительного знака

(ICON:Question) и текстом сообщения, определенным в свойстве Errors.

Файл ABERROR.INC содержит список стандартных символических констант для параметра *код ошибки*.

Файл EQUATES.CLW содержит символические константы для параметров *кнопки* и *кнопку по умолчанию*. Вот эти константы:

```
BUTTON:OK
BUTTON:YES
BUTTON:NO
BUTTON:ABORT
BUTTON:RETRY
BUTTON:IGNORE
BUTTON:CANCEL
BUTTON:HELP
```

Тип результата: LONG

Пример:

```
!Попытка автонумерации ключа потерпела неудачу, покажем
!окно с кнопками Да и Нет, по умолчанию Нет

GlobalErrors.SetErrors                  !Установим значение макропеременной
                                        !%ErrorText
IF GlobalErrors.Message(Msg:RetryAutoInc,|
            BUTTON:Yes+BUTTON:No,|
            BUTTON:No) = BUTTON:Yes
           CYCLE
END
```

RemoveErrors (удаляет или восстанавливает ошибки)

RemoveErrors(*блок ошибок* **)**

RemoveErrors Удаляет элементы, обозначенные в *блоке ошибок*, из свойства Errors.

блок ошибок Структура GROUP, первое поле которой имеет тип USHORT, а его значение задает количество определений ошибок в этой структуре. Последующие поля определяют элементы списка ошибок.

Метод **RemoveErrors** принимает элементы списка ошибок и удаляет их из существующего свойства Errors.

Свойство Errors может содержать несколько ошибок с одинаковым кодом. Ошибки, добавленные позже, переопределяют более ранние определения. Если вы удаляете такую переопределяющую ошибку, то переопределенная ошибка полностью восстанавливается.

Реализация: Каждый элемент *блока ошибок* состоит из поля типа USORT, в котором содержится код ошибки, поля типа BYTE, хранящего уровень серьезности, поля типа PSTRING, содержащего заголовок окна с сообщением об ошибке, и еще одного поля типа PSTRING, содержащего, собственно, текст сообщения.

Пример:

```
GlobalErrors  ErrorClas                          !объявление GlobalErrors
Payroll       PROCEDURE
PayErrors     GROUP,STATIC
Number        USHORT(2)                          !Количество ошибок в группе
USHORT(Msg:RebuildKey)                           !Код первой ошибки
BYTE(Level:Notify)                               !Уровень серьезности
PSTRING('Invalid Key')                           !Заголовок окна
PSTRING('%File key is invalid.')                 !Текст сообщения
USHORT(Msg:RebuildFailed)                        !Код второй ошибки
BYTE(Level:Fatal)                                !Уровень серьезности
PSTRING('Key was not built')                     !Заголовок окна
PSTRING('Repairing key for %File.')              !Текст сообщения
              END
              CODE
GlobalErrors.AddErrors(PayErrors)          !Добавить специфические ошибки
                                           !процедуры
   .                                       !некототрые действия
GlobalErrors.RemoveErrors(PayErrors)       !удалить специфические ошибки
                                           !процедуры
```

См. также: AddErrors, Init, Errors

SetErrors (сохраняет состояние ошибки)

SetErrors

Метод **SetErrors** сохраняет состояние текущей ошибки для использования объектом ErrorClass.

Реализация: Метод **SetErrors** сохраняет значения, возвращаемые функциями ERROR(), ERRORCODE(), FILEERROR() и FILERERRORCODE(). Сохраненные значения подставляются вместо макропеременных %Error, %ErrorCode, %FileError, или %FileErrorCode, встречающихся в тексте сообщения об ошибке.

Метод Throw вызывает SetErrors перед обработкой конкретной ошибки, таким образом вам нужно только вызвать метод SetErrors, когда вы не вызываете метод Throw.

Пример:

```
!произошла ошибка
GlobalErrors.SetErrors                  !сохранить состояние ошибки
OPEN(LogFile)                           !открыть файл протокола (состояние
                                        !ошибки изменилось)
Log:Text = FORMAT(TODAY(),@D1)&' '&FORMAT(CLOCK(),@T1)
ADD(LogFile)                            !записать в протокол (состояние
                                        !ошибки изменилось)
RETURN GlobalErrors.TakeError(Msg:AddFailed)
                                        !обработать ошибку с сохраненным остоянием
```

См. также: Throw

SetFatality (устанавливает уровень серьезности для конкретной ошибки)

SetFatality(*код ошибки, серьезность* **)**

SetFatality	Задает уровень серьезности конкретной ошибки в свойстве Errors.
код ошибки	Целочисленная константа, переменная или выражение, указывающее, какое определение ошибки следует модифицировать.
серьезность	Целочисленная константа, переменная или выражение, задающее серьезность ошибки.

Метод **SetFatality** задает уровень серьезности конкретной ошибки в свойстве Errors. Если имеется несколько ошибок с одинаковым кодом, модифицируется только последняя ошибка.

Реализация: Метод SetFatality вызывает метод SetId для того, чтобы найти указанную ошибку.

Файл ABERROR.INC содержит список символических констант для параметра *код ошибки*. Там же содержатся символические константы для параметра *серьезность*. Ниже перечислены константы уровней серьезности и соответствующие им стандартные действия:

Level:Benign	нет действий, возвращается Level:Benign

Level:User	выводится сообщение, возвращается Level:Benign или Level:Cancel
Level:Notify	выводится сообщение, возвращается Level:Benign
Level:Fatal	выводится сообщение, программа завершается
Level:Program	обрабатывается как Level:Fatal

любое другое значение обрабатывается как Level:Program

Вы можете определить ваши собственные дополнительные уровни серьезности и соответствующие им действия.

Пример:

```
GlobalErrors ErrorClass
CODE
GlobalErrors.Init
GlobalErrors.SetFatality(Msg:CreateFailed,Level:Fatal)    !изменить уровень
                                                          !серьезности до фатального
CREATE(MyFile)
IF ERRORCODE()
GlobalErrors.SetFile('MyFile')                !задать файл, на котором произошла
                                              !неудача
GlobalErrors.Throw(Msg:CreateFailed)          !показать сообщение о фатальной ошибке
        END
```

См. также: Errors, SetId

SetField (устанавливает значение макропеременной %Field)

SetField(*имя поля* **)**

SetField	Устанавливает значение макропеременной %Field.
имя поля	Строковая константа, переменная или выражение, указывающая поле, послужившее источником ошибки.

Метод **SetField** устанавливает значение макропеременной %Field. Это значение заменяет все макропеременные %Field в тексте сообщения об ошибке.

Реализация: Присваивает параметр *имя поля* свойству ErrorClass.FieldName.

Пример:

```
!Контроль поля State завершился неудачей
GlobalErrors.SetField('State')          !установим поле - источник ошибки
GlobalErrors.ThrowMessage(Msg:FieldNotInFile,'State File')
                                        !обработаем ошибку
```

См. также: FieldName

SetFile (устанавливает значение макропеременной %File)

SetFile(*имя файла* **)**

SetFile	Устанавливает значение макропеременной %File.
имя файла	Строковая константа, переменная или выражение, указывающее файл, послуживший источником ошибки.

Метод **SetFile** устанавливает значение макропеременной %File. Это значение заменяет все макропеременные %File в тексте сообщения об ошибке.

Метод ThrowFile устанавливает значение макропеременной %File перед обработкой конкретной ошибки. Иными словами, ThrowFile объединяет функциональность методов SetFile и Throw в один метод.

Реализация: Присваивает параметр *имя файла* свойству ErrorClass.FileName.

Пример:

```
CREATE(MyFile)
IF ERRORCODE()                          !если ошибка
GlobalErrors.SetFile(NAME(MyFile))      !установим файл
GlobalErrors.Throw(Msg:CreateFailed)    !обработаем ошибку
        END
```

См. также: FileName, ThrowFile

SetId (делает указанную ошибку текущей)

SetId(*код ошибки* **), PROTECTED**

SetId	Делает указанную ошибку текущей.
код ошибки	Целочисленная константа, переменная или выражение, указывающее код текущей ошибки.

Метод **SetID** делает указанную ощибку текущей для обработки методами ErrorClass. Если имеется несколько ошибок с одинаковым кодом, используется последнее определение. Это позволяет определенным позже ошибкам заменять ранее определенные, сохраняя в то же время ранее определенные ошибки в качестве значения макропеременной %Previous.

Этот метод имеет атрибут PROTECTED, следовательно, может быть вызван

только из методов класса ErrorClass или метода порожденного от него класса.

Реализация: Файл ABERROR.INC содержит список стандартных констант для параметра *код ошибки*.

Пример:

```
ErrorClass.TakeError PROCEDURE(SHORT Id)
 CODE
          SELF.SetId(Id)
          CASE SELF.Errors.Fatality
          OF Level:Benign
                                    RETURN SELF.TakeBenign()
          OF Level:User
          OROF Level:Cancel
                                    RETURN SELF.TakeUser()
          OF Level:Program
                                    RETURN SELF.TakeProgram()
          OF Level:Fatal
                                    RETURN SELF.TakeFatal()
          OF Level:Notify
                                    SELF.TakeNotify()
                                    RETURN Level:Notify
          ELSE
                                    RETURN SELF.TakeOther()
          END
```

См. также: Errors

SubsString (подставляет значения макросиволов)

SubsString, PROTECTED

Метод **SubsString** возвращает текущее сообщение об ошибке, выполнив подстановки всех макропеременных.

Этот метод имеет атрибут PROTECTED, следовательно, может быть вызван только из методов класса ErrorClass или метода порожденного от него класса.

Реализация: Методы TakeFatal, TakeNotify, TakeUser и Message вызывают метод SubsString для выполнения макроподстановок.

Тип результата: STRING

Пример:

```
ErrorClass.TakeFatal PROCEDURE
  CODE
MESSAGE(Self.SubsString() &' Press OK to end this application',|
Self.Errors.Title,ICON:Exclamation,Button:OK,BUTTON:OK,0)
          HALT(0,Self.Errors.Title)
          RETURN Level:Fatal
```

См. также: FileName, FieldName, Macro Expansion, Message, MessageText, TakeFatal, TakeNotify, TakeUser

TakeBenign (обработка ошибки уровня Level:Benign)

TakeBenign, PROTECTED, VIRTUAL, PROC

Метод **TakeBenign** вызывается в тех случаях, когда ошибка уровня Level:Benign передается обработчику ошибок (см. Throw, ThrowFile, ThrowMessage).

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса (ErrorClass) могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROTECTED, следовательно, может быть вызван только из методов класса ErrorClass или методов порожденного от него класса.

Эта функция имеет атрибут PROC, поэтому вы можете вызывать ее как процедуру, игнорируя возвращаемый результат.

TakeBenign должен возвращать уровень серьезности.

Реализация: Метод базового класса (ErrorClass.TakeBenign) возвращает Level:Benign.

Тип результата: BYTE

Пример:

```
INCLUDE('ABERROR.INC')
MyErrorClass CLASS(ErrorClass)          !породим класс
TakeBenign FUNCTION,BYTE,VIRTUAL        !новый виртуальный метод
          END
GlobalErrors  MyErrorClass              !объявим объект GlobalErrors
```

```
CODE
GlobalErrors.Init                    !Инициализация
.
.
.
GlobalErrors.Throw(Msg:NoError)      !метод Throw вызывает
                                     !SELF.TakeBenign, это приводит
                                     !к вызову метода порожденного
                                     !класса
.
.
.

MyErrorClass.TakeBenign  FUNCTION    !виртуальный метод порожденного
                                     !класса
         CODE                        !некотрый код
       RETURN Level:Benign
```

См. также: TakeError, Throw, ThrowFile, ThrowMessage

TakeError (обрабатывает указанную ошибку)

TakeError(*код ошибки* **), PROC**

TakeError Ищет указанную ошибку, вызывает подходящий метод для ее обработки, затем возвращает уровень серьезности.

код ошибки Целочисленная константа, переменная или выражение, указывающая код ошибки, требующей обработки.

Метод **TakeError** ищет указанную ошибку, в зависимости от уровня ее серьезности вызывает подходящий метод (TakeLevel) для ее обработки, затем возвращает уровень серьезности.

TakeError работает в предположении, что метод SetErrors был вызван для сохранения текущего состояния ошибки.

Эта функция имеет атрибут PROC, поэтому вы можете вызывать ее как процедуру, игнорируя возвращаемый результат.

Реализация: Файл ABERROR.INC содержит список символических констант для параметра *код ошибки*.

По умолчанию обработчик ошибок распознает шесть различных уровней серьезности ошибок. Метод TakeError вызывает различные виртуальные методы (TakeLevel) для каждого уровня. Это позволяет легко заменить стандартные действия по обработке ошибок на действия, специфичные для вашего приложения. Константы уровней серьезности определены в файле ABERROR.INC. Они перечислены ниже с соответствующими им стандартными действиями:

Level:Benign	нет действий, возвращается Level:Benign
Level:User	выводится сообщение, возвращается Level:Benign или Level:Cancel
Level:Notify	выводится сообщение, возвращается Level:Benign
Level:Fatal	выводится сообщение, программа завершается
Level:Program	обрабатывается как Level:Fatal

любое другое значение обрабатывается как Level:Program

Тип результата: BYTE

См. также: Errors, SetErrors, TakeBenign, TakeNotify, TakeUser, TakeFatal, TakeProgram, TakeOther, Throw

TakeFatal (обработка ошибки уровня Level:Fatal)

TakeFatal, PROTECTED, VIRTUAL, PROC

Метод **TakeFatal** вызывается в тех случаях, когда ошибка уровня Level: Fatal передается обработчику ошибок (см. Throw, ThrowFile, ThrowMessage).

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса (ErrorClass) могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROTECTED, следовательно, может быть вызван только из методов класса ErrorClass или методов порожденного от него класса.

Эта функция имеет атрибут PROC, поэтому вы можете вызывать ее как процедуру, игнорируя возвращаемый результат.

TakeFatal должен возвращать уровень серьезности (если программа не остановлена).

Реализация: Метод базового класса (ErrorClass.TakeFatal) выводит сообщение об ошибке и останавливает программу (операция HALT). Хотя этот метод реально не возвращает результата, оператор RETURN все же необходим во избежание ошибок компиляции.

Тип результата: BYTE
Пример:

```
INCLUDE('ABERROR.INC')
MyErrorClass  CLASS(ErrorClass)                    !породим класс
TakeFatal     FUNCTION,BYTE,VIRTUAL                !новый виртуальный метод
              END
GlobalErrors  MyErrorClass                         !объявим объект GlobalErrors
              CODE
              GlobalErrors.Init                    !Инициализация
              .
              .
              GlobalErrors.Throw(Msg:CreateFailed) !метод Throw вызывает
                                                   !SELF.TakeFatal, это
                                                   !приводит к вызову метода
                                                   !порожденного класса
        .
        .

MyErrorClass.TakeFatal FUNCTION                    !виртуальный метод
                                                   !порожденного
                                                   !класса

        CODE                        !некотрый код
        RETURN Level:Fatal
```

См. также: TakeError, Throw, ThrowFile, ThrowMessage

TakeNotify (process notify error)

TakeNotify, PROTECTED, VIRTUAL

Метод **TakeNotify** вызывается в тех случаях, когда ошибка уровня Level:Notify передается обработчику ошибок (см. Throw, ThrowFile, ThrowMessage).

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса (ErrorClass) могут непосредственно обращаться к этому методу в

порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROTECTED, следовательно, может быть вызван только из методов класса ErrorClass или методов порожденного от него класса.

Реализация: Метод базового класса (ErrorClass.TakeNotify) выводит сообщение об ошибке и не возвращает ничего. Однако обратите внимание, что прочие методы "Throw" возвращают Level:Benign (посредством метода TakeError), в то время, как ошибка Level:Notify уже обработана.

Пример:

```
INCLUDE('ABERROR.INC')
MyErrorClass    CLASS(ErrorClass)                    !породим класс
TakeNotify   FUNCTION,BYTE,VIRTUAL                   !новый виртуальный метод
             END
GlobalErrors  MyErrorClass                           !объявим объект GlobalErrors
             CODE
             GlobalErrors.Init
                                                     !Инициализация
             .
             .
             GlobalErrors.Throw(Msg:CreateFailed)    !метод Throw вызывает
                                                     !SELF. TakeNotify, это
                                                     !приводит к вызову метода

                                          !порожденного класса
             .
             .

MyErrorClass.TakeNotify  FUNCTION         !виртуальный метод порожденного
                                          !класса

             CODE                          !некотрый код
             RETURN
```

См. также: TakeError, Throw, ThrowFile, ThrowMessage

TakeOther (обрабатывает прочие ошибки)

TakeOther, PROTECTED, VIRTUAL, PROC

Метод **TakeNotify** вызывается в тех случаях, когда ошибка нераспознанного уровня передается обработчику ошибок (см. Throw, ThrowFile, ThrowMessage). По умолчанию такие ошибки обрабатываются как программные (Level:Program).

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса (ErrorClass) могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROTECTED, следовательно, может быть вызван только из методов класса ErrorClass или методов порожденного от него класса.

Эта функция имеет атрибут PROC, поэтому вы можете вызывать ее как процедуру, игнорируя возвращаемый результат.

TakeOther должен возвращать уровень серьезности.

Реализация: Метод базового класса (ErrorClass.TakeOther) вызывает TakeProgram.

Тип результата: BYTE

Пример:

```
INCLUDE('ABERROR.INC')
MyErrorClass  CLASS(ErrorClass)                    !породим класс
TakeOther   FUNCTION,BYTE,VIRTUAL                  !новый виртуальный метод
END
GlobalErrors  MyErrorClass                         !объявим объект GlobalErrors
              CODE
              GlobalErrors.Init                    !Инициализация
              .
              .
              GlobalErrors.Throw(Msg:CreateFailed)!метод Throw вызывает
                                                   !SELF.TakeOther, это
                                                   !приводит к вызову метода
                                                   !порожденного класса
              .
              .
```

```
MyErrorClass.TakeOther  FUNCTION          !виртуальный метод порожденного
                                          !класса
        CODE                              !некотрый код
        RETURN Level:Program
```

См. также: TakeError, Throw, ThrowFile, ThrowMessage

TakeProgram (обрабатывает ошибки уровня Level:Program)

TakeProgram, PROTECTED, VIRTUAL, PROC

Метод **TakeNotify** вызывается в тех случаях, когда ошибка уровня Level:Program передается обработчику ошибок (см. Throw, ThrowFile, ThrowMessage). По умолчанию такие ошибки обрабатываются как фатальные (Level:Fatal)

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса (ErrorClass) могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROTECTED, следовательно, может быть вызван только из методов класса ErrorClass или методов порожденного от него класса.

Эта функция имеет атрибут PROC, поэтому вы можете вызывать ее как процедуру, игнорируя возвращаемый результат.

TakeProgram должен возвращать уровень серьезности.

Реализация: Метод базового класса (ErrorClass.TakeProgram) вызывает TakeFatal.

Тип результата: BYTE

Пример:

```
INCLUDE('ABERROR.INC')
MyErrorClass  CLASS(ErrorClass)           !породим класс
TakeProgram  FUNCTION,BYTE,VIRTUAL        !новый виртуальный метод
        END
GlobalErrors  MyErrorClass                !объявим объект GlobalErrors
        CODE
        GlobalErrors.Init                 !Инициализация
        .
```

```
        .
        GlobalErrors.Throw(Msg:CreateFailed)!метод Throw вызывает
                                            !SELF.TakeProgram, это
                                            !приводит к вызову метода
                                            !порожденного класса
        .
        .
MyErrorClass.TakeProgram  FUNCTION          !виртуальный метод порожденного
                                            !класса
        CODE                                !некотрый код
    RETURN Level:Program
```

См. также: TakeError, Throw, ThrowFile, ThrowMessage

TakeUser (обрабатывает ошибки уровня Level:User)

TakeUser, PROTECTED, VIRTUAL, PROC

Метод **TakeUser** вызывается в тех случаях, когда ошибка уровня Level:User передается обработчику ошибок (см. Throw, ThrowFile, ThrowMessage).

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса (ErrorClass) могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROTECTED, следовательно, может быть вызван только из методов класса ErrorClass или методов порожденного от него класса.

Эта функция имеет атрибут PROC, поэтому вы можете вызывать ее как процедуру, игнорируя возвращаемый результат.

TakeUser должен возвращать уровень серьезности, позволяющий проанализировать ответ пользователя.

Реализация: Метод базового класса(ErrorClass.TakeUser) выводит сообщение об ошибке и возвращает Level:Benign либо Level:Cancel в зависимости от ответа пользователя.

Тип результата: BYTE

Пример:

```
INCLUDE('ABERROR.INC')
MyErrorClass  CLASS(ErrorClass)          !породим класс
TakeUser   FUNCTION,BYTE,VIRTUAL         !новый виртуальный метод
           END
GlobalErrors  MyErrorClass               !объявим объект GlobalErrors
           CODE
           GlobalErrors.Init             !Инициализация
           .
           .
GlobalErrors.Throw(Msg:CreateFailed)     !метод Throw вызывает
                                         !SELF.TakeUser, это
                                         !приводит к вызову метода
                                         !порожденного класса
           .
           .
MyErrorClass.TakeUser  FUNCTION          !Виртуальный метод
                                         !порожденного класса
           CODE                          !некоторый код
IF MESSAGE(SELF.SubsString(),SELF.Errors.Title,ICON:Question, |
 Button:Yes+Button:No,BUTTON:Yes,0) = Button:Yes
                                         !некоторый код
           RETURN Level:Benign
           ELSE                          !некоторый код
           RETURN Level:Cancel
           END
```

См. также: TakeError, Throw, ThrowFile, ThrowMessage

Throw (обрабатывает указанную ошибку)

Throw(*код ошибки*), PROC

Throw	Обрабатывает указанную ошибку, затем возвращает ее уровень серьезности.
код ошибки	Целочисленная константа, переменная или выражение, указывающая код ошибки, требующей обработки.

Метод **Throw** обрабатывает указанную ошибку, вызывая другие методы класса ErrorClass, после чего возвращает ее уровень серьезности.

Обычно Throw - это метод, который вызывает ваша программа, встречая известную ошибку. То есть, когда ваша программа встречает ошибки или другие

особые ситуации, она просто вызывает метод Throw или одну из его разновидностей (ThrowFile или ThrowMessage), передавая им соответствующий *код ошибки*. Throw в свою очередь вызывает другие методы ErrorClass, необходимые для обработки этой ошибки.

Эта функция имеет атрибут PROC, поэтому вы можете вызывать ее как процедуру, игнорируя возвращаемый результат.

Реализация: Метод Throw сохраняет состояние ошибки (ERROR, ERRORCODE, FILEERROR или FILEERRORCODE), отыскивает указанную ошибку, вызывает подходящий метод для ее обработки, основываясь на уровне серьезности данной ошибки. По завершении обработки возвращается уровень серьезности.

Файл ABERROR.INC содержит список символических констант для параметра *код ошибки*.

Примечание: Метод Throw не всегда возвращает результат вызывающей программе - в зависимости от уровня ошибки.

Тип результата: BYTE

Пример:

```
                                    !Произошла ошибка пользовательского
                                    !уровня.
                                    !Попросим у пользователя подтверждение
Severity = GlobalErrors.Throw(Msg:ConfirmCancel)
                                    !обработать ошибку
IF Severity = Level:Cancel
        LocalResponse = RequestCancelled
        DO ProcedureReturn
END
```

См. также: Errors, ThrowFile, ThrowMessage

ThrowFile (устанавливает значение макропеременной %File, затем обрабатывает ошибку)

ThrowFile(*код ошибки, имя файла* **), PROC**

ThrowFile Устанавливает значение макропеременной %File, затем обрабатывает ошибку.

код ошибки Целочисленная константа, переменная или выражение, указывающая код ошибки, требующей обработки.

имя файла Строковая константа, переменная или выражение, указывающая файл, послуживший источником ошибки.

Метод **ThrowFile** устанавливает значение макропеременной %File, затем обрабатывает ошибку и в завершение возвращает ее уровень серьезности.

ThrowFile объединяет в одном методе функциональность методов SetFile и Throw.

Эта функция имеет атрибут PROC, поэтому вы можете вызывать ее как процедуру, игнорируя возвращаемый результат.

Реализация: Файл ABERROR.INC содержит список символических констант для параметра *код ошибки*. Значение свойства ErrorClass.FileName подставляется вместо макропеременной %File в тексте сообщения.

Примечание: Метод ThrowFile не всегда возвращает результат вызывающей программе - в зависимости от уровня ошибки.

Тип результата: BYTE

Пример:

```
OPEN(MyFile)
IF ERRORCODE()
Severity = GlobalErrors.ThrowFile(Msg:OpenFailed, NAME(MyFile))
END
```

См. также: FileName, SetFile, Throw

ThrowMessage (устанавливает значение макропеременной %Message, затем обрабатывает ошибку)

ThrowMessage(*код ошибки, текст сообщения* **), PROC**

ThrowMessage Устанавливает значение макропеременной %Message, затем обрабатывает ошибку

код ошибки Целочисленная константа, переменная или выражение, указывающая код ошибки, требующей обработки.

текст сообщения Строковая константа, переменная или выражение, значение которой подставляется вместо макропеременной Message в тексте сообщения.

Метод **ThrowFile** устанавливает значение макропеременной %Message, затем обрабатывает ошибку и в завершение возвращает ее уровень серьезности.

Эта функция имеет атрибут PROC, поэтому вы можете вызывать ее как процедуру, игнорируя возвращаемый результат.

Реализация: Файл ABERROR.INC содержит список символических констант для параметра *код ошибки*. Значение свойства ErrorClass.MessageText подставляется вместо макропеременной %Message в тексте сообщения.

Примечание: Метод ThrowFile не всегда возвращает результат вызывающей программе - в зависимости от уровня ошибки.

Тип результата: BYTE

Пример:

```
OPEN(MyFile)
IF ERRORCODE()
  Severity=GlobalErrors.ThrowMessage(Msg:OpenFailed,NAME(MyFile))
END
```

См. также: MessageText, Throw

Классы обработки пар полей

Обзор

В программах, ориентированных на работу с базами данных, присутствует ряд часто повторяющихся фундаментальных операций. Среди них - сохранение и восстановление значений полей с последующим сравнением текущих и предыдущих значений.

В этой главе описаны два класса библиотеки ABC, которые поддерживают фундаментальные операции с буферами данных. Эти классы полностью универсальны в том смысле, что они могут работать с любыми парами полей, независимо от происхождения этих полей.

Замечание: Основное достоинство этих классов состоит в их универсальности. Они позволяют вам перемещать данные между любыми парами полей и сравнивать их, не зная структур буферов данных, в которых находятся эти поля.

В некотором смысле FieldPairsClass напоминает оператор глубокого присваивания языка Clarion (:=: , описание этого оператора см. В *Описании языка*). Однако FieldPairsClass имеет ряд преимуществ перед глубоким присваиванием:

- метки полей, составляющих пару, не должны в точности совпадать;
- пары полей не ограничены структурами GROUP, RECORD, и QUEUE;
- пары полей не ограничены единственным источником и единственным назначением;
- вы можете сравнивать на равенство наборы полей;
- вы можете имитировать структуру данных, в то время, как реальной структуры не существует.

Недостатком FieldPairsClass является то обстоятельство, что он не работает с массивами (поскольку в основе FieldPairsClass лежит тип данных ANY, который может ссылаться только на объекты простых типов). Более полную информацию о типе ANY вы можете получить в *Описании языка.*

FieldPairsClass (класс пар полей)

FieldPairsClass позволяет вам перемещать данные между парами полей и сравнивать пары полей, обнаруживая изменения, произошедшие с момента последней операции.

Этот класс предоставляет методы, которые позволяют вам идентифицировать или "устанавливать" требуемые пары полей.

Примечание: **Поля, объединенные в пару, не обязаны ни занимать непрерывную область памяти, ни быть частью структуры. Вы можете создать виртуальную структуру, просто добавляя несвязанные поля в объект FieldPairs. Другие методы FieldPairs могут работать с этой виртуальной структурой.**

Идентифицировав пару полей, вы можете перемещать данные в одном направлении (слева направо), вызывая один метод, и в другом направлении (справа налево), вызвая другой метод. Вам следует лишь помнить, какую из сущностей (набор полей) вы описали как "левую", а какую - как "правую". Третий метод сравнивает два набора полей и возвращает результат, показывающий, равны они или нет.

BufferedPairsClass (класс буферизованных пар)

BufferedPairsClass - класс, порожденный от FieldPairsClass, следовательно, он обеспечивает ту же функциональность, однако он предоставляет третий буфер (область сохранения), а также возможность сравнивания сохраненного буфера с первичными. Кроме этого, имеется возможность восстанавления данных в первичных буферах из сохраненного буфера (для выполнения стандартной операции отказа).

Связи с другими ABC-классами

ViewManager и BrowseClass используют FieldPairsClass и BufferedPairsClass для выполнения различных задач.

Реализация ABC-шаблонов

Объекты FieldPairsClass и BufferedPairsClass создаются другими объектами библиотеки ABC в соответствии с их потребностями. Поэтому шаблон не содержит непосредственных ссылок на FieldPairsClass и BufferedPairsClass.

Исходные модули

Классы обработки пар полей по умолчанию устанавливаются в каталог \CLARION4\LIBSRC. Ниже перечислены конкретные файлы и соответствующие компоненты:

ABUTIL.INC Объявления FieldPairsClass и BufferedPairsClass
ABUTIL.CLW Определения методов FieldPairsClass и BufferedPairsClass

Концептуальный пример

Классы обработки пар весьма абстрактны, поэтому мы приводим конкретный пример, который должен облегчить понимание этих классов. Помещенный пример показывает типичную последовательность операторов для объявления, создания, инициализации, использования и завершения объекта FieldPairsClass.

Предположим, файл Customer объявлен таким образом:

```
Customer  FILE,DRIVER('TOPSPEED'),PRE(CUST),CREATE,BINDABLE
ByNumber                          KEY(CUST:CustNo),NOCASE,OPT,PRIMARY
Record                            RECORD,PRE()
CustNo                                  LONG
Name                                    STRING(30)
Phone                                   STRING(20)
Zip                                     DECIMAL(5)
                                  .     .
```

И у вас есть очередь Customer, объявленная так:

```
CustQ     QUEUE
CustNo                            LONG
Name                                    STRING(30)
Phone                                   STRING(20)
Zip                                     DECIMAL(5)
          END
```

И вы хотите перемещать данные между буферами файла и очереди.

```
          INCLUDE('ABUTIL.INC')
FieldPairs Class
Fields    FieldPairsClass         !объявим объект Fields
          CODE
          Fields.Init             !инициализируем объект Fields
Fields.AddPair(CUST:CustNo,       CustQ.CustNo)
```

```
                                                    !организуем пару
CustNo
Fields.AddPair(CUST:Name,CustQ.Name)        !организуем пару Name
Fields.AddPair(CUST:Phone,CustQ.Phone)      !организуем пару Phone
Fields.AddPair(CUST:Zip,CustQ.Zip)          !организуем пару Zip
                                            !некоторый код
Fields.AssignLeftToRight                    !скопировать из файла Customer в
                                              очередь CustQ
                                            !некоторый код
        IF Fields.Equal()                   !сравнить очередь CustQ
                                            !и файл Customer
        MESSAGE('Buffers are equal')
        ELSE
        MESSAGE('Buffers not equal')
        END
!некоторый код
Fields.AssignRightToLeft            !скопировать из очереди CustQ в файл Customer
!некоторый код
Fields.Kill                 !Завершить объект FieldPairs
DISPOSE (Fields)            !Освободить память
```

Свойства FieldPairsClass

Единственное свойство класса FieldPairsClass - список пар полей, распознаваемых объектом FieldPairsClass. Пустой список создается методом инициализации FieldPairsClass.Init. Элементы этого списка добавляются впоследствии методами, предназначенными для этой цели (AddPair и AddItem).

List (распознаваемые пары полей)

```
List    &FieldPairsQueue
```

Свойство **List** является ссылкой на структуру, которая содержит все пары полей, распознаваемые объектом FieldPairsClass. Для добавления пар полей в свойство List служат методы AddPair и AddItem. Для каждой пары полей свойство включает “левое” и “правое” поля.

“Левое” и “правое” назначения отражены в названиях других методов (например, методах присваивания полей - AssignLeftToRight и AssignRightToLeft), поэтому вы можете легко и точно управлять перемещением данных между двумя наборами полей.

Реализация: Свойство List представляет собой ссылку на очередь,

объявленнуюABUTIL.INC таким образом:

```
FieldPairsQueue   QUEUE,TYPE
Left              ANY
Right             ANY
                  END
```

Метод Init создает пустой список List, а метод Kill ликвидирует его. AddPair и AddItem добавляют в него пары полей.

См. Также: AddPair, AddItem, Init

Методы FieldPairsClass

Функциональная организация - ожидаемое использование

Для помощи в понимании FieldPairsClass можно разделить многообразные методы этого класса на две большие категории в соответствии с их ожидаемым использованием - первичный интерфейс и виртуальные методы. Такое структурирование отражает то, что мы понимает под типичным использованием методов FieldPairsClass.

Методы первичного интерфейса

Методы первичного интерфейса, которые вы, скорее всего, явно вызываете в своих программах, могут быть разделены на три категории:

Разового применения:

Init	инициализирует объект FieldPairsClass
AddItem	добавляет пару полей, основываясь на одном поле-источнике
Kill	завершает объект FieldPairsClass

Основного применения:

AssignLeftToRight	присваивает каждое “левое” поле соответствующему “правому”
AssignRightToLeft	присваивает каждое “правое” поле соответствующему “левому”
Equal	возвращает 1, если равенство полей соблюдается для всех пар, и 0, если хотя бы в одной паре равенство нарушено

Эпизодического применения:

ClearLeft	очищает (операция CLEAR) каждое “левое” поле
ClearRight	очищает (операция CLEAR) каждое “правое” поле
EqualLeftRight	возвращает 1, если равенство полей соблюдается для всех пар, и 0, если хоть в одной паре равенство нарушено

Виртуальные методы

Обычно эти методы не вызываются непосредственно. Однако мы предполагаем, что у вас возникнет необходимость переопределить эти методы, и, поскольку они являются виртуальными, сделать это несложно. В противном случае, и их стандартное поведение представляется разумным.

AddPair **добавляет пару полей в свойство List**

AddItem (добавляет пару полей на основании поля-источника)

AddItem(*левое* **)**

AddItem	Добавляет пару полей в свойство List на основании поля-источника
левое	Метка “левого” поля пары. Поле может иметь любой тип, но не может быть массивом.
	Метод **AddItem** добавляет пару полей в свойство List на основании одного поля-источника. “Правое” поле создается автоматически и инициализируется тем же значением, которое имеет “левое” поле. Поля, объединенные в пару, не обязаны ни занимать непрерывную область памяти, ни быть частью структуры. Таким образом, вы можете построить виртуальную структуру, просто добавляя несвязанные поля в объект FieldPairs. Другие методы FieldPairs могут работать с этой виртуальной структурой.
Реализация:	AddItem работает в предположении, что свойство List уже создано при помощи метода Init или какого-либо другого.
	Вызывая метод AddItem для серии полей, вы эффективно создаете две виртуальные структуры, содержащие поля, причем “левые” - это исходные поля, а “правые” в момент вызова AddItem содержат копии данных исходных полей.

Пример:

```
        INCLUDE('ABUTIL.INC')                !объявить класс FieldPairs
DKeyPair FieldPairsClass                     !объявить объект FieldPairs

Org     FILE                                 !объявить файл
DptKey  KEY(Dept,Grade)                      !объявить многокомпонентный ключ
        RECORD
```

```
Dept        SHORT
Mgr         SHORT
Grade       SHORT

                                                    ..
            CODE
DKeyPair.Init                       !инициализация объекта FieldPairs
DKeyPair.AddItem(Org:Dept)          !добавить поле Dept (левое) и
                                    !копию поля Dept (правое)
DKeyPair.AddItem(Org:Grade)         !добавить поле (левое) и
                                    !копию поля Grade (правое)

!некоторый код
DKeyPair.AssignLeftToRight          !сохранить значения ключевых полей
SET(Org:DptKey,Org:DptKey)          !спозиционироваться в файле
NEXT(Org)                           !прочесть (возможно) некоторую
                                    !запись
IF ERRORCODE() OR |                 !проверить результат чтения на
                                    !совпадение
 ~DKeyPair.Equal()                  !сравнением значений ключевых полей
                                    !с сохраненными значениями

MESSAGE('Record not found!')
            END
```

См. Также: Init, List

AddPair (добавляет пару полей)

AddPair(*левое, правое* **), VIRTUAL**

AddPair	Добавляет пару полей в свойство List.
левое	Метка "левого" поля пары. Поле может иметь любой тип, но не может быть массивом.
правое	Метка "правого" поля пары. Поле может иметь любой тип, но не может быть массивом.
Метод **AddPair**	добавляет пару полей в свойство List. Эти поля не обязаны ни занимать непрерывную область памяти, ни быть частью структуры. Таким образом, вы можете построить виртуальную структуру, просто добавляя несвязанные поля в объект FieldPairs. Другие методы FieldPairs могут работать с этой виртуальной структурой. Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.
Реализация:	AddPair работает в предположении, что свойствоList уже создано при помощи метода Init или какого-либо другого. Вызывая AddPair для серии полей (например соответствующих полей

в структуре RECORD и в структуре QUEUE), вы эффективно строите две содержащие поля виртуальные структуры и отношение 1:1 между ними.

Пример:

```
INCLUDE('ABUTIL.INC')                    !объявить класс FieldPairs
Fields      &FieldPairsClass!объявить объект FieldPairs
Customer  FILE,DRIVER('TOPSPEED'),PRE(CUST)
ByNumber KEY(CUST:CustNo),NOCASE,OPT,PRIMARY
Record      RECORD,PRE()
CustNo                                   LONG
Name                                     STRING(30)
Phone                                    STRING(20)
ZIP                                      DECIMAL(5)
            END
CustQ                                    QUEUE
CustNo                                   LONG
Name                                     STRING(30)
Phone                                    STRING(20)
ZIP                                      DECIMAL(5)
            END
CODE
Fields.Init                              !инициализировать объект FieldPairs
Fields.AddPair(CUST:CustNo,CustQ.CustNo)     !создать пару CustNo
Fields.AddPair(CUST:Name,CustQ.Name)         !создать пару Name
Fields.AddPair(CUST:Phone,CustQ.Phone)       !создать пару Phone
Fields.AddPair(CUST:ZIP,CustQ.ZIP)           !создать пару ZIP
```

См. Также: Init, List

AssignLeftToRight (копирует из “левых” полей в “правые”)

AssignLeftToRight

Метод **AssignLeftToRight** копирует содержимое каждого “левого” поля в соответствующее ему “правое”.

Реализация: В случае пар, добавленных при помощи метода AddPairs, “левое” поле является первым (левым) параметром метода AddPair; “правое” поле - второй (правый) параметр метода AddPair. В случае пар, добавленных методом AddItem, “левое” поле - единственный параметр метода AddItem, “правое” - копия “левого”.

Пример:

```
Fields.AddPair(CUST:Name,     CustQ.Name)     !создать пару Name
Fields.AddPair(CUST:Phone,    CustQ.Phone)    !создать пару Phone
Fields.AddPair(CUST:ZIP,CustQ.ZIP)            !создать пару ZIP
!некоторый код
```

```
IF ~Fields.Equal                                    !сравнить поля пар
CASE MESSAGE('Потерять изменения?',,,BUTTON:Yes+BUTTON:No)
        OF BUTTON:No
Fields.AssignRightToLeft                !скопировать изменения в буфер CUST
        OF BUTTON:Yes
Fields.AssignLeftToRight                !восстановить в буфере CustQ
                                        !исходные значения
        END
        END
```

См. Также: AddPair, AddItem, List

AssignRightToLeft (копирует из "правых" полей в "левые")

AssignRightToLeft

Метод **AssignRightToLeft** копирует содержимое каждого "правого" поля в соответствующее ему "левое".

Реализация: В случае пар, добавленных при помощи метода AddPairs, "левое" поле является первым (левым) параметром метода AddPair; "правое" поле - второй (правый) параметр метода AddPair. В случае пар, добавленных методом AddItem, "левое" поле - единственный параметр метода AddItem, "правое" - копия "левого".

Пример:

```
Fields.AddPair(CUST:Name,           CustQ.Name)     !создать пару Name
        Fields.AddPair(CUST:Phone,  CustQ.Phone)    !создать пару Phone
        Fields.AddPair(CUST:ZIP,    CustQ.ZIP)      !создать пару ZIP
!некоторый код
IF ~Fields.Equal                                    !сравнить поля пар
        CASE MESSAGE('Потерять изменения?',,,BUTTON:Yes+BUTTON:No)
        OF BUTTON:No
        Fields.AssignRightToLeft            !скопировать изменения
                                            !в буфер CUST
        OF BUTTON:Yes
        Fields.AssignLeftToRight            !восстановить в буфере CustQ
                                            !исходные значения
          END
    END
```

См. Также: AddPair, AddItem, List

ClearLeft (очищает каждое “левое” поле)

ClearLeft

Метод **ClearLeft** очищает содержимое каждого “левого” поля в свойстве List.

Реализация: В случае пар, добавленных при помощи метода AddPairs, “левое” поле является первым (левым) параметром метода AddPair; “правое” поле - второй (правый) параметр метода AddPair. В случае пар, добавленных методом AddItem, “левое” поле - единственный параметр метода AddItem, “правое” - копия “левого”.
Метод ClearLeft использует оператор CLEAR для очистки поля. Более подробно об операторе CLEAR см. *Описание языка*.

Пример:

```
Fields &= NEW FieldPairsClass          !создать объект FieldPairs
Fields.Init                            !инициализировать его
Fields.AddPair(CUST:CustNo,CustQ.CustNo)     !создать пару CustNo
Fields.AddPair(CUST:Name,CustQ.Name)         !создать пару Name
Fields.AddPair(CUST:Phone,CustQ.Phone)       !создать пару Phone
Fields.AddPair(CUST:ZIP,CustQ.ZIP)           !создать пару ZIP
!некоторый код
IF LocalRequest = InsertRecord
        Fields.ClearLeft               !очистить поля CustQ в нулевые значения
                                       !или заполнить пробелами
        END
```

См. Также: AddPair, AddItem, List

ClearRight (очищает каждое “правое” поле)

ClearRight

Метод **ClearRight** очищает содержимое каждого “левого” поля в свойстве List.

Реализация: В случае пар, добавленных при помощи метода AddPairs, “левое” поле является первым (левым) параметром метода AddPair; “правое” поле - второй (правый) параметр метода AddPair. В случае пар, добавленных методом AddItem, “левое” поле - единственный параметр метода AddItem, “правое” - копия “левого”.
Метод ClearRight использует оператор CLEAR для очистки поля. Более подробно об операторе CLEAR см. *Описание языка*.

Пример:

```
Fields &= NEW FieldPairsClass          !создать объект FieldPairs
Fields.Init                            !инициализировать его
Fields.AddPair(CUST:CustNo,CustQ.CustNo)     !создать пару CustNo
Fields.AddPair(CUST:Name,CustQ.Name)         !создать пару Name
```

```
Fields.AddPair(CUST:Phone,CustQ.Phone)      !создать пару Phone
Fields.AddPair(CUST:ZIP,CustQ.ZIP)          !создать пару ZIP
!некоторый код
IF LocalRequest = InsertRecord
        Fields.ClearRight                   !очистить поля CustQ в нулевые
                                            !значения или  заполнить пробелами
        END
```

См. Также: AddPair, AddItem, List

Equal (возвращает 1, если во всех парах соблюдено равенство полей)

Equal

Метод **Equal** возвращает единицу (1), если во всех парах соблюдено равенство полей, и ноль (0), если равенство нарушено хотя бы для одной пары.

Реализация: Метод Equal просто вызывает метод EqualLeftRight, который и выполняет сравнение. Таким образом, имеются два различных метода, делающих одно и то же.

Это обеспечивает альтернативное соглашение по вызовам для классов FieldPairsClass и BufferedPairsClass. Имя EqualLeftRight согласуется с именами других методов сравнения в классе BufferedPairsClass и служит этой цели. Более подробно см. в разделе *Методы BufferedPairsClass*.

Пример:

```
Fields.AddPair(CUST:Name,CustQ.Name)        !создать пару Name
Fields.AddPair(CUST:Phone,CustQ.Phone)      !создать пару Phone
Fields.AddPair(CUST:ZIP,CustQ.ZIP)          !создать пару ZIP
                                            !некоторый код
IF ~Fields.Equal                            !сравнить поля пар
CASE MESSAGE('Потерять изменения?',,,BUTTON:Yes+BUTTON:No)
        OF BUTTON:No
        Fields.AssignRightToLeft            !скопировать изменения
                                            !в буфер CUST
        OF BUTTON:Yes
           Fields.AssignLeftToRight         !восстановить в буфере CustQ
                                            !исходные значения
           END
   END
```

См. Также: EqualLeftRight

EqualLeftRight (возвращает 1, если во всех парах соблюдено равенство полей)

EqualLeftRight

Метод **EqualLeftRight** возвращает единицу (1), если во всех парах соблюдено равенство полей, и ноль (0) если равенство нарушено хотя бы для одной пары.

Реализация: Метод Equal просто вызывает метод EqualLeftRight, который и выполняет сравнение. Таким образом, имеются два различных метода, делающих одно и то же.
Это обеспечивает альтернативное соглашение по вызовам для классов FieldPairsClass и BufferedPairsClass. Имя EqualLeftRight согласуется с именами других методов сравнения в классе BufferedPairsClass и служит этой цели. Более подробно см. в разделе *Методы BufferedPairsClass*.

Пример:

```
Fields.AddPair(CUST:Name,          CustQ.Name)      !создать пару Name
        Fields.AddPair(CUST:Phone, CustQ.Phone)     !создать пару Phone
        Fields.AddPair(CUST:ZIP,   CustQ.ZIP)       !создать пару ZIP
                                                    !некоторый код
        IF ~Fields.EqualLeftRight                   !сравнить поля пар
        CASE MESSAGE('Потерять изменения?',,,BUTTON:Yes+BUTTON:No)
        OF BUTTON:No
        Fields.AssignRightToLeft            !скопировать изменения
                                            !в буфер CUST
        OF BUTTON:Yes
        Fields.AssignLeftToRight            !восстановить в буфере CustQ
                                    !исходные значения
        END
        END
```

См. Также: Equal

Init (инициализирует объект FieldPairsClass)

Init

Метод **Init** инициализирует объект FieldPairsClass

Реализация: Метод Init создает свойство List.

Пример:

```
INCLUDE('ABUTIL.INC')                   !объявить класс FieldPairs
Fields      &FieldPairsClass            !объявить ссылку на FieldPairs

CODE
```

```
Fields &= NEW FieldPairsClass        !сохдать объект FieldPairs
        Fields.Init                  !инициализировать объект FieldPairs
        .
        .
        Fields.Kill                  !завершить объект FieldPairs
        DISPOSE(Fields)              !освободить выделенную под него память
```

См. Также: Kill, List

Kill (завершает объект FieldPairsClass)

Kill

Метод **Kill** освобождает всю память, выделенную за время жизни объекта и выполняет весь необходимый для завершения код.

Реализация: Метод Kill освобождает память, занимаемую свойством List, которое было создано методом Init.

Пример:

```
INCLUDE('ABUTIL.INC')                !объявить класс FieldPairs
Fields  &FieldPairsClass             !объявить ссылку на FieldPairs

        CODE
Fields &= NEW FieldPairsClass        !сохдать объект FieldPairs
Fields.Init                          !инициализировать объект FieldPairs
        .
        .
        Fields.Kill                  !завершить объект FieldPairs
        DISPOSE(Fields)              !освободить выделенную под него
                                     память
```

См. Также: Init, List

Свойства BufferedPairsClass

Существует два свойства класса BufferedPairsClass - свойство List, унаследованное от класса FieldPairsClass, и свойство RealList ,которое содержит пары полей, распознаваемые объектом BufferedPairsClass. В действительности, существует лишь один список полей, т.к. RealList и List инициализируются ссылкой на одну и ту же структуру.

Пустой список создается инициализационым методом класса BufferedPairsClass.Init. Впоследствии в него добавляются элементы при помощи

метода AddPair.

RealList (распознаваемые пары полей)

RealList &FieldPairsQueue

Свойство **RealList** представляет собой ссылку на структуру, которая содержит все пары полей, распознаваемые объектом BufferedPairsClass. Для добавления пар полей к свойству RealList используется метод AddPair. Для каждой пары полей в свойстве RealList предусмотрены “правое” и “левое” поля, а также поле “буфер”, который можно использовать в качестве промежуточного хранилища данных (области сохранения).

Понятия “левое”,”правое” и “буфер” отражены в названиях методов класса BufferedPairsClass (например, методы присвоения полей - AssignLeftToRight и AssignRightToBuffer), поэтому вы можете легко и точно управлять перемещением данных между тремя наборами полей.

Реализация: В процессе инициализации унаследованное свойство List получает в качестве значения ссылку на RealList. Таким образом, существует лишь один список полей, на который можно ссылаться по метке RealList.

RealList представляет собой ссылку на очередь, объявленную в файле ABUTIL.INC таким образом:

```
BufferedPairsQueue      QUEUE,TYPE
Left                    ANY
Right                   ANY
Buffer                  NY

END
```

Метод Init создает свойства List и RealList; метод Kill ликвидирует их. Метод AddPair добавляет пары полей в свойство RealList.

См. Также: AddPair, Init, Kill

Методы BufferedPairsClass

Функциональная организация - ожидаемое использование

В качестве помощи для понимания BufferedPairsClass, можно разделить многообразные методы этого класса на две большие категории в соответствии с их

ожидаемым использованием - первичный интерфейс и виртуальные методы. Такое структурирование отражает то, что мы понимаем под типичным использованием методов BufferedPairsClass.

Методы первичного интерфейса

Методы первичного интерфейса, которые вы, скорее всего, явно вызываете в своих программах, могут, в свою очередь, быть разделены на три категории:

Разового применения:

Init	инициализирует объект BufferedPairsClass
Kill	завершает объект BufferedPairsClass

Эпизодического применения:

AssignLeftToRight	присваивает каждое “левое” поле соответствующему “правому”
AssignLeftToBuffer	присваивает каждое “левое” поле соответствующему “буферу”
AssignRightToLeft	присваивает каждое “правое” поле соответствующему “левому”
AssignRightToBuffer	присваивает каждое “правое” поле соответствующему “буферу”
AssignBufferToLeft	присваивает каждое “буферное” поле соответствующему “левому”
AssignBufferToRight	присваивает каждое “буферное” поле соответствующему “правому”
EqualLeftRight	возвращает 1, если равенство “левого” и “правого” полей соблюдается для всех пар, и 0, если хотя бы в одной паре равенство нарушено
EqualLeftBuffer	возвращает 1, если равенство “левого” и “буферного” полей соблюдается для всех пар, и 0, если хотя бы в одной паре равенство нарушено
EqualRightBuffer	возвращает 1, если равенство “правого” и “буферного” полей соблюдается для всех пар, и 0, если хотя бы в одной паре равенство арушено
ClearLeft	очищает (операция CLEAR) каждое “левое” поле

ClearRight	очищает (операция CLEAR) каждое "правое" поле

Неприменяемые:

Эти методы наследуются от базового класса и, как правило, не используются в контексте рассматриваемого порожденного (BufferedPairsClass) класса.

AddItem	добавляет пару полей, основываясь на одном поле-источнике
Equal	возвращает 1, если равенство полей соблюдается для всех пар, и 0, если хотя бы в одной паре равенство нарушено

Виртуальные методы

Обычно эти методы не вызываются непосредственно. Однако мы предполагаем, что у вас возникнет необходимость переопределить эти методы, и, поскольку они являются виртуальными, сделать это несложно. В противном случае, и их стандартное поведение представляется разумным.

AddPair	**добавляет пару в свойство**

AddPair (добавляет пару полей)

AddPair(*левое, правое* **), VIRTUAL**

AddPair	Добавляет пару полей в свойство RealList.
левое	Метка "левого" поля пары. Поле может иметь любой тип, но не может быть массивом.
правое	Метка "правого" поля пары. Поле может иметь любой тип, но не может быть массивом.

Метод **AddPair** добавляет пару полей в свойство RealList. При этом автоматически создается третье поле - "буфер", которое может быть использовано для промежуточного хранения данных.

Поля пары не обязаны ни занимать непрерывную область памяти, ни быть частью структуры. Таким образом, вы можете построить виртуальную структуру, просто добавляя несвязанные поля в объект BufferedPairsClass. Другие методы BufferedPairsClass могут работать с этой виртуальной структурой.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: AddPair работает в предположении, что свойство RealList уже создано при помощи метода Init или какого-либо другого.

Вызывая AddPair для серии полей (например, соответствующих полей в структуре RECORD и в структуре QUEUE), вы эффективно строите три содержащие поля виртуальные структуры и отношение 1:1 между ними.

Пример:

```
        INCLUDE('ABUTIL.INC')                !объявить класс BufferedPairsClass
Fields  &BufferedPairsClass                  !объявить объект BufferedPairsClass
                                             !(ссылку)
Customer FILE,DRIVER('TOPSPEED'),PRE(CUST)
ByNumber KEY(CUST:CustNo),NOCASE,OPT,PRIMARY
Record  RECORD,PRE()
CustNo  LONG
Name    STRING(30)
Phone   STRING(20)
        END
CustQ   QUEUE
CustNo  LONG
Name    STRING(30)
Phone   STRING(20)
        END
        CODE
Fields &= NEW BufferedPairsClass      !создать объект BufferedPairs
Fields.Init                           !инициализировать объект BufferedPairsClass
Fields.AddPair(CUST:CustNo,CustQ.CustNo)    !создать пару CustNo
Fields.AddPair(CUST:Name,CustQ.Name)        !создать пару Name
Fields.AddPair(CUST:Phone,CustQ.Phone)      !создать пару Phone
```

См. Также: Init, RealList

AssignBufferToLeft (копирует данные из "буферных" в "левые" поля)

AssignBufferToLeft

Метод **AssignBufferToLeft** копирует содержимое каждого "буферного" поля в соответствующее "левое" поле свойства RealList.

Реализация: "Левое" поле - первый (левый) параметр метода AddPair. "Правое" поле - второй (правый) параметр метода AddPair. Класс BufferedPairsClass автоматически поддерживает поле "буфер".

Пример:

```
        Fields.AddPair(CUST:Name,    CustQ.Name)        !создать пару Name
```

```
        Fields.AddPair(CUST:Phone,    CustQ.Phone)      !создать пару Phone
        Fields.AddPair(CUST:ZIP,      CustQ.ZIP)        !создать пару ZIP
        !некоторый код
        IF ~Fields.EqualRightBuffer   !сравнить поля QUEUE
                                      !и сохраненные в буфере значения
CASE MESSAGE('Потерять изменения?',,,BUTTON:Yes+BUTTON:No)
        OF BUTTON:No
        Fields.AssignRightToLeft      !скопировать изменения
                                      !в буфер CUST
        OF BUTTON:Yes
        Fields.AssignBufferToRight    !восстановить в буфере CustQ
                                      !исходные значения
        END
        END
```

См. Также: AddPair, RealList

AssignBufferToRight (копирует данные из "буферных" полей в "правые" поля)

AssignBufferToRight

Метод **AssignBufferToRight** копирует содержимое каждого "буферного" поля в соответствующее "правое" поле свойства RealList.

Реализация: "Левое" поле - первый (левый) параметр метода AddPair. "Правое" поле - второй (правый) параметр метода AddPair. Класс BufferedPairsClass автоматически поддерживает поле "буфер".

Пример:

```
        Fields.AddPair(CUST:Name,     CustQ.Name)       !создать пару Name
        Fields.AddPair(CUST:Phone,    CustQ.Phone)      !создать пару Phone
        Fields.AddPair(CUST:ZIP,      CustQ.ZIP)        !создать пару ZIP
        !некоторый код
        IF ~Fields.EqualRightBuffer   !сравнить поля QUEUE
                                      !и сохраненные в буфере значения
    CASE MESSAGE('Потерять изменения?',,,BUTTON:Yes+BUTTON:No)
        OF BUTTON:No
                Fields.AssignRightToBuffer
        OF BUTTON:Yes
        Fields.AssignBufferToRight
        END
        END
```

См. Также: AddPair, RealList

AssignLeftToBuffer (копирует данные из “левых” полей в “буферные” поля)

AssignLeftToBuffer

Метод **AssignLeftToBuffer** копирует содержимое каждого “левого” поля в соответствующее “буферное” поле свойства RealList.

Реализация: “Левое” поле - первый (левый) параметр метода AddPair. “Правое” поле - второй (правый) параметр метода AddPair. Класс BufferedPairsClass автоматически поддерживает поле “буфер”.

Пример:

```
Fields.AddPair(CUST:Name,    CustQ.Name)     !создать пару Name
Fields.AddPair(CUST:Phone,   CustQ.Phone)    !создать пару Phone
Fields.AddPair(CUST:ZIP,     CustQ.ZIP)      !создать пару ZIP
!некоторый код
IF ~Fields.EqualRightBuffer  !сравнить поля QUEUE
                             !и сохраненные в буфере значения
CASE MESSAGE('Потерять изменения?',,,BUTTON:Yes+BUTTON:No)
OF BUTTON:No
Fields.AssignRightToLeft
OF BUTTON:Yes
Fields.AssignLeftToBuffer
END
END
```

См. Также: AddPair, RealList

AssignRightToBuffer (копирует данные из “правых” полей в “буферные” поля)

AssignRightToBuffer

Метод **AssignRightToBuffer** копирует содержимое каждого “правого” поля в соответствующее “буферное” поле свойства RealList.

Реализация: “Левое” поле - первый (левый) параметр метода AddPair. “Правое” поле - второй (правый) параметр метода AddPair. Класс BufferedPairsClass автоматически поддерживает поле “буфер”.

Пример:

```
Fields.AddPair(CUST:Name,    CustQ.Name)     !создать пару Name
Fields.AddPair(CUST:Phone,   CustQ.Phone)    !создать пару Phone
Fields.AddPair(CUST:ZIP,     CustQ.ZIP)      !создать пару ZIP
```

```
                                                    !некоторый код
    IF ~Fields.EqualRightBuffer                     !сравнить поля QUEUE
                                                    !и сохраненные в буфере значения
    CASE MESSAGE('Потерять изменения?',,,BUTTON:Yes+BUTTON:No)
    OF BUTTON:No
    Fields.AssignRightToBuffer
    OF BUTTON:Yes
    Fields.AssignBufferToRight
    END
    END
```

См. Также: AddPair, RealList

EqualLeftBuffer (сравнивает "левые" поля с "буферными")

EqualLeftBuffer

Метод **EqualLeftBuffer** возвращает единицу (1), если равенство "левого" и "буферного" полей соблюдается для всех пар, и ноль (0), если хотя бы в одной паре равенство нарушено.

Реализация: "Левое" поле - первый (левый) параметр метода AddPair. "Правое" поле - второй (правый) параметр метода AddPair. Класс BufferedPairsClass автоматически поддерживает поле "буфер".

Пример:

```
    Fields.AddPair(CUST:Name,   CustQ.Name)     !создать пару Name
    Fields.AddPair(CUST:Phone,  CustQ.Phone)    !создать пару Phone
    Fields.AddPair(CUST:ZIP,    CustQ.ZIP)      !создать пару ZIP
                                                !некоторый код
    IF ~Fields.EqualLeftBuffer  !сравнить поля CUST
                                !и сохраненные в буфере значения
    CASE MESSAGE('Потерять изменения?',,,BUTTON:Yes+BUTTON:No)
    OF BUTTON:No
    Fields.AssignRightToLeft                    !скопировать изменения в CUST
      OF BUTTON:Yes
      Fields.AssignBufferToRight                !восстановить исходные
                                                !значения в CustQ
      END
  END
```

См. Также: AddPair, RealList

EqualRightBuffer (сравнивает "правые" поля с "буферными")

EqualRightBuffer

Метод **EqualRightBuffer** возвращает единицу (1), если равенство "правого" и "буферного" полей соблюдается для всех пар, и ноль (0), если хотя бы в одной паре равенство нарушено.

Реализация: "Левое" поле - первый (левый) параметр метода AddPair. "Правое" поле - второй (правый) параметр метода AddPair. Класс BufferedPairsClass автоматически поддерживает поле "буфер".

Пример:

```
Fields.AddPair(CUST:Name,    CustQ.Name)      !создать пару Name
Fields.AddPair(CUST:Phone,   CustQ.Phone)     !создать пару Phone
Fields.AddPair(CUST:ZIP,     CustQ.ZIP)       !создать пару ZIP
                                              !некоторый код
IF ~Fields.EqualRightBuffer  !сравнить поля QUEUE
                             !и сохраненные в буфере значения
CASE MESSAGE('Потерять изменения?',,,BUTTON:Yes+BUTTON:No)
OF BUTTON:No
Fields.AssignRightToLeft     !скопировать изменения в CUST
OF BUTTON:Yes
Fields.AssignBufferToRight   !восстановить исходные
                             !значения в CustQ
END
END
```

См. Также: AddPair, RealList

Init (инициализирует объект BufferedPairsClass)

Init

Метод **Init** инициализирует объект BufferedPairsClass.

Реализация: Метод **Init** создает свойства List и RealList. Свойство List получает в качестве значения ссылку на RealList. Таким образом, существует лишь один список полей, на который можно ссылаться по метке RealList.

Пример:

```
          INCLUDE('ABUTIL.INC')     !объявить класс BufferedPairsClass
Fields    &BufferedPairsClass       !объявить ссылку на BufferedPairsClass

          CODE
Fields &= NEW BufferedPairsClass    !создать объект
                                    !BufferedPairsClass
          Fields.Init               !инициализировать объект
```

```
                                !BufferedPairsClass
        .
        .
        Fields.Kill             !завершить объект BufferedPairsClass
        DISPOSE(Fields)         !освободить выделенную под него память
```

См. Также: Kill, List, RealList

Kill (завершает объект BufferedPairsClass)

Kill

Метод **Kill** освобождает всю память, выделенную за время жизни объекта и выполняет весь необходимый для завершения код.

Реализация: Метод Kill ликвидирует свойства List и RealList, которые были созданы методом Init.

Пример:

```
        INCLUDE('ABUTIL.INC')       !объявить класс BufferedPairsClass
Fields  &BufferedPairsClass         !объявить ссылку на BufferedPairsClass

        CODE
Fields &= NEW BufferedPairsClass    !создать объект
                                    !BufferedPairsClass
        Fields.Init                 !инициализировать объект
                                    !BufferedPairsClass
        .
        .
        Fields.Kill                 !завершить объект BufferedPairsClass
        DISPOSE(Fields)             !освободить выделенную под него память
```

См. Также: Init, List, RealList

Класс FileManager

Обзор

Класс FileManager - диспетчер работы с файлами, который согласованно и гибко выполняет все типовые операции с данным файлом. FileManagerClass предоставляет методы настройки, которые позволяют вам описать файл и его ключи, а также другие методы для открытия, чтения и закрытия файла.

Диспетчер файлов автоматически обрабатывает автоинкрементные ключи. ABC-шаблоны используют средства этого класса для реализации проверки корректности значений полей и некоторых других особенностей обработки файла в соответствии с настройками в Clarion-словаре или генераторе приложений.

Однако, если даже вы не используете словарь базы данных, генератор приложений, или не указываете правил проверки корректности значений, диспетчер файлов может компетентно и эффективно выполнять типовые операции с вашими файлами.

Примечание: **Класс FileManager работает с отдельными файлами; он не поддерживает ссылочную целостность между связанными файлами. Вопросами ссылочной целостности занимается класс RelationManager.**

Двусторонний подход к операциям с базами данных

Методы FileManager, выполняющие стандартные операции с базами данных, существуют в двух версиях - простой (или интерактивной) и версии “Try” (или “тихой”).

Интерактивные операции с базами данных

При вызове любого из этих методов (Open, Fetch, Next, Previous, Insert и Update) может быть предпринято несколько попыток выполнить требуемую операцию, причем при необходимости выводятся сообщения об ошибках. Эти методы могут запрашивать у конечного пользователя информацию, необходимую для выполнения требуемой задачи. При соответствующих обстоятельствах они могут даже прекратить выполнение программы. Это означает, что программист может считать, что если метод вернул управление, то он успешно отработал.

“Тихие” операции с базами данных

Названия этих методов имеют префикс Try (TryOpen, TryFetch, TryNext, TryPrevious, TryInsert и TryUpdate). Будучи вызванными, они делают единственную попытку выполнить требуемое действие, после чего возвращают результат, показывающий, насколько успешно завершилась операция. Этот результат может быть обработан вызвающей процедурой.

Отношения с другими классами

FileManager использует средства класса ErrorClass для обработки большинства ошибок. Таким образом, если в вашей программе имеются объявления FileManager, то в ней должен быть объявлен и ErrorClass. Более подробно см. *ErrorClass - обработчик ошибок*.

Более важно, возможно, то, что FileManager служит основой или “мальчиком на побегушках” для класса RelationManager. Если в вашей программе имеются объявления RelationManager, то в ней должен быть объявлен и FileManager. Более подробно см. *RelationManager*.

Диспетчер файлов и многопоточные файлы

FileManager также поддерживает работу с файлами на нескольких потоках. Это означает, что несколько MDI-процедур могут обращаться к одному и тому же файлу в одно и тоже время, причем каждая из процедур оперирует своим буфером записи и позиционной информацией, что позволяет избежать конфликтов между процедурами.

Для того, чтобы получить эту степень свободы между MDI-процедурами, вам всего лишь надо добавить атрибут THREAD к описанию вашего файла (см. *Описание языка* для получения более подробной информации), затем создать глобальный объект для каждого файла. Этот глобальный объект автоматически обеспечивает работу на различных потоках, так что вы можете пользоваться им в любых процедурах, обращающихся к файлу. ABC-шаблоны генерируют точно такой же код для файлов с атрибутом THREAD.

Если вы хотите иметь доступ к файлу с единственным разделяемым между потоками буфером, просто опустите атрибут THREAD в описании файла и опять-таки заведите глобальный объект FileManager для каждого файла. Это позволит всем процедурам вашей программы обращаться к файлу с единственным буфером записи и набором позиционной информации.

Реализация ABC-шаблонов

Имеется ряд важных замечаний по поводу того, как реализована работа с FileManager в ABC-шаблонах.

Во-первых, ABC-шаблон порождает от FileManager класс для каждого файла, с которым работает приложение. Порожденный класс называется Hide:Access:*filename*, но на него можно ссылаться как на Access:*filename*. Эти порожденные классы и их методы объявлены в генерируемых файлах *appna*BC0.CLW - *appna*BC9.CLW (их количество зависит от того, сколько файлов использует ваше приложение). Методы порожденных классов имеют связанную с конкретными файлами специфику, и реализуют многие из свойств, указанных в словаре базы данных, как, например, режим открытия, инициализацию значений полей и проверку их корректности и т.д.

Во-вторых, ABC-шаблоны генерируют процедуры для инициализации и завершения объектов FileManager. Это процедуры DctInit и DctKill. Они размещены в файле *appna*BC.CLW.

В-третьих, порожденные от FileManager классы настраиваются при помощи диалога **Global Properties**. Более подробно см. *Обзор шаблонов - Опции управления файлами* и *Опции Классов*.

В заключение, ABC-шаблоны также порождают для каждого файла класс от RelationManager. Эти объекты носят название Hide:Relate:*filename*, но на них можно ссылаться как на Relate:*filename*. Сгенерированный шаблоном код редко непосредственно вызывает методы порожденных от FileManager классов. Вместо этого вызываются методы RelationManager, которые вызывают соответствующие (для связанных файлов) методы порожденных от FileManager классов. Более подробно о RelationManager см. раздел *RelationManager*.

Исходные модули

Исходные модули класса FileManager по умолчанию устанавливаются в каталог \CLARION4\LIBSRC. Конкретные файлы и соответствующие компоненты перечислены ниже:

ABFILE.INC	Объявления FileManager
ABFILE.CLW	Определения методов FileManager

Концептуальный пример

Пример, приведеденный ниже, иллюстрирует типичную последовательность операторов, объявляющих, создающих, инициализирующих, использующих и завершающих объект FileManager.

Этот пример использует FileManager для добавления верной записи с автоинкрементным ключом.

```
            PROGRAM

            INCLUDE('ABFILE.INC')              !объявляем класс FileManager
            MAP

            END

GlobalErrors ErrorClass                        !объявляем объект GlobalErrors
Access:Client CLASS(FileManager)               !порождаем объект Access:Client
Init        PROCEDURE
                                               !Инициализация объекта Access:File
PrimeRecord PROCEDURE,BYTE,PROC,VIRTUAL
                                               !автоинкремент при добавлении
                                               !записи
ValidateField PROCEDURE(UNSIGNED Id),BYTE,VIRTUAL

                                               !проверка поля
ValidateRecord PROCEDURE(<*UNSIGNED Id>),BYTE,VIRTUAL
                                               !проверка всех полей
            END

Client
FILE,DRIVER('TOPSPEED'),PRE(CLI),CREATE,BINDABLE,THREAD
IDKey
KEY(CLI:ID),NOCASE,OPT,PRIMARY
NameKey   KEY(CLI:Name),DUP,NOCASE
Record      RECORD,PRE()
ID          LONG
Name                                  STRING(20)
StateCode                             STRING(2)
END
            END

InsertWindow  WINDOW('Add a new Client'), |
```

```
  AT(,,159,73),IMM,SYSTEM,GRAY
PROMPT('&Name:'),AT(8,20),USE(?CLI:Name:Prompt)
ENTRY(@s20),AT(61,20,84,10),USE(CLI:Name),| MSG('Client Name'),REQ
PROMPT('State Code:'),AT(8,34),| USE(?CLI:StateCode:Prompt)
ENTRY(@s2),AT(61,34,40,10),|          USE(CLI:StateCode),MSG('State Code')
BUTTON('OK'),AT(12,53,45,14),USE(?OK),DEFAULT
          END

          CODE
          OPEN(InsertWindow)
          GlobalErrors.Init              !инициализация объекта GlobalErrors
          Access:Client.Init             !инициализация объекта Access:Client
          Access:Client.Open             !открыть файл Client
          IF Access:Client.PrimeRecord()
                                         !Выполним автоинкремент
          POST(Event:CloseWindow)        !если не вышло - закроем окно
          END
          ACCEPT
CASE FIELD()
          OF ?OK
          IF EVENT() = Event:Accepted          !по кнопке OK
          IF Access:Client.Insert() = Level:Benign
                                               !добавим новую запись
          POST(Event:CloseWindow)  !если вышло - закроем окно
          ELSE                                 !в противном случае
          SELECT(?CLI:Name:Prompt)
                                               !выберем поле - имя клиента
                                    CYCLE      !и все сначала
          END
          END
          OF ?CLI:StateCode                    !на поле StateCode
          IF EVENT() = EVENT:Accepted
          IF Access:Client.ValidateField(3)
                                               !проверим поле StateCode (3-е по счету)
          SELECT(?CLI:StateCode)
                                               !значение неверно - переход к StateCode
          CYCLE
                                               !и все сначала
          .                              .     .          .

          Access:Client.Close            !закрыть файл Client
        Access:Client.Kill               !завершить объект Access:Client
        GlobalErrors.Kill                !завершить объект GlobalErrors
```

```
            RETURN

Access:Client.Init  PROCEDURE
            CODE
            PARENT.Init(Client, GlobalErrors)
                                                !вызов метода Init базового класса
            SELF.FileNameValue = 'Client'       !установим имя файла
            SELF.Buffer &= CLI:Record           !Access:Client ссылается на буфер
                                                !файла Client
            SELF.AddKey(CLI:IDKey,'Client ID',1) !пишем первичный ключ с
                                                !автоинкрементом
            SELF.AddKey(CLI:NameKey,'Client Name')
                                                !опишем другой ключ

Access:Client.PrimeRecord PROCEDURE             !вызвается методом Insert базового
                                                !класса
Result BYTE,AUTO
            CODE
            Result = PARENT.PrimeRecord()       !вызов метода PrimeRecord
                                                !базового класса
            CLI:StateCode = 'FL'
            RETURN Result
Access:Client.ValidateField PROCEDURE(UNSIGNED Id)
            !вызывается методом ValidateFields базового класса
            CODE                    !и данной программой
            IF ID = 3               !проверка поля Statecode (3-его посчету)
            GlobalErrors.SetField('StateCode')
                                    !установим имя поля на случай ошибки
            IF ~CLI:StateCode       !если поле оставлено пустым
            RETURN SELF.Throw(Msg:FieldNotInList)
                                    !передать управление обработчику ошибок
            END
            END
            RETURN Level:Benign

Access:Client.ValidateRecord PROCEDURE(<*UNSIGNED F>)
                                    !вызывается методом Insert базового класса
            CODE
            RETURN SELF.ValidateFields(1,3,F)
                                    !проверить все три поля
```

Свойства класса FileManager

Свойства класса FileManager включают как ссылки на обрабатываемый файл, так и различные флаги и переключатели, задающие способы обработки этого файла.

Имеются ссылки на сам файл, его имя и буфер записи. Эти ссылки позволяют объекту абстрактного класса FileManager обрабатывать конкретный файл.

Переключателям задают режим доступа (разделения), создавать ли файл, режим захвата записей и параметр ожидания локирования файла.

Каждое из этих свойств полностью описано ниже

AliasedFile (буфер записи)

AliasedFile &FileManager

Свойство **AliasedFile** - это ссылка на FileManager файла, псевдонимом которого является данный файл. FileManager использует это свойство для синхронизации команд, буферов и т.д. между обрабатываемым файлом и его псевдонимом.

Реализация: Если обрабатываемый файл имеет файл-псевдоним, вам следует инициализировать свойство AliasedFile после вызова метода Init или в порожденном методе Init, отражающем специфику обрабатываемого файла. См *Концептуальный пример*.

Buffer (буфер записи)

Buffer &GROUP, PROTECTED

Свойство **Buffer** - ссылка на буфер записи обрабатываемого файла. Вы можете использовать это свойство для доступа к записи файла из порожденного класса.

Это свойство имеет атрибут PROTECTED, следовательно, к нему могут обращаться только методы класса FileManager или порожденного от него.

Реализация: Метод SaveBuffer сохраняет копию текущего содержимого буфера в свойство Buffers; сохраненная информация может быть извлечена при помощи метода RestoreBuffer.

Вам следует инициализировать свойство Buffer после вызова метода Init или в порожденном методе Init, отражающем специфику обрабатываемого файла. См *Концептуальный пример*.

См. Также: Buffers, RestoreBuffer, SaveBuffer

Buffers (сохраненные буферы записей)

Buffers &BufferQueue, PROTECTED

Свойство **Buffers** содержит сохраненные копии буфера записи обрабатываемого файла. Сохраненные образы записи могут использоваться для обнаружения изменений, сделанных с других рабочих станций, для выполнения операции отмены и т.д.

Это свойство имеет атрибут PROTECTED, следовательно, к нему могут обращаться только методы класса FileManager или порожденного от него.

Реализация: Метод SaveBuffer сохраняет копию текущего содержимого буфера в свойство Buffers, возвращая идентификатор, который может служить методу RestoreBuffer в качестве управляющего параметра при извлечении сохраненной информации.

Метод RestoreBuffer освобождает память, захваченную методом SaveBuffer. Следовательно, для предотвращения потерь памяти каждый вызов метода SaveBuffer должен сопровождаться вызовом метода RestoreBuffer.

Свойство Buffers представляет собой ссылку на очередь, объявленную в файле ABFILE.INC следующим образом:

```
BufferQueue  QUEUE,TYPE                !Сохраненные записи
Id           LONG                      !Код для распознавания сохраненного
                                       !экземпляра
Buffer       &STRING                   !ссылка на сохраненную запись
                                                              END
```

См. Также: Buffer, SaveBuffer, RestoreBuffer

Create (флаг создания файла)

Create BYTE

Свойство **Create** содержит значение, которое указывает диспетчеру файла, создавать ли файл, если он не существует.

Если значение равно единице (1) - файл создается, если оно равно нулю (0) - файл не создается.

Реализация: Метод Init устанавливает свойство Create равным единице (1), что приводит к автоматичекому созданию файла. ABC-шаблоны изменяют это стандартное поведение в соответствии с установками в словаре базы данных или в генераторе приложений. Более подробно см. *Обзор шаблонов - Работа с файлами*.

Метод Open создает файл, если попытка открыть его завершается неудачей из-за отсутствия файла.

См. Также: Init, Open

File (обрабатываемый файл)

File &FILE

Свойство **File** представляет собой ссылку на обрабатываемый файл, идентифицируя тем самым его для различных методов FileManager.

Реализация: Метод Init устанавливает значение свойства File.

См. Также: Init

FileName (переменное имя файла)

FileName ANY, PROTECTED

Свойство **FileName** представляет собой ссылку на переменную, указанную в атрибуте NAME обрабатываемого файла. Свойство FileName определяет, какой DOS/Windows файл служил предметом обработки объекта FileManager. Это свойство может также быть использовано при выводе сообщенияй об ошибках, а также в других интерфесных целях.

Это свойство имеет атрибут PROTECTED, следовательно к нему могут обращаться только методы класса FileManager или порожденного от него.

Метод SetName задает свойства FileName. Метод GetName возвращает имя файла.

Реализация: Вы должны инициализировать свойство FileName или свойство FileNameValue (но ни в коем случае - оба этих свойства) после вызова метода Init или в методе Init порожденного класса, отражающем специфику обрабатываемого файла. См. *Концептуальный пример*.

Пример:

```
PROGRAM
            INCLUDE('ABFILE.INC')          !объявить ккласс FileManager
            MAP
END
GlobalErrors ErrorClass                    !объявить объект GlobalErrors
Access:Client CLASS(FileManager)           !породить объект Access:Client
Init        PROCEDURE                      !прототип метода Init
            END
ClientFileName STRING('Client01.tps')      !переменная - имя файла
Client      FILE,DRIVER('TOPSPEED'),NAME(ClientFileName)
                                           !файл с переменным именем
Record      RECORD,PRE()
ID          LONG
Name        STRING(20)
. .
            CODE
            GlobalErrors.Init
            Access:Client.Init             !program code
Access:Client.Init  PROCEDURE              !инициализация объекта Access:Client
            CODE
            PARENT.Init(GlobalErrors)      !вызов метода Init базового класса
            SELF.File  &= Client                  !установить свойство File
            SELF.FileName &= ClientFileName       !установить переменное имя файла
```

См. Также: FileNameValue, GetName, SetName

FileNameValue (постоянное имя файла)

FileNameValue STRING(File:MaxFilePath), PROTECTED

Свойство **FileNameValue** содержит константу, значение которой указано в атрибуте NAME обрабатываемого файла. Это свойство служит в качестве имени файла при выводе сообщений об ошибках, а также в других интерфейсных целях.

Это свойство имеет атрибут PROTECTED, следовательно, к нему могут обращаться только методы класса FileManager или порожденного от него.

Метод GetName возвращает имя файла.

Реализация: Вы должны инициализировать свойство FileName или свойство FileNameValue (но ни в коем случае - оба этих свойства) после вызова метода Init или в методе Init порожденного класса, отражающем специфику обрабатываемого файла. См. *Концептуальный пример*.

Пример:

```
PROGRAM
        INCLUDE('ABFILE.INC')      !объявить ккласс FileManager
        MAP
        END

GlobalErrors ErrorClass                 !объявить объект GlobalErrors
Access:Client CLASS(FileManager)        !породить объект Access:Client
Init    PROCEDURE                       !прототип метода Init
        END
Client
FILE,DRIVER('TOPSPEED'),NAME('Client.TPS')
                                        !постоянное имя файла
Record  RECORD,PRE()
ID      LONG
Name    STRING(20)
.       .
        CODE
        GlobalErrors.Init
        Access:Client.Init
        !program code
Access:Client.Init  PROCEDURE           !инициализация объекта Access:Client
    CODE
    PARENT.Init(GlobalErrors)           !вызов метода Init базового класса
    SELF.File    &= Client              !установить свойство File
    SELF.FileNameValue = 'Client.TPS'
                                        !установить постоянное имя файла
```

См. Также: FileName, GetName, SetName

LazyOpen (задержка открытия файла до моменнта реального доступа)

LazyOpen BYTE

Свойство **LazyOpen** указывает, открывать ли обрабатываемый файл непосредственно при открытии связанного файла или отложить это действие до момента реального доступа. Значение, равное единице (1 или True), задерживает открытие, значение, равное нулю (0 или False) -открывает файл немедленно.

Задержка открытия может повысить производительность, если обрабатывается лишь один из группы связанных файлов.

Реализация: Метод Init устанавливает значение свойства LazyOpen в True. ABC-шаблоны изменяют это стандартное поведение в соответствии с установками в генераторе приложений. Более подробно см. *Обзор шаблонов - Работа с файлами.*

Различные методы доступа к файлу (Open, TryOpen, Fetch, TryFetch, Next, TryNext, Insert, TryInsert, и т. д.) используют метод UseFile для реализации действий, задаваемых свойством LazyOpen.

См. Также: Init, Open, TryOpen, Fetch, TryFetch, Next, TryNext, Insert, TryInsert, UseFile

LockRecover (параметр времени ожидания выражения /RECOVER)

LockRecover SHORT

Свойство **LockRecover** содержит параметр времени ожидания для строки /RECOVER драйвера баз данных Clarion. Более подробную информацию о строке драйвера /RECOVER см. *Драйверы баз данных - Clarion.*

Реализация: Метод Init устанавливает свойство LockRecover равным десяти (10) секундам. ABC-шаблоны изменяют это стандартное значение в соответствии с установками в генераторе приложений. Более подробно см. *Обзор шаблонов - Работа с файлами.*

Метод Open пытается начать восстановление файла, если его открытие завершилось неудачей из-за того, что файл оказался залокирован. Более подробно об операции LOCK см. *Описание языка.*

См. Также: Init, Open

OpenMode (режим доступа к файлу)

OpenMode BYTE

Свойство **OpenMode** содержит значение, которое определяет уровень доступа, предоставляемого как пользователю, открывающему файл, так и другим пользователям в сети.

Реализация: Метод Init устанавливает значение свойства OpenMode равным шестнадцатиричному числу 42h (чтение + запись / нет запрета). ABC-шаблоны переопределяют это умолчание в соответствии с установками генератора приложений. Более подробно см. *Обзор шаблонов*.

Метод Open использует свойство OpenMode при открытии файла. Более подробно об операторе OPEN и режимах доступа см. *Описание языка*.

См. Также: Init, Open

SkipHeldRecords (управление обработкой захваченных записей)

SkipHeldRecords BYTE

Свойство **SkipHeldRecords** содержит значение, определяющее, как диспетчер файла должен реагировать, когда он встречает захваченные записи. Более подробно об операторе HOLD см. *Описание языка*.

Значение, равное единице (1), означает, что следует пропустить захваченную запись и продолжить обработку; если значение равно нулю (0), текущая операция обрывается.

Реализация: Метод Init устанавливает значение свойства SkipHeldRecords равным нулю (0).

Методы Next, TryNext, Previous, и TryPrevious действуют в соответствии со значением свойства SkipHeldRecords, когда попытка чтения записи завершается неудачей из-за того, что запись захвачена.

См. Также:: Init, Next, Previous, TryNext, TryPrevious

Методы класса FileManager

Соглашение по именам и два подхода к операциям с базами данных

В процессе изучения функциональной организации класса FileManager следует иметь в виду: большинство наиболее общих операций с базами данных (Open, Next, Previous, Fetch, Insert и Update) существуют в двух версиях. Эти версии легко различаются, т.к. их названия основаны на следующем соглашении по именам:

Операция	Выполняется *Операция* и обрабатываются любые ошибки (автоматическая версия)
Try*Операция*	Выполняется *Операция,* но обработка ошибок отсутствует (ручная версия)

Интерактивные операции с базами данных

Когда вызывается любой из этих методов (Open, Fetch, Next, Previous, Insert и Update), может быть сделано несколько попыток выполнить требуемую операцию, при этом могут выводиться сообщения об ошибках. Эти методы предоставляют автоматическую обработку ошибок. Они могут запрашивать у конечного пользователя информацию о том, как следует выполнять требуемую задачу. При наличии веских причин, они могут даже оборвать выполнение программы. Это означает, что программист может считать, что если метод вернул управление, значит он успешно отработал.

"Бесшумные" операции с базами данных

При вызове любого из методов, названия которых начинаются с "Try" (TryOpen, TryFetch, TryNext, TryPrevious, TryInsert и TryUpdate), выполняется единственная попытка выполнения требуемой операции, после чего возвращается результат, по которому можно судить, успешно ли завершилась операция. Эти методы требуют ручной обработки ошибок.

Функциональная организация - ожидаемое использование

В качестве помощи для понимания класса FileManager, целесообразно разделить многообразные методы этого класса на две большие категории в соответствии с их ожидаемым использованием - первичный интерфейс и виртуальные методы. Такое структурирование отражает то, что мы понимаем под типичным использованием методов FileManager.

Методы первичного интерфейса

Методы первичного интерфейса, которые вы, скорее всего, явно вызываете в своих программах, могут, в свою очередь, быть разделены на три категории:

Разового применения:

Init	инициализирует объект FileManager
Kill	завершает объект FileManager

Основного применения:

Open[V]	открывает файл
TryOpen	открывает файл
Next	читает следующую запись при последовательной обработке
TryNext	читает следующую запись при последовательной обработке
Previous	читает предыдущую запись при последовательной обработке
TryPrevious	читает предыдущую запись при последовательной обработке
Fetch	читает конкретную запись по ключу
TryFetch	читает конкретную запись по ключу
Position	возвращает уникальную позицию текущей записи
TryReget	читает конкретную запись в указанной позиции
PrimeAutoInc[V]	автоматически инкрементирует поля записи перед добавлением
Insert	добавляет новую запись
TryInsert	добавляет новую запись
CancelAutoInc[V]	восстанавливает файл в состояние перед вызовом метода PrimeAutoInc
Update	изменяет текущую запись
TryUpdate	изменяет текущую запись
Close[V]	закрывает файл

[V] Эти методы также являются виртуальными.

Эпизодического применения:

ClearKey	очищает диапазон ключевых полей
SetKey	устанавливает определенный ключ в качестве текущего для других методов
KeyToOrder	возвращает выражение ORDER,

	эквивалентное указанному ключу
GetComponents	возвращает количество компонентов ключа
GetField	возвращает ссылку на ключевое поле
GetFieldName	возвращает название ключевого поля
GetEOF	возвращает текущее состояние конца файла
GetError	возвращает текущий код ошибки
SetError	сохраняет текущее состояние ошибки
GetName	возвращает имя файла
SetName	устанавливает имя файла
SaveBuffer	сохраняет текущий буфер записи
RestoreBuffer	восстанавливает ранее сохраненный буфер записи
SaveFile	сохраняет текущее состояние файла
RestoreFile	восстанавливает ранее созданенное состояние файла
UseFile	открывает LazyOpen файл
AddKey	описывает программные ключи

Виртуальные методы

Обычно эти методы не вызываются непосредственно. Однако мы предполагаем, что у вас возникнет необходимость переопределить эти методы, и, поскольку они являются виртуальными, сделать это несложно. В противном случае, и их стандартное поведение представляется разумным.

Open	открывает файл
BindFields	выполняет операцию BIND для всех полей файла
PrimeAutoInc	автоматически инкрементирует поля записи перед добавлением
TryPrimeAutoInc	автоматически инкрементирует поля записи перед добавлением
CancelAutoInc	восстанавливает файл в состояние перед вызовом метода PrimeAutoInc
EqualBuffer	определяет изменения буфера записи
PrimeRecord	инициализирует запись перед добавлением
Throw	обрабатывает ошибку
ThrowMessage	устанавливает собственное сообщение об ошибке, после чего обрабатывает ошибку
ValidateField	проверяет определенное поле записи
ValidateFields	проверяет диапазон полей записи
ValidateRecord	проверяет все поля записи
Close	закрывает файл

AddKey (устанавливает ключи файла)

AddKey (*ключ, описание* [,*автоинремент*] **)**

AddKey	Описыввает ключ (KEY) или статический индекс (INDEX) обрабатываемого файла.
ключ	Метка ключа или статического ииндекса.
описание	Символьная константа, переменная, EQUATE или выражение, описывающая ключ.
автоинремент	Целочисленная константа, переменная, EQUATE или выражение, указывающая, должен ли FileManager автоматически генерировать инкрементирующиеся числовые значения для данного ключа при добавлении новой записи. Значение, равное единице (1,) вызвает автоматическое инкрементирование ключа, нулевое (0) значение - отключает автоинкремент. Если параметр опущен, его значение принимается равным нулю.

Метод **AddKey** описывает ключ или статический индекс обрабатываемого файла для того, чтобы с ним могли работать другие методы FileManager. Обычно вам следует вызывать AddKey после вызова метода Init (или в порожденном методе Init).

Реализация: Во время выполнения программы *описание* используется в некоторых сообщениях об ошибках, имеющих отношение к ключу.

Пример:

```
Access:Client.Init  PROCEDURE
CODE
PARENT.Init(Client, GlobalErrors)        !вызвать метод Init базового класса
SELF.FileNameValue = 'Client'            !установить имя файла
SELF.Buffer &= CLI:Record                !создать ссылку на буфер записи
SELF.AddKey(CLI:IDKey,'Client ID',1)     !описать первичный ключ
SELF.AddKey(CLI:NameKey,'Client Name')
                                         !описать другой ключ
```

См. Также: Init

BindFields (выполняет операцию BIND для полей после открытия файла)

BindFields, VIRTUAL

Метод **BindFields** выполняет операцию BIND для полей после открытия файла. Более подробно о BIND см. *Описание языка*.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод Open вызвает метод BindFields.

Пример:

```
PROGRAM
            INCLUDE('ABFILE.INC')       !объявить класс FileManager
            MAP
            END

GlobalErrors ErrorClass                 !объявить объект GlobalErrors
Access:Client CLASS(FileManager)        !породить объект Access:Client
BindFields ROCEDURE,VIRTUAL             !подготовить поля для
                                        !динамического использования
END
Client
FILE,DRIVER('TOPSPEED'),PRE(CLI),CREATE,BINDABLE,THREAD
IDKey
KEY(CLI:ID),NOCASE,OPT,PRIMARY
Record      RECORD,PRE()
ID          LONG
Name        STRING(20)
StateCode   STRING(2)
            END
            END
            CODE                        !Код программы

Access:Client.BindFields PROCEDURE !вызывается методом Open базового класса
            CODE
            BIND(CLI:RECORD)            !связать все поля для динамического
                                        !использования.
```

См. Также: Open

CancelAutoInc (отменяет PrimeAutoInc)

CancelAutoInc([*диспетчер связей*] **), VIRTUAL, PROC**

CancelAutoInc	Отменяет PrimeAutoInc
диспетчер связей	Метка объекта класса RelationManager для обрабатываемого файла. Если параметр присутствует, отмена операции охватывает все связанные файлы. Если параметр опущен, отмена операции не распространяется на связанные файлы.

Метод **CancelAutoInc** восстанавливает обрабатываемый файл и, опционально, все связанные файлы в состояние, в котором они находились перед вызовом метода PrimeAutoInc. Обычно это происходит при отмене операции добавления. CancelAutoInc возвращает результат, указывающий, успешно ли завершилась операция. Нулевой результат (0 или Level:Benign) означает успешное завершение, любой другое значение свидетельствует о возникших проблемах.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод PrimeAutoInc при добавлении записей с автоинкрементными ключами добавляет пустую запись. CancelAutoInc удаляет эту пустую запись и, если присутствует параметр *диспетчер связей*, также удаляет всех потомков этой пустой записи.

Если CancelAutoInc завершается успешно, он возвращает Level:Benign (объявлен в файле ABERROR.INC). Если же в конечном счете завершение неудачно, возвращается уровень серьезности ошибки, возникшей при попытке восстановить файлы. Более подробно об уровнях серьезности см. *ErrorClass*.

Тип результата: BYTE

Пример:

```
PROGRAM
INCLUDE('ABFILE.INC')      !объявить класс FileManager
MAP
```

```
            END

GlobalErrors  ErrorClass                    !объявить объект GlobalErrors
Access:Client CLASS(FileManager)            !породить объект Access:Client
Init          PROCEDURE
CancelAutoInc PROCEDURE,VIRTUAL             !прототип CancelAutoInc
            END
Client
FILE,DRIVER('TOPSPEED'),PRE(CLI),CREATE,BINDABLE,THREAD
IDKey
KEY(CLI:ID),NOCASE,OPT,PRIMARY
Record      RECORD,PRE()
ID          LONG
Name        STRING(20)
StateCode STRING(2)
            END
            END
InsertWindow WINDOW('Add a new Client'),AT(,,159,73),IMM,|SYSTEM,GRAY
PROMPT('&Name:'),AT(8,20),USE(?CLI:Name:Prompt)
ENTRY(@s20),AT(61,20,84,10),USE(CLI:Name),|
             MSG('Client Name'),REQ
PROMPT('StateCode:'),AT(8,34),|
 USE(?CLI:StateCode:Prompt)
ENTRY(@s2),AT(61,34,40,10),USE(CLI:StateCode),|      MSG('State Code')
BUTTON('OK'),AT(12,53,45,14),USE(?OK),DEFAULT
BUTTON('Cancel'),AT(82,53,45,14),USE(?Cancel)
END
            CODE
            GlobalErrors.Init               !инициализировать объект GlobalErrors
Access:Client.Init                          !инициализировать объект Access:Client
Access:Client.Open                          !открыть файл Client
IF Access:Client.PrimeRecord()              !инициализировать запись Client
                                            !(автоинкремент)
            POST(Event:CloseWindow)         !если не вышло - завершить работу
            END

            OPEN(InsertWindow)

            ACCEPT
                                    CASE FIELD()
                                    OF ?OK
            IF EVENT() = Event:Accepted             !по кнопке OK
               IF Access:Client.Insert() = Level:Benign
```

```
                                        !новая запись в файл Client успешно
                                        !добавлена
            POST(Event:CloseWindow)     !закрыть окно
            ELSE                        !добавление потерпело неудачу
            SELECT(?CLI:Name:Prompt)    !выбрать поле имя клиента
            CYCLE                       !начать все сначала
            END
            END
            OF ?Cancel
            IF EVENT() = EVENT:Accepted !по кнопке Cancel
            Access:Client.CancelAutoInc !вернуть файл Client в исходное состояние
            POST(Event:CloseWindow)     !закрыть окно
            END
            EMD
            END

            Access:Client.Close         !закрыть файл Client
            Access:Client.Kill          !завершить объект Access:Client
            GlobalErrors.Kill           !завершить объект GlobalErrors
            RETURN
Access:Client.CancelAutoInc PROCEDURE!вернуть файл в исходное состояние
            CODE                        !здесь ваш код
            PARENT.CancelAutoInc        !вызвать метод базового класса
            !здесь ваш код
```

См. Также: PrimeAutoInc

ClearKey (очищает указанные ключевые поля)

ClearKey (*ключ* [, *первое поле*] [, *последнее поле*] [, *максимальные значения*] **)**

ClearKey	Очищает или (ре)инициализирует указанный диапазон ключевых полей.
ключ	Метка ключа.
последнее поле	Численная константа, переменная, EQUATE или выражение, указывающее первое очищаемое поле ключа. Если параметр опущен, его значение принимается равным единице (1)
lastcomponent	Численная константа, переменная, EQUATE или выражение, указывающая последнее очищаемое поле ключа. Если параметр опущен, его значение принимается равным двадцати двум (22).
максимальные значения	Целочисленная константа, переменная, EQUATE или выражение, указывающая, очищать ключевые поля в нулевые (или пробельные для строк) или в максимально возможные значения. Если параметр

равен единице (1), полям присваиваются максимально возможные значения. Если параметр равен нулю (0), строковые поля зачищаются пробелами, численные - нулями. Если параметр опущен, его значение принимается равным нулю (0).

Метод **ClearKey** очищает или (ре)инициализирует указанный диапазон ключевых полей.

Реализация: ClearKey полезен для ограничения диапазона старших ключевых полей при выборке записей по ключу. Присваивая старшим полям ключа определенные значения и зачищая младшие поля, вы можете прочесть первую (или последнюю) запись, у которой старшие поля ключа соответствуют присвоенным ранее значениям. Так, например, можно найти первый заказ (младшее поле ключа) клиента (старшее поле ключа).

Значения, которые присваивает метод ClearKey очищаемым полям, зависят от трех факторов: типа ключевого поля (численное или символьное), порядка сортировки по данному полю (восходящий или нисходящий), и значения, переданного в качестве параметра *максимальные значения* (True или False). В следующей таблице представлены значения, присваиваемые методом ClearKey для каждой комбинации трех вышеописанных факторов.

	Числовые поля		**Символьные поля**	
максимальные значения	**Восходящий**	**Нисходящий**	**Восходящий**	**Нисходящий**
True (1)	Макс. значения	нули	Макс. значения	пробелы
False (0)	нули	Макс. значения	пробелы	Макс. значения

Пример:

```
PROGRAM
        INCLUDE('ABFILE.INC')          !объявить класс FileManager
        MAP
        END
GlobalErrorsErrorClass                 !объявить объект Errors
Access:Order CLASS(FileManager)        !объявить объект Access:Order
END
Order
FILE,DRIVER('TOPSPEED'),PRE(ORD),CREATE,BINDABLE,THREAD
IDKey
KEY(Ord:Cust,Ord:ID,Ord:Date),NOCASE,OPT,PRIMARY
```

```
Record      RECORD,PRE()
Cust        LONG
ID          LONG
Date        LONG
            END
            END
            CODE                              !код программы
                                              !найти первый заказ клиента, очистив
                                              !компоненты ключа за   !исключением Ord:Cust
Access:Order.ClearKey(ORD:IDKey,2)  !очистить Ord:ID и Ord:Date
Access:Order.Fetch                  !прочесть следующую запись по ключу
```

Close (close the file)

Close, VIRTUAL, PROC

Метод **Close** сообщает менеджеру файла, что вызывающая процедура завершила работу с файлом, затем закрывает его, если нет других процедур, работающих с этим файлом. Метод Close обрабатывает все ошибки, которые могут возникнуть при закрытии файла.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод Close возвращает результат, равный Level:Benign (EQUATE, описанный в файле ABERROR.INC). Более подробно об уровнях серьезности ошибок см. *ErrorClass*.

Тип результата: BYTE

Пример:

```
PROGRAM
            INCLUDE('ABFILE.INC')         !объявить класс FileManager
            MAP
```

```
        END

GlobalErrors ErrorClass                         !объявить объект GlobalErrors
    Access:Client   CLASS(FileManager)          !породить объект Access:Client
    Init      PROCEDURE                         !прототип Access:File.Init
      END
Client FILE,DRIVER('TOPSPEED'),PRE(CLI),CREATE,BINDABLE,THREAD
                                                !объявление файла
      END
      CODE
      GlobalErrors.Init         !инициализировать объект GlobalErrors
      Access:Client.Init        !инициализировать объект Access:Client
      Access:Client.Open        !открыть файл Client
                                !код программы
      Access:Client.Close       !закрыть файл Client
      Access:Client.Kill        !завершить объект Access:Client
      GlobalErrors.Kill         !завершить объект GlobalErrors
```

EqualBuffer (определяет изменения буфера записи)

EqualBuffer(*идентификатор буфера*), **VIRTUAL**

EqualBuffer Сравнивает буфер записи обрабатываемого файла с указанным буфером и возвращает результат, показывающий, равны ли они.

идентификатор буфера
Целочисленная константа, переменная, EQUATE или выражение, указывающее, с каким буфером следует проводить сравнение - обычно значение, возвращенное методом SaveBuffer.

Метод **EqualBuffer** сравнивает буфер записи обрабатываемого файла, включая все MEMO-поля, с указанным буфером и возвращает результат, показывающий, равны ли они. Возвращаемое значение, равное единице (1 или True), свидетельствует о равенстве буферов. Если буферы не равны, возвращается нулевое (0 или False) значение.

Тип результата: BYTE

Пример:

```
MyWindowManager.TakeCloseEvent PROCEDURE
        CODE
IF SELF.Response = RequestCancelled     !конечный пользователь отказался от
                                        !редактирования записи
```

```
IF ~SELF.Primary.Me.EqualBuffer(SELF.Saved) !проверить, были ли изменения
                                            !и обработать их
        END
        END
```

См. Также: SaveBuffer

Fetch (читает определенную запись по ключу)

Fetch(*ключ*), PROC

Fetch Читает определенную запись по ключу и обрабатывает все ошибки.
ключ Метка ключа, по которому производится чтение.

Метод **Fetch** читает определенную запись по ключу и обрабатывает все ошибки. Вы должны присвоить значения соответствующих ключевых полей прежде, чем вызывать Fetch. Если ключ не уникальный, считывается первая запись с указанным значением ключа.

Метод TryFetch является альтернативным (с ручной обработкой ошибок) методом чтения определенных записей.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод Fetch пытается прочесть указанную запись. Если попытка завершается успешно, он возвращает Level:Benign (объявлен в файле ABERROR.INC). Если запись прочесть не удалось, возвращается Level:Notify (также объявлен в файле ABERROR.INC) и *чистит буфер записи.* Более подробно об уровнях серьезности ошибок см. *ErrorClass.*

Тип результата: BYTE

Пример:

```
PROGRAM
  INCLUDE('ABFILE.INC')                !объявить кклacc FileManager
  MAP

        END
GlobalErrors ErrorClass                !объявить объект GlobalErrors
Access:States CLASS(FileManager)       !объявить объект Access:States
      END
```

```
States
FILE,DRIVER('TOPSPEED'),PRE(ST),CREATE,BINDABLE,THREAD
StateCodeKey
KEY(ST:StateCode),NOCASE,OPT,PRIMARY
Record      RECORD,PRE()
StateCode  STRING(2)
State
STRING(20)
                                                                    .       .
 CODE                                    !код программы
                                         !прочесть запись, относящуюся к штату
Флорида
ST:StateCode = 'FL'                      !заполнить ключ соответствующими
                                         !значениями
Access:States.Fetch(ST:StateCodeKey) !прочесть запись и обработать все ошибки
```

См. Также: TryFetch

GetComponents (возвращает количество ключевых полей)

GetComponents(*ключ*)

GetComponents Возвращает количество полей в указанном ключе.
ключ Метка ключа.

Метод **GetComponents** возвращает количество полей в указанном ключе.

Тип результата: BYTE

Пример:

```
PROGRAM
            INCLUDE('ABFILE.INC')        !объявить класс FileManager
            MAP
            END
GlobalErrors ErrorClass                  !объявить объект GlobalErrors
Access:Order CLASS(FileManager)          !породить объект Access:Order
            END
I           BYTE
                                         !объявить файл заказов
Order       FILE,DRIVER('TOPSPEED'),PRE(ORD),THREAD
IDKey
KEY(Ord:Cust,Ord:ID,Ord:Date),NOCASE,OPT,PRIMARY
Record      RECORD,PRE()
Cust        LONG
ID          LONG
Date        LONG
```

```
KeyQueue QUEUE,PRE(KeyQ)          !список ключевых полей
Field    ANY
                                  !ссылка на ключевое поле
FieldName  STRING(12)             !имя ключевого поля
         END
         CODE                     !код программы
         LOOP Access:Order.GetComponents( ORD:IDKey ) TIMES
                                  !цикл по ключевым полям
         I += 1                   !увеличить счетчик
         KeyQ.Field = Access:Order.GetField(ORD:IDKey,I)
                                  !получить ссылку на ключевое поле
         KeyQ.FieldName = Access:Order.GetFieldName(ORD:IDKey,I)
                                  !определить его имя
         END
```

GetEOF (возвращает статус конца файла)

GetEOF

Метод **GetEOF** возвращает текущий статус конца обрабатываемого файла.

Реализация: GetEOF возвращает единицу (1), если при последовательной обработке был достигнут конец файла, в противном случае - возвращается ноль (0).

Тип результата: BYTE

Пример:

```
PROGRAM
         INCLUDE('ABFILE.INC')            !объявить объектFileManager
         MAP
         END
GlobalErrors ErrorClass                   !объявить объект GlobalErrors
Access:Client CLASS(FileManager)          !породить объект Access:Client
         END
         CODE                             !код программы
         LOOP UNTIL Access:Client.GetEOF() !цикл по файлу Client
         CASE Access:Client.Next()        !прочесть следующую запись
         OF Level:Notify
         OROF Level:Fatal                 !ошибка
         POST(Event:CloseWindow)          !завершить работу
         BREAK
         ELSE
```

```
                                    !в противном случае
        PRINT(Rpt:Detail)           !распечатать запись
        END
        END
```

См. Также: Next, TryNext, Previous, TryPrevious

GetError (возвращает текущий код ошибки)

GetError

Метод **GetError** возвращает текущий код ошибки для обрабатываемого файла. Более о кодах ошибок см. *ErrorClass*.

Тип результата: SIGNED

Пример:

```
PROGRAM
        INCLUDE('ABFILE.INC')       !объявить класс FileManager
        MAP
        LogError (STRING filename, SHORT error)
        END
GlobalErrors ErrorClass             !объявить объект GlobalErrors
Access:Client CLASS(FileManager)    !породить объект Access:Client
        END
                                    !объявить файл протокола
ErrorLog FILE(TopSpeed),PRE(LOG),CREATE,THREAD
Record  RECORD
Date    LONG
Time    LONG
File    STRING(20)
ErrorId SHORT
        CODE                        !код программы
        IF Access:Client.Open()     !ошибка
        LogError(Access:Client.GetName(),Access:Client.GetError())
                                    !протоколируем код ошибки и сообщение
        END                         !код программы
LogError PROCEDURE(STRING filename, SHORT error)
        CODE
        LOG:Date = TODAY()          !запомнить дату
    LOG:Time    = CLOCK()           !запомнить время
    LOG:File    = filename          !запомнить имя фала
        LOG:ErrorId = error
```

```
                                    !запомнить код ошибки
        ADD(ErrorLog)               !записать в протокол
```

GetField (возвращает ссылку на ключевое поле)

GetField(*ключ, компонент* **)**

GetField Возвращает ссылку на указанный компонент ключа.

ключ Метка ключа.

компонент Численная константа, переменная, EQUATE или выражение, задающая компонент ключа, ссылку на который нужно получить. Значение, равное единице (1), означает первый компонент, двум (2) - второй, и т.д.

Метод **GetField** возвращает ссылку на указанный компонент ключа.

Тип результата: *? (нетипизованный параметр-переменная)

Пример:

```
PROGRAM
        INCLUDE('ABFILE.INC')       !объявить класс FileManager
        MAP
        END
GlobalErrors ErrorClass             !объявить объект GlobalErrors
Access:Order CLASS(FileManager)     !породить объект Access:Order
        END
I       BYTE
                                    !объявить файл заказов
Order   FILE,DRIVER('TOPSPEED'),PRE(ORD),THREAD
IDKey   KEY(Ord:Cust,Ord:ID,Ord:Date),NOCASE,OPT,PRIMARY
Record  RECORD,PRE()
Cust    LONG
ID      LONG
Date    LONG
KeyQueue QUEUE,PRE(KeyQ)            !список ключевых полей
Field   ANY                         !ссылка на ключевое поле
FieldName STRING(12)                !имя ключевого поля
        END
        CODE                        !код программы
LOOP Access:Order.GetComponents( ORD:IDKey ) TIMES!цикл по ключевым полям
        I += 1                      !увеличить счетчик
        KeyQ.Field = Access:Order.GetField(ORD:IDKey,I)
                                    !получить ссылку на ключевое поле
        KeyQ.FieldName              = Access:Order.GetFieldName(ORD:IDKey,I)
                                     !определить его имя
        END
```

GetFieldName (возвращает имя ключевого поля)

GetFieldName(*ключ, компонент* **)**

GetFieldName Возвращает имя указанного ключевого поля.

ключ Метка ключа.

компонент Численная константа, переменная, EQUATE или выражение, задающая компонент ключа, ссылку на который нужно получить. Значение, равное единице (1), означает первый компонент, двум (2) - второй, и т.д.

Метод **GetFieldName** возвращает имя ключевого поля.

Тип результата: STRING

Пример:

```
PROGRAM
            INCLUDE('ABFILE.INC')          !объявить класс FileManager
            MAP
            END
GlobalErrors ErrorClass                    !объявить объект GlobalErrors
Access:Order CLASS(FileManager)            !породить объект Access:Order
            END
I           BYTE
                                           !объявить файл заказов
Order       FILE,DRIVER('TOPSPEED'),PRE(ORD),THREAD
IDKey       KEY(Ord:Cust,Ord:ID,Ord:Date),NOCASE,OPT,PRIMARY
Record      RECORD,PRE()
Cust        LONG
ID          LONG
Date        LONG
            .                              .
KeyQueue    QUEUE,PRE(KeyQ)                !список ключевых полей
Field       ANY
                                           !ссылка на ключевое поле
FieldName   STRING(12)                     !имя ключевого поля
            END
            CODE                           !код программы
            LOOP Access:Order.GetComponents( ORD:IDKey ) TIMES
                                           !цикл по ключевым полям
            I += 1                         !увеличить счетчик
            KeyQ.Field = Access:Order.GetField(ORD:IDKey,I)
```

```
                                        !получить ссылку на ключевое поле
KeyQ.FieldName = Access:Order.GetFieldName(ORD:IDKey,I)
                                        !определить его имя
        END
```

GetName (возвращает имя файла)

GetName

Метод **GetName** возвращает имя обрабатываемого файла, которое может быть использовано при выводе сообщений об ошибках и т.д.

Метод SetName устанавливает имя (если оно - переменная) обрабатываемого файла.

Реализация: GetName возвращает значение свойства FileNameValue, если оно имеет значение, в противном случае возвращается значение свойства FileName.

Тип результата: STRING

Пример:

```
PROGRAM
        INCLUDE('ABFILE.INC')           !объявить класс FileManager
        MAP
LogError (STRING filename, SHORT error)
        END
GlobalErrors ErrorClass                  !объявить объект GlobalErrors
Access:Client CLASS(FileManager)        !породить объект Access:Client
        END
                                        !объявить файл протокола
ErrorLog FILE(TopSpeed),PRE(LOG),CREATE,THREAD
Record  RECORD
Date    LONG
Time    LONG
File    STRING(20)
ErrorId SHORT
        CODE                            !код программы
        IF Access:Client.Open()         !ошибка
LogError(Access:Client.GetName(),Access:Client.GetError())
                                        !протоколируем код ошибки и сообщение
        END                             !код программы
        LogError PROCEDURE(STRING filename, SHORT error)
        CODE
        LOG:Date = TODAY()              !запомнить дату
```

```
        LOG:Time = CLOCK()          !запомнить время
        LOG:File = filename         !запомнить имя фала
        LOG:ErrorId = error         !запомнить код ошибки
        ADD(ErrorLog)
                                    !записать в протокол
```

См. Также: FileName, FileNameValue, SetName

Init (инициализирует объект FileManager)

Init(*файл, обработчик ошибок* **)**

Init Инициализирует объект FileManager.
файл Метка обрабатываемого файла.
обработчик ошибок
Метка объекта ErrorClass. Более подробно см. *ErrorClass.*

Метод **Init** инициализирует объект FileManager.

Реализация: Метод Init не инициализирует ряд специфических свойств файла (Buffer, FileName и FileNameValue). Вы должны явно инициализировать эти свойства после вызова метода Init (или в методе Init порожденного класса). См. *Концептуальный пример*.

Пример:

```
PROGRAM
        INCLUDE('ABFILE.INC')       !объявить класс FileManager
        MAP
                                    !program map
        END
GlobalErrors ErrorClass             !объявить объект GlobalErrors
Access:Client CLASS(FileManager)    !породить объект Access:Client
Init    PROCEDURE                   !инициализация объекта Access:Client
        END
Client  FILE,DRIVER('TOPSPEED'),PRE(CLI),CREATE,BINDABLE,THREAD
IDKey   KEY(CLI:ID),NOCASE,OPT,PRIMARY
Record  RECORD,PRE()
ID      LONG
Name    STRING(20)
StateCode STRING(2)
        END
        END
        CODE
        GlobalErrors.Init           !инициализировать GlobalErrors
```

```
        Access:Client.Init          !инициализировать Access:Client
                                    !код программы
        Access:Client.Kill          !завершить Access:Client
        GlobalErrors.Kill           !завершить the GlobalErrors
Access:Client.Init  PROCEDURE
        CODE
        PARENT.Init(Client, GlobalErrors)
                                    !вызвать метод Init базового класса
SELF.FileNameValue = 'Client'       !установить имя
        SELF.Buffer &= CLI:Record   !создать ссылку на буфер записи файла Client
        SELF.AddKey(CLI:IDKey,'Client ID',1)
                                    !описать первичный ключ
```

См. Также: Buffer, File, FileName, FileNameValue

Insert (добавляет новую запись)

Insert, PROC

Метод **Insert** добавляет новую запись в файл, гарантируя, что поля записи имеют корректные значения, а также выполнены все необходимые автоинкременты. Метод Insert обрабатывает все ошибки, возникающие при добавлении записи.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат

TryInsert является альтернативным методом для добавления записей. Его отличие состоит в том, что он требует ручной обработки ошибок.

Реализация: Если метод Insert завершается успешно, он возвращает Level:Benign (описан в файле ABERROR.INC). Если добавить запись не удалось, возвращается уровень серьезности последней ошибки, возникшей в процессе добавления записи. Более подробно об уровнях серьезности см. *ErrorClass*.

Тип результата: BYTE

Пример:

```
PROGRAM
        INCLUDE('ABFILE.INC')       !объявить класс FileManager
        MAP                         !program map
        END
GlobalErrors ErrorClass             !объявить объект GlobalErrors
```

```
Access:Client CLASS(FileManager)      !породить объект Access:Client
              END
InsertWindow WINDOW('Add a new Client'),AT(,,159,73),IMM,|
              SYSTEM,GRAY
              PROMPT('&Name:'),AT(8,20),USE(?CLI:Name:Prompt)
              ENTRY(@s20),AT(61,20,84,10),USE(CLI:Name),|
              MSG('Client Name'),REQ
              PROMPT('State Code:'),AT(8,34),|USE(?CLI:StateCode:Prompt)
              ENTRY(@s2),AT(61,34,40,10),USE(CLI:StateCode),|
              MSG('State Code')
              BUTTON('OK'),AT(12,53,45,14),USE(?OK),DEFAULT
               END
  CODE
                                        !код программы
              ACCEPT
              CASE FIELD()
              OF ?OK
              IF EVENT() = Event:Accepted    !по кнопке OK
              IF Access:Client.Insert() = Level:Benign
                                        !добавили новую запись в файл Client
              POST(Event:CloseWindow)   !завершим работу
              ELSE
                                        !ошибка при добавлении
Access:Client.CancelPrimeAutoInc        !восстановить файл
              CYCLE
                                        !и все сначала
                                        .       .
                                        !еще код
```

См. Также: TryInsert, PrimeRecord

KeyToOrder (возвращает выражение ORDER для ключа)

KeyToOrder(*ключ, компонент* **)**

KeyToOrder Возвращает выражение-список для атрибута ORDER (для структуры VIEW), отражающее компоненты указанного ключа.

ключ Метка ключа.

компонент Численная константа, переменная, EQUATE или выражение, задающая компонент ключа, ссылку на который нужно получить. Значение, равное единице (1), означает первый компонент, двум (2) - второй, и т.д.

Метод **KeyToOrder в**озвращает выражение-список для атрибута ORDER (для структуры VIEW), отражающее компоненты указанного ключа. Выражение-

список включает указанный компонент и все последующие.

Более подробно об ORDER см. *Описание языка.*

Реализация: По умолчанию *компонент* равен единице (1). Максимальная длина возвращаемого выражения - 512 символов.

Тип результата: STRING

Пример:

```
PROGRAM
            INCLUDE('ABFILE.INC')       !объявить класс FileManager
            MAP
                                        !program map
            END
GlobalErrors ErrorClass                 !объявить объект GlobalErrors
Access:Client CLASS(FileManager)        !породить объект Access:Client
            END                         !declare order file
Order       FILE,DRIVER('TOPSPEED'),PRE(ORD),THREAD
IDKey       KEY(ORD:Cust,ORD:ID,ORD:Date),NOCASE,OPT,PRIMARY
Record      RECORD,PRE()
Cust        LONG
ID          LONG
Date        LONG
            .                           .
ClientView  VIEW(Order)                 !объявить VIEW по файлу Order
            PROJECT(ORD:Cust,ORD:ID,ORD:Date)
            END
            CODE                        !код программы
            ClientView{PROP:Order}=Access:Order.KeyToOrder(ORD:IDKey,2)
                                        !установить динамический порядок промотра
            !ClientView{PROP:Order}='ORD:ID,ORD:Date'
                                        !эквивалентный этому
            OPEN(ClientView)
            SET(ClientView)
```

Kill (завершение)

Kill

Метод **Kill** освобождает всю память, выделенную за время жизни объекта и выполняет любой другой, необходимый при завершении код.

```
Пример:
PROGRAM
            INCLUDE('ABFILE.INC')      !объявить класс FileManager
            MAP
            END
GlobalErrors ErrorClass                !объявить объект GlobalErrors
Access:Client CLASS(FileManager)       !породить объект Access:Client
            END

Client FILE,DRIVER('TOPSPEED'),PRE(CLI),CREATE,BINDABLE,THREAD
IDKey KEY(CLI:ID),NOCASE,OPT,PRIMARY
Record RECORD,PRE()
ID          LONG
Name        STRING(20)
StateCode STRING(2)
            END
            END
            CODE
            GlobalErrors.Init          !инициализировать GlobalErrors
            Access:Client.Init         !инициализировать Access:Client
            !код программы
            Access:Client.Kill         !завершить Access:Client
            GlobalErrors.Kill          !завершить GlobalErrors
```

Next (читает следующую запись при последовательной обработке)

Next, PROC

Метод **Next** читает следующую запись при последовательной обработке. Метод Next обрабатывает все ошибки (кроме достижения конца файла), возникающие при чиении записи.

TryNext является альтернативным методом, читающим следующую запись при последовательной обработке. Отличие состоит в необходимости ручной обработки ошибок.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Если метод Next успешно считывает запись, он возвращает Level:Benign (определен в ABERROR.INC). В противном случае возвращается уровень серьезности последней ошибки, возникшей в процессе чтения записи. Более подробно об уровнях серьезности ошибок см. *ErrorClass*.

Тип результата: BYTE

Пример:

```
PROGRAM
        INCLUDE('ABFILE.INC')       !объявить объектFileManager
        MAP
        END
GlobalErrors ErrorClass             !объявить объект GlobalErrors
Access:Client CLASS(FileManager)    !породить объект Access:Client
        END
        CODE
                                    !код программы
LOOP UNTIL Access:Client.GetEOF()   !цикл по файлу Client
CASE Access:Client.Next()           !прочесть следующую запись
OF Level:Notify OROF Level:Fatal    !ошибка
        POST(Event:CloseWindow)     !завершить работу
        BREAK
        ELSE                        !в противном случае
        PRINT(Rpt:Detail)           !распечатать запись
        END
        END
```

См. Также: TryNext

Open (открывает файл)

Open, VIRTUAL, PROC

Метод **Open** сигнализирует диспетчеру файлов, что вызывающая процедура использует файл, затем открывает его, если тот еще не открыт. Метод Open обрабатывает все ошибки, возникающие при открытии файла, в том числе, - при необходимости - создает файл и перестраивает ключи.

TryOpen является альтернативным методом, читающим следующую запись при последовательной обработке. Отличие состоит в необходимости ручной обработки ошибок.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Если файл отсутствует, и свойство Create имеет ненулевое значение, метод Open пытается создать файл. Если метод Open завершается успешно, он возвращает Level:Benign (объявлено в файле ABERROR.INC). В противном случае возвращается уровень серьезности последней произошедшей ошибки. Более подробно об уровнях серьезности см. *ErrorClass*.

Тип результата: BYTE

Пример:

```
PROGRAM
            INCLUDE('ABFILE.INC')          !объявить класс FileManager
            MAP
            END
GlobalErrors ErrorClass                    !объявить объект GlobalErrors
Access:Client CLASS(FileManager)           !породить объект Access:Client
            END
Client FILE,DRIVER('TOPSPEED'),PRE(CLI),CREATE,BINDABLE,THREAD
IDKey KEY(CLI:ID),NOCASE,OPT,PRIMARY
Record RECORD,PRE()
ID          LONG
Name        STRING(20)
StateCode STRING(2)
            END
            END
            CODE
            GlobalErrors.Init              !инициализировать GlobalErrors
            Access:Client.Init             !инициализировать Access:Client
            Access:Client.Open             !открыть файл Client
                                           !код программы
            Access:Client.Close            !закрыть файл
            Access:Client.Kill             !завершить Access:Client
            GlobalErrors.Kill              !завершить GlobalErrors
```

См. Также: Create, TryOpen

Position (возвращает позицию текущей записи)

Position

Метод **Position** возвращает уникальную позицию текущей записи.

Метод TryReget читает запись на основании значения, возвращенного методом Position.

Реализация: Метод Position возвращает результат выполнения функции POSITION по первичному ключу, если таковой имеется, в противном случае - по файлу. Более подробно о функции POSITION см. *Описание языка*.

Тип результата: STRING

Пример:

```
Hold = SELF.Position()
PUT( SELF.File )
CASE ERRORCODE()
OF NoError
OF RecordChangedErr

SELF.SetError(Msg:ConcurrencyFailedFromForm)
                                    SELF.Throw
                                    WATCH( SELF.File )
                                    SELF.TryReget(Hold)
ELSE
                                    SELF.SetError(Msg:PutFailed)
                                    RETURN SELF.Throw()
END
```

См. Также: TryReget

Previous (читает предыдущую запись при последовательной обработке)

Previous, PROC

Метод **Previous** читает следующую запись при последовательной обработке. Метод Previous обрабатывает все ошибки (кроме достижения конца файла), возникающие при чиении записи.

Try Previous является альтернативным методом, читающим следующую запись при последовательной обработке. Отличие состоит в необходимости ручной обработки ошибок.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Если метод Previous успешно считывает запись, он возвращает Level:Benign (определен в ABERROR.INC). В противном случае возвращается уровень серьезности последней ошибки, возникшей в процессе чтения записи. Более подробно об уровнях серьезности ошибок см. *ErrorClass*.

Тип результата: BYTE

Пример:

```
PROGRAM
        INCLUDE('ABFILE.INC')          !объявить объектFileManager
        MAP
        END
GlobalErrors ErrorClass                !объявить объект GlobalErrors
Access:Client CLASS(FileManager)       !породить объект Access:Client
        END
        CODE
                                       !program code
        LOOP                           !цикл по файлу Client
        CASE Access:Client.Next()      !прочесть ппредыдущую запись
        OF Level:Notify OROF Level:Fatal       !ошибка
        POST(Event:CloseWindow)        !завершить работу
        BREAK
        ELSE                           !в противном случае
        PRINT(Rpt:Detail)              !распечатать запись
        END
        END
```

См. Также: TryPrevious

PrimeAutoInc (выполняет автоинкрементирование полей при добавлении записи)

PrimeAutoInc, VIRTUAL, PROC

Метод **PrimeAutoInc** выполняет автоинкрементирование полей обрабатываемого файла при добавлении записи и обрабатывает все возникающие при этом ошибки. Если вы хотите создать форму для редактирования записи, на которой показывается автоинкрементируемый код записи, или необходимо поддерживать ссылочную целостность дочерних записей, вам следует для выполения автоинкремента использовать метод PrimeAutoInc.

TryPrimeAutoInc является альтернативным методом, выполняющим автоинкрементирование полей. Отличие состоит в необходимости ручной обработки ошибок.

Метод CancelAutoInc восстанавливает обрабатываемый файл в состояние, предшествующее вызову метода PrimeAutoInc.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод PrimeRecord вызывает метод PrimeAutoInc, если файл имеет автоинрементируемые ключи.

Если метод PrimeAutoInc завершается успешно, возвращается Level:Benign (объявлен в файле ABERROR.INC). В противном случае возвращается уровень серьезности последней ошибки, возникшей в процессе выполнения подготовки записи. Более подробно об уровнях серьезности ошибок см. *ErrorClass*.

Тип результата: BYTE

Пример:

```
PROGRAM
            INCLUDE('ABFILE.INC')          !объявить объектFileManager
            MAP
            END
GlobalErrors ErrorClass                    !объявить объект GlobalErrors
Access:Client CLASS(FileManager)           !породить объект Access:Client
Init        PROCEDURE                      !инициализация объекта Access:Client
PrimeAutoInc PROCEDURE,VIRTUAL             !подготовка новой записи при
                                           !добавлении
            END
Client      FILE,DRIVER('TOPSPEED'),PRE(CLI),CREATE,BINDABLE,THREAD
IDKey       KEY(CLI:ID),NOCASE,OPT,PRIMARY
Record      RECORD,PRE()
ID          LONG
Name        STRING(20)
StateCode   STRING(2)
            END
            END
InsertWindow WINDOW('Add a new Client'),AT(,,159,73),IMM,|
            SYSTEM,GRAY
            PROMPT('&Name:'),AT(8,20),USE(?CLI:Name:Prompt)
ENTRY(@s20),AT(61,20,84,10),USE(CLI:Name),|
 MSG('Client Name'),REQ PROMPT('State Code:'),AT(8,34),|
```

```
  USE(?CLI:StateCode:Prompt)
ENTRY(@s2),AT(61,34,40,10),USE(CLI:StateCode),|
  MSG('State Code')
BUTTON('OK'),AT(12,53,45,14),USE(?OK),DEFAULT
BUTTON('Cancel'),AT(82,53,45,14),USE(?Cancel)
ND
            CODE
            GlobalErrors.Init          !инициализировать объект GlobalErrors
            Access:Client.Init         !инициализировать объект Access:Client
            Access:Client.Open         !открыть файл Client
            IF Access:Client.PrimeAutoInc()          !подготовить запись
            POST(Event:CloseWindow)                  !если ошибка - закрыть окно
            END
            OPEN(InsertWindow)
            ACCEPT
            CASE FIELD()
            OF ?OK
            IF EVENT() = Event:Accepted              !по кнопке OK
            IF Access:Client.Insert() = Level:Benign !добавление завершено
            POST(Event:CloseWindow)                  !успешно - закрыть окно
            ELSE                                     в противном случае
            SELECT(?CLI:Name:Prompt)          !выбрать соответствующее поле
            CYCLE                             !и начать все сначала
            END
            END
            OF ?Cancel
            IF EVENT() = EVENT:Accepted       !по кнопке Cancel
            Access:Client.CancelAutoInc       !восстановить первоначальное
                                              !состояние файла Client
            POST(Event:CloseWindow)           !закрыть окно
            END
            EMD
            END

            Access:Client.Close               !закрыть файл Client
            Access:Client.Kill                !завершить объект Access:Client
            GlobalErrors.Kill                 !завершить объект GlobalErrors
            RETURN
Access:Client.PrimeAutoInc PROCEDURE
            CODE                              !ваш код
            PARENT.PrimeAutoInc               !вызов метода базового класса
                                              !ваш код
```

См. Также: CancelAutoInc, PrimeRecord, TryPrimeAutoInc

PrimeRecord (подготавливает запись при добавлении)

PrimeRecord(*[не очищать]*), VIRTUAL, PROC

PrimeRecord Подготавливает запись при добавлении в обрабатываемый файл.

не очищать Целочисленная константа, переменная, EQUATE или выражение, указывающее, следует ли очищать буфер записи. При нулевом значении (0 или False) запись очищается, при единичном (1 или True) - нет. Если параметр опущен, его значение принимается равным нулю(0).

Метод **PrimeRecord** подготавливает запись при добавлении в обрабатываемый файл.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод PrimeRecord подготавливает запись, опционально очищая ее буфер, затем присваивает полям требуемые значения, выполняя в том числе автоинкремент ключевых полей. Если метод завершается успешно, воозвращается Level:Benign (объявлен в файле ABERROR.INC). В противном случае возвращается уровень серьезности последней возникшей ошибки. Более подробно об уровнях серьезности ошибок см. *ErrorClass*.

Параметр *не очищат*ь позволяет вам очистить значения полей, или оставить их текущие значения.

Тип результата: BYTE

Пример:

```
 PROGRAM
          INCLUDE('ABFILE.INC')          !объявить объектFileManager
          MAP
          END
GlobalErrors  ErrorClass                 !объявить      объект GlobalErrors
Access:Client CLASS(FileManager)         !породить объект Access:Client
```

```
Init          PROCEDURE                !инициализация объекта Access:Client
PrimeAutoInc PROCEDURE,VIRTUAL         !подготовка новой записи при
                                       !добавлении
              END
Client
FILE,DRIVER('TOPSPEED'),PRE(CLI),CREATE,BINDABLE,THREAD
IDKey KEY(CLI:ID),NOCASE,OPT,PRIMARY
Record        RECORD,PRE()
ID            LONG
Name          STRING(20)
StateCode STRING(2)
              END
              END
InsertWindow WINDOW('Add a new Client'),AT(,,159,73),IMM,| SYSTEM,GRAY
              PROMPT('&Name:'),AT(8,20),USE(?CLI:Name:Prompt)
              ENTRY(@s20),AT(61,20,84,10),USE(CLI:Name),|
 MSG('Client Name'),REQ
PROMPT('State Code:'),AT(8,34),| USE(?CLI:StateCode:Prompt)
              ENTRY(@s2),AT(61,34,40,10),USE(CLI:StateCode),|
                                          MSG('State Code')
              BUTTON('OK'),AT(12,53,45,14),USE(?OK),DEFAULT
              BUTTON('Cancel'),AT(82,53,45,14),USE(?Cancel)
              END
              GlobalErrors.Init            !инициализировать объект GlobalErrors
              Access:Client.Init           !инициализировать объект Access:Client
              Access:Client.Open           !открыть файл Client
              IF Access:Client.PrimeRecord()       !подготовить запись
              POST(Event:CloseWindow)              !если ошибка - закрыть окно
              END
              OPEN(InsertWindow)
              ACCEPT  CASE FIELD()
              OF ?OK
              IF EVENT() = Event:Accepted          !по кнопке OK
              IF Access:Client.Insert() = Level:Benign
                                                   !добавление завершено
              POST(Event:CloseWindow)              !успешно - закрыть окно
              ELSE                                 !в противном случае
              SELECT(?CLI:Name:Prompt)             !выбрать соответствующее поле
              CYCLE
                                                   !и начать все сначала
              END
              END
              OF ?Cancel
```

```
            IF EVENT() = EVENT:Accepted          !по кнопке Cancel
            Access:Client.CancelAutoInc          !восстановить первоначальное
                                                 !состояние файла Client
            POST(Event:CloseWindow)              !закрыть окно
            END
            EMD
            END
            Access:Client.Close                  !закрыть файл Client
            Access:Client.Kill                   !завершить объект Access:Client
            GlobalErrors.Kill                    !завершить объект GlobalErrors
            RETURN
Access:Client.PrimeAutoInc PROCEDURE
            CODE                                 !ваш код
            PARENT.PrimeAutoInc                  !вызов метода базового класса
                                                 !ваш код
```

См. Также: PrimeAutoInc, CancelAutoInc

RestoreBuffer (восстанавливает ранее сохраненный буфер записи)

RestoreBuffer(*буфер* [, *восстановить*] **)**

RestoreBuffer	Восстанавливает содержимое ранее сохраненного буфера записи.
буфер	Целочисленная константа, переменная, EQUATE или выражение, идентифицирующее ранее сохраненный буфер, содержимое которого должно быть восстановлено. Это значение возвращается методом SaveBuffer.
восстановить	Целочисленная константа, переменная, EQUATE или выражение, указывающее, следует ли восстановить содержимое указанного сохраненного буфера в буфер записи обрабатываемого файла, или просто ликвидировать сохраненный буфер (операция DISPOSE). Если значение параметра равно единице (1 или True), содержимое буфера восстанавливается, при нулевом значении (0 или False) буфер записи файла не изменяется. Если параметр опущен, его значение принимается равным единице (True).

Метод **RestoreBuffer** восстанавливает содержимое буфера, сохраненного ранее при помощи метода SaveBuffer (включая все MEMO-поля), в буфер записи обрабатываемого файла.

Реализация: RestoreBuffer освобождает память, выделенную методом SaveBuffer. Таким образом, во избежание потерь памяти, каждый вызов метода SaveBuffer должен сопровождаться парным вызовом метода RestoreBuffer.

Метод RestoreBuffer читает в свойстве Buffers указанный буфер, после чего ликвидирует прочитанный буфер (операция DISPOSE).

Тип результата:

Пример:

```
FileManager.RestoreFile PROCEDURE(*USHORT Id)
 CODE
          IF ~SELF.UseFile()
          SELF.Saved.Id = Id
          GET(SELF.Saved,SELF.Saved.Id)
          ASSERT(~ERRORCODE())
          IF SELF.Saved.Key &= NULL
          RESET(SELF.File,SELF.Saved.Pos)
          ELSE
          RESET(SELF.Saved.Key,SELF.Saved.Pos)
          END
          IF SELF.Saved.WHeld
          HOLD(SELF.File)
          END
          IF SELF.Saved.WWatch
          WATCH(SELF.File)
          END
          NEXT(SELF.File)
          SELF.RestoreBuffer(SELF.Saved.Buffer)
          DELETE(SELF.Saved)
          Id = 0
          END
```

См. Также: Buffer, Buffers, SaveBuffer

RestoreFile (восстанавливает ранее сохраненное состояние файла)

RestoreFile(*состояние файла* **)**

RestoreFile	Восстанавливает ранее сохраненное состояние файла.
состояние файла	Значение типа USHORT, возвращеннео методом SaveFile, которое идентифицирует состояние, которое требуется восстановить.

Метод **RestoreFile** восстанавливает указанное состояние обрабатываемого файла. Восстанавливаются состояния, ранее сохраненные при помощи метода SaveFile.

Реализация: Метод RestoreFile восстанавливает позицию файла, а также активный статус HOLD или WATCH. RestoreFile вызвает метод RestoreBuffer для восстановления содержимого буфера записи.

Пример:

```
SaveState  USHORT                                !должно быть USHORT
           CODE
           SaveState = Access:MyFile.SaveFile()  !сохранить состояние файла
           SET(MyKey,MyKey)                      !доступ к файлу (меняет состояние
                                                 !файла)
           LOOP UNTIL Access:MyFile.Next()
                                                 !Провверка границ диапазона
                                                 !Обработка зааписи
           END
           Access:MyFile.RestoreFile(SaveState)
                                                 !восстановить ранее сохраненное
                                                 !состояние файла
```

См. Также:: SaveFile, RestoreBuffer

SaveBuffer (сохраняет копию буфера записи)

SaveBuffer

Метод **SaveBuffer** сохраняет копию буфера записи обрабатываемого файла (свойства Buffer) и возвращает уникальный идентификатор сохраненной записи. Сохраненный буфер затем восстанавливается при помощи метода RestoreBuffer.

Реализация: SaveBuffer наряду с другими полями сохраняет и MEMO-поля.

SaveBuffer выделяет память, которая освобождается методом RestoreBuffer. Таким образом, во избежание потерь памяти, каждый вызов метода SaveBuffer должен сопровождаться парным вызовом метода RestoreBuffer.

Тип результата: USHORT

Пример:

```
FileManager.SaveFile PROCEDURE
Id LONG,AUTO
I  SHORT,AUTO
 CODE
   Id = RECORDS(SELF.Saved)
   IF Id
    GET(SELF.Saved,Id)
    ASSERT(~ERRORCODE())
    Id = SELF.Saved.Id + 1
   ELSE
    Id = 1
```

```
END
SELF.Saved.Id = Id
SELF.Saved.Buffer = SELF.SaveBuffer()
SELF.Saved.Key &= SELF.File{PROP:CurrentKey}
SELF.Saved.WHeld = SELF.File{PROP:Held}
SELF.Saved.WWatch = SELF.File{PROP:Watched}
IF SELF.Saved.Key &= NULL
 SELF.Saved.Pos = POSITION(SELF.File)
ELSE
 SELF.Saved.Pos = POSITION(SELF.Saved.Key)
END
ADD(SELF.Saved)
RETURN Id
```

См. Также: Buffer, Buffers, RestoreBuffer

SaveFile (сохраняет текущее состояние файла)

SaveFile

Метод **SaveFile** сохраняет текущее состояние обрабатываемого файла и возвращает уникальный идентификатор сохраненного состояния. Это состояние затем восстанавливается при помощи метода RestoreFile.

Реализация: Метод SaveFile сохраняет текущую позицию файла, а также активные статусы HOLD и WATCH. SaveFile вызывает метод SaveBuffer для сохранения копии буфера записи обрабатываемого файла.

Тип результата: USHORT

Пример:

```
SaveState  USHORT                              !должно быть USHORT
           CODE
           SaveState = Access:MyFile.SaveFile()  !сохранить состояние файла
           SET(MyKey,MyKey)                    !доступ к файлу (меняет состояние
                                               !файла)
           LOOP UNTIL Access:MyFile.Next()
                                               !Провверка границ диапазона
                                               !Обработка зааписи
           END
           Access:MyFile.RestoreFile(SaveState)
                                               !восстановить ранее сохраненное
                                               !состояние файла
```

См. Также: RestoreFile, SaveBuffer

SetError (сохраняет указанную ошибку и ее состояние)

SetError(*код ошибки* **)**

SetError	Сохраняет указанную ошибку и ее состояние для использования методами Throw и т. д.
код ошибки	Численная константа, переменная, EQUATE или выражение, указывающее ошибку. Более подробно о кодах ошибок см. *ErrorClass*.

Метод **SetError** сохраняет указанную ошибку и ее состояние для использования методами Throw и т. д.

Пример:

```
Access:Client.Next FUNCTION(BYTE HandleError)          !функция Next
        CODE                                  !с альтернативной обработкой ошибок
        LOOP
        NEXT( SELF.File )                     !прочесть следующую запись
        CASE ERRORCODE()                      !обработка ошибок
        OF BadRecErr OROF NoError
        RETURN Level:Benign
        OF IsHeldErr                           !запись захвачена другой станцией
        SELF.SetError(Msg:RecordHeld)
                                              !сделать RecordHeld текущей ошибкой
        IF HandleError                        !если интерактивная обработка ошибок
        RETURN SELF.Throw()                   !передать текущую ошибку обработчику
        ELSE                                  !иначе (“тихая” обработка ошибок)
        RETURN Level:Notify                   !вернуть код ошибки вызывающему
        END
        END
        END
```

См. Также: Throw

SetKey (устанавливает текущий ключ)

SetKey(*ключ* **), PROTECTED**

SetKey	Делает указанный ключ текущим для использования другими методами класса FileManager.
ключ	Метка ключа.

Метод **SetKey** делает указанный ключ текущим для использования другими методами класса FileManager.

Этот метод имеет атрибут PROTECTED, следовательно, может быть вызван только из методов класса FileManager или порожденного от него класса.

Пример:

```
FileManager.GetComponents FUNCTION(KEY K)
                                        !возвращает число компонентов ключа
        CODE
        SELF.SetKey(K)                  !найти указанный ключ
        RETURN RECORDS( SELF.Keys.Fields )  !сосчитать компоненты
```

SetName (устанавливает текущее имя файла)

SetName(*имя файла* **)**

SetName	Задает значение переменной-имени файла.
имя файла	Строковая константа, переменная, EQUATE или выражение, содержащая имя файла для обрабатываемого файла.

Метод **SetName** задает значение переменной, указанной в атрибуте NAME обрабатываемого файла. Это значение определяет, какой файл в действительности будет открываться и обрабатываться объектом FileManager. Это имя файла также выводится в сообщениях об ошибках и т. д.

Метод GetName возвращает имя обрабатываемого файла.

Реализация: Метод SetName работает в предположении, что свойство FileName ссылается на переменную, указанную в атрибуте NAME обрабатываемого файла.

Пример:

```
PROGRAM
        INCLUDE('ABFILE.INC')           !объявить объектFileManager
        MAP
        END
GlobalErrors ErrorClass                 !объявить объект GlobalErrors
Access:Client CLASS(FileManager)        !породить объект Access:Client
Init    PROCEDURE                       !инициализация Access:File
        END
Client  FILE,DRIVER('TOPSPEED'),PRE(CLI),THREAD,NAME(CLientFile)
IDKey   KEY(CLI:ID),NOCASE,OPT,PRIMARY
    Record      RECORD,PE()
ID       LONG
Name     STRING(20)
StateCode STRING(2)
        CODE
        GlobalErrors.Init               !инициализировать объект
GlobalErrors Access:Client.Init         !инициализировать объект
Access:Client
```

```
        LOOP I# = 1 TO 12                   !цикл по 12 месячным файлам
        Access:Client.SetName('Client'&I#)  !установить имя файла
        Access:Client.Open                  !открыть месячный файл
                                            !обработка файла
        Access:Client.Close                 !закрыть месячный файл
        END
Access:Client.Init  PROCEDURE
        CODE
        PARENT.Init(GlobalErrors)           !вызов метода базового класса
        SELF.File &= Client                 !создать ссылку на файл
        SELF.Buffer &= CLI:Record           !создать ссылку на буфер записи
        SELF.FileName &= ClientFile
                                            !создать ссылку на переменную-имя
                                            !файла
```

См. Также: FileName, FileNameValue, GetName

Throw (передает ошибку обработчику)

Throw([*код ошибки*] **), VIRTUAL, PROC**

Throw	Передает ошибку обработчику.
код ошибки	Численная константа, переменная, EQUATE или выражение, определяющая ошибку, подлежащую обработке. Если параметр опущен, Throw обрабатывает текущую ошибку, т. е. ошибку, идентифицированную предыдущим вызовом SetError или Throw.

Метод **Throw** передает текущую (последнюю встретившуюся) ошибку, включая значения FILEERROR() и FILEERRORCODE() назначенному обработчику ошибок, затем возвращает уровень серьезности ошибки.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод SetError сохраняет указанную ошибку и ее состояние для использования методом Throw. Более подробно о кодах ошибок и их уровнях серьезности см. *ErrorClass*.

Метод Init получает и устанавливает объект, обрабатывающий ошибки.

Тип результата: BYTE

Пример:

```
Access:Client.Next FUNCTION(BYTE HandleError) !функция Next
        CODE                                  !с альтернативной обработкой
                                              !ошибок
        LOOP
        NEXT( SELF.File )                     !прочесть следующую запись
        CASE ERRORCODE()                      !обработка ошибок
        OF BadRecErr OROF NoError
        RETURN Level:Benign
        OF IsHeldErr                          !запись захвачена другой станцией
        SELF.SetError(Msg:RecordHeld) !сделать RecordHeld текущей ошибкой
        IF HandleError                        !если интерактивная обработка ошибок
        RETURN SELF.Throw()                   !передать текущую ошибку обработчику
        ELSE                                  !иначе (“тихая” обработка ошибок)
        RETURN Level:Notify                   !вернуть код ошибки вызывающему
        END
        END
        END
```

См. Также: Init, SetError

ThrowMessage (передает ошибку и текст обработчику)

ThrowMessage(*код ошибки, текст*), VIRTUAL, PROC

ThrowMessage	Передает ошибку и текст обработчику.
код ошибки	Численная константа, переменная, EQUATE или выражение, определяющая ошибку, подлежащую обработке.
текст	Строковая константа, переменная, EQUATE или выражение, включаемая в сообщение об ошибке.

Метод **ThrowMessage** передает указанную ошибку, включая значения FILEERROR() and FILEERRORCODE(), и текст обработчику ошибок затем возвращает уровень серьезности ошибки. Более подробно о кодах ошибок и их уровнях серьезности см. *ErrorClass*.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод Init получает и устанавливает объект, обрабатывающий ошибки. Включение *текста* в сообщение об ошибке зависит от обработчика

ошибок. См. *ErrorClass*.

Тип результата: BYTE

Пример:

```
GlobalErrors ErrorClass
!объявить объект GlobalErrors
Access:Client CLASS(FileManager)      !породить объект Access:Client
ValidateField FUNCTION(UNSIGNED Id),BYTE,VIRTUAL   !прототип функции проверки
END
Client      FILE,DRIVER('TOPSPEED'),PRE(CLI),THREAD
IDKey       KEY(CLI:ID),NOCASE,OPT,PRIMARY
Record      RECORD,PRE()
ID          LONG
Name        STRING(20)
StateCode STRING(2)
            .
            CODE                              !код программы

Access:Client.ValidateField FUNCTION(UNSIGNED Id)
            CODE
            IF ID = 3                         !проверить код штата (3-е поле)
            IF ~CLI:StateCode                 !нет такого штата,
                                              !передать ошибку и текст обработчику
            RETURN Access:Client.ThrowMessage(Msg:RequiredField,'StateCode')
            .                                 .
            RETURN Level:Notify
```

См. Также: Init

TryFetch (пытается прочесть запись по ключу)

TryFetch(*ключ* **), PROC**

TryFetch Пытается прочесть запись по ключу.
ключ Метка ключа, по которому производится чтение.

Метод **TryFetch** читает определенную запись по ключу. Вы должны заполнить ключевые поля, прежде, чем вызывать TryFetch. Если ключ не уникальный, читается первая запись с указанным значением ключа.

Fetch является альтернативным методом для чтения записи по ключу. Отличие состоит в автоматической обработке ошибок.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: TryFetch делает попытку прочесть указанную запись. Если

операция завершается успешно, возвращается Level:Beningn. В противном случае возвращается Level:Notify, при этом буфер записи не очищается. Более подробно о Level:Benign и Level:Notify см. *ErrorClass*.

Тип результата: BYTE

Пример:

```
PROGRAM
        INCLUDE('ABFILE.INC')          !объявить объектFileManager
        MAP
        END
GlobalErrors  ErrorClass               !объявить объект GlobalErrors
Access:States CLASS(FileManager)       !объявить объект Access:States
        END
States
FILE,DRIVER('TOPSPEED'),PRE(ST),CREATE,BINDABLE,TREAD
StateCodeKey
KEY(ST:StateCode),NOCASE,OPT,PRIMARY
Record     RECORD,PRE()
StateCode STRING(2)
State
STRING(20)                                                    .        .
 CODE      !код программы
        !прочесть запись, касающуюся штата Флорида
        ST:StateCode = 'FL'                          !заполнить ключ
        IF Access:States.TryFetch(ST:StateCodeKey) !прочесть запись
        GlobalErrors.Throw(Msg:FieldNotInFile)
                                                !самостоятельно обработать ошибки
        END
```

См. Также: Fetch

TryInsert (пытается добавить новую запись)

TryInsert, PROC

Метод **TryInsert** добавляет новую запись, гарантируя ее корректность и автоинрементировав при необходимости ключевые поля. Метод TryInsert не пытается обрабатывать ошибки.

Insert является альтернативным методом добавления записей. Отличие состоит в автоматической обработке ошибок.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат

Реализация: Метод TryInsert пытается добавить запись. Если это удается, возвращается Level:Benign (объявлен в файле ABERROR.INC). В противном

случае возвращается уровень серьезности произошедшей ошибки. Более подробно об уровнях серьезности ошибок см. *ErrorClass*.

Тип результата: BYTE

Пример:

```
PROGRAM
        INCLUDE('ABFILE.INC')       !объявить объектFileManager
        MAP
        END
GlobalErrors ErrorClass             !объявить объект GlobalErrors
Access:Client CLASS(FileManager)    !породить объект Access:Client
        END
InsertWindow WINDOW('Add a new Client'),AT(,,159,73),|
        IMM,SYSTEM,GRAY
        ('&Name:'),AT(8,20),USE(?CLI:Name:Prompt)
        ENTRY(@s0),AT(61,20,84,10),USE(CLI:Name),|
        MSG('Client Name'),REQ
        PROMPT('State Code:'),AT(8,34),|
          USE(?CLI:StateCode:Prompt)
        ENTRY(@s2),AT(61,34,40,10),USE(CLI:StateCode),|
          MSG('State Code')
        BUTTON('OK'),AT(12,53,45,14),USE(?OK),DEFAULT
        END
        CODE                        !код программы
        ACCEPT
        CASE FIELD()
        OF ?OK
        IF EVENT() = Event:Accepted     !по кнопке OK
        IF Access:Client.TryInsert() = Level:Benign     !добавить новую запись
        POST(Event:CloseWindow)                         !если успешно - закрыть окно
        ELSE                                            !ошибка
        Access:Client.Throw(Msg:InsertFailed)           !обработать
        Access:Client.CancelPrimeAutoInc                !восстановить файл
        CYCLE

                                                        !все сначала
        . .
  !еще код
```

См. Также: Insert, PrimeRecord

TryNext (пытается прочесть следующую запись при последовательной обработке)

TryNext, PROC

Метод **TryNext** читает следующую запись при последовательной обработке. Метод TryNext не пытается обрабатывать ошибки, возникающие при чтении записи.

Next является альтернативным методом чтения следующей записи при последовательной обработке. Отличие состоит в автоматической обработке ошибок.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод TryNext пытается прочесть следующую запись. Если это удается, возвращается Level:Benign (объявлен в файле ABERROR.INC). В противном случае возвращается уровень серьезности произошедшей ошибки. Более подробно об уровнях серьезности ошибок см. *ErrorClass*.

Тип результата: BYTE

Пример:

```
PROGRAM
        INCLUDE('ABFILE.INC')              !объявить объектFileManager
        MAP
        END
GlobalErrors ErrorClass                    !объявить объект GlobalErrors
Access:Client CLASS(FileManager)           !породить объект Access:Client
        END
        CODE                               !код программы
        LOOP
                                           !цикл по файлу
        CASE Access:Cliet.TryNext()        !прочесть очередную запись
        OF Level:Notify OROF Level:Fatal   !ошибка
        POST(Event:CloseWindow)            !закрыть окно
        BREAK
        ELSE
                                   !иначе
        PRINT(Rpt:Detail)          !распечатать запись
                                   END
```

```
        END
```

См. Также: Next

TryOpen (пытается открыть файл)

TryOpen, PROC

Метод **TryOpen** сигнализирует диспетчеру файлов, что вызывающая процедура использует файл, после чего открывает файл, если тот еще не открыт. Метод TryOpen не пытается обрабатывать ошибки.

Open является альтернативным методом для открытия файлов. Отличие состоит в автоматической обработке ошибок.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: TryOpen пытается открыть файл. Если это удается, возвращается Level:Benign (объявлен в файле ABERROR.INC). В противном случае возвращается уровень серьезности произошедшей ошибки. Более подробно об уровнях серьезности ошибок см. *ErrorClass*.

Тип результата: BYTE

Пример:

```
PROGRAM
        INCLUDE('ABFILE.INC')                   !объявить объектFileManager
        MAP
        END
GlobalErrors ErrorClass                         !объявить объект GlobalErrors
Access:Client CLASS(FileManager)                !породить объект Access:Client
Init    PROCEDURE                               !прототип метода инициализации
        END
Client  FILE,DRIVER('TOPSPEED'),PRE(CLI),CREATE,BINDABLE,THREAD
                                                !объявление файла
        END
        CODE
        GlobalErrors.Init               !инициализировать объект GlobalErrors
        Access:Client.Init              !инициализировать объект Access:Client
    IF Access:Client.TryOpen   !попробуем открыть файл Client
        ELSE                                            !ошибка
        MESSAGE('Could not open the Client file')       !обработать ошибку
        RETURN
        END
```

```
        !код программы
        Access:Client.Close                 !закрыть файл Client
        Access:Client.Kill                  !завершить объект Access:Client
        GlobalErrors.Kill                   !завершить объект GlobalErrors
```

См. Также: Open

TryPrevious (пытается прочесть предыдущую запись при последовательной обработке)

TryPrevious, PROC

Метод **TryPrevious** читает предыдущую запись при последовательной обработке. Метод TryPrevious не пытается обрабатывать ошибки, происходящие в процессе чтения записи.

Previous является альтернативным методом для чтения предыдущей записи при последовательной обработке. Отличие состоит в автоматической обработке ошибок.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: TryPrevious пытается прочесть предыдущую запись при последовательной обработке. Если это удается, возвращается Level:Benign (объявлен в файле ABERROR.INC). В противном случае возвращается уровень серьезности произошедшей ошибки. Более подробно об уровнях серьезности ошибок см. *ErrorClass*.

Тип результата: BYTE

Пример:

```
PROGRAM
        INCLUDE('ABFILE.INC')               !объявить объектFileManager
        MAP

        END
GlobalErrors ErrorClass                     !объявить объект GlobalErrors
Access:Client CLASS(FileManager)            !породить объект Access:Client
        END
        CODE                                !код программы
        LOOP                                !цикл по файлу
        CASE Access:Cliet.TryPrevious()             !прочесть очередную запись
        OF Level:NotifyOROF Level:Fatal             !ошибка
        POST(Event:CloseWindow)                     !закрыть окно
        BREAK
        ELSE                                        !иначе
```

```
        PRINT(Rpt:Detail)                    !распечатать запись
                                END
        END
```

См. Также: Previous

TryPrimeAutoInc (пытается подготовить запись с автоинкрементируемыми полями для добавления)

TryPrimeAutoInc, VIRTUAL, PROC

Метод **TryPrimeAutoInc** подготавливает запись с автоинкрементируемыми полями для добавления в обрабатываемый файл. Метод TryPrimeAutoInc не пытается обрабатывать ошибки.

PrimeAutoInc является альтернативным методом для выполнения автоинкремента полей. Разница состоит в автоматической обработке ошибок.

Метод CancelAutoInc восстанавливает файл в состояние, в котором тот находился до вызова метода TryPrimeAutoInc.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: TryPrimeAutoInc пытается прочесть предыдущую запись при последовательной обработке. Если это удается, возвращается Level:Benign (объявлен в файле ABERROR.INC). В противном случае возвращается уровень серьезности произошедшей ошибки. Более подробно об уровнях серьезности ошибок см. *ErrorClass.*

Тип результата: BYTE

Пример:

```
PROGRAM
        INCLUDE('ABFILE.INC')              !объявить объектFileManager
        MAP

        END
```

```
GlobalErrors ErrorClass                    !объявить объект GlobalErrors
Access:Client CLASS(FileManager)           !породить объект Access:Client
Init          PROCEDURE
TryPrimeAutoInc PROCEDURE,VIRTUAL
              END

Client        FILE,DRIVER('TOPSPEED'),PRE(CLI),CREATE,BINDABLE,THREAD
IDKey         KEY(CLI:ID),NOCASE,OPT,PRIMARY
Record        RECORD,PRE()
ID            LONG
Name          STRING(20)
StateCode     STRING(2)
              END
              END
InsertWindow WINDOW('Add a new Client'),AT(,,159,73),|
IMM,SYSTEM,GRAY
PROMPT('&Name:'),AT(8,20),USE(?CLI:Name:Prompt)
ENTRY(@s20),AT(61,20,84,10),USE(CLI:Name),|
 MSG('Client Name'),REQ
PROMPT('State Code:'),AT(8,34),|
 USE(?CLI:StateCode:Prompt)
ENTRY(@s2),AT(61,34,40,10),USE(CLI:StateCode),|
 MSG('State Code')
BUTTON('OK'),AT(12,53,45,14),USE(?OK),DEFAULT
BUTTON('Cancel'),AT(82,53,45,14),USE(?Cancel)
              END
              CODE
  GlobalErrors.Init                        !инициализировать объект GlobalErrors
  Access:Client.Init                       !инициализировать объект Access:Client
              Access:Client.Open           !открыть файл Client
              IF Access:Client.TryPrimeAutoInc()          !подготовить запись
              POST(Event:CloseWindow)                     !ошибка - завершить работу
              END
              OPEN(InsertWindow)
              ACCEPT
              CASE FIELD()
              OF ?OK
              IF EVENT() = Event:Accepted          !по кнопке OK
              IF Access:Client.Insert() = Level:Benign      !добавление завершено
              POST(Event:CloseWindow)              !все нормально - закрыть окно
              ELSE                                        !если ошибка
              SELECT(?CLI:Name:Prompt)             !выбрать соответствующее поле
              CYCLE                                !начать все сначала
```

```
        END
        END
        OF ?Cancel
        IF EVENT() = EVENT:Accepted          !по кнопке Cancel
        Access:Client.CancelAutoInc          !восстановить файл в исходное
                                             !состояние
        POST(Event:CloseWindow)              !закрыть окно
        END
        EMD
        END
        Access:Client.Close                  !закрыть файл Client
        Access:Client.Kill                   !завершить объект Access:Client
        GlobalErrors.Kill                    !завершить объект GlobalErrors
        RETURN

Access:Client.PrimeAutoInc PROCEDURE
        CODE
        !ваш код
        PARENT.PrimeAutoInc                  !вызов метода базового класса
                                             !ваш код
```

См. Также: CancelAutoInc, PrimeAutoInc

TryReget (пытается прочесть запись по ее позиции)

TryReget(*позиция* **), PROC**

TryReget	Читает запись по ее позиции.
позиция	Строковая константа, переменная, EQUATE или выражение, указывающая позицию записи в файле. Обычно это результат, возвращаемый методом Position.

Метод **TryReget** читает запись по ее позиции и возвращает результат, сигнализирующий об успешном завершении или об ошибке.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод **TryReget** пытается прочесть указанную запись. Если это удается, возвращается Level:Benign (объявлен в файле ABERROR.INC). В противном случае возвращается уровень серьезности произошедшей ошибки. Более подробно об уровнях серьезности ошибок см. *ErrorClass*.

Тип результата: BYTE

См. Также: Position

TryUpdate (пытается изменить текущую запись)

TryUpdate, PROC

Метод **TryUpdate** изменяет (переписывает) текущую запись. Метод TryUpdate не пытается обрабатывать ошибки, возникающие в процессе обновления записи.

Метод Update является альтернативным методом для обновления записей. Отличие состоит в автоматической обработке ошибок.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: TryUpdate пытается обновить запись. Если это удается, возвращается Level:Benign (объявлен в файле ABERROR.INC). В противном случае возвращается уровень серьезности произошедшей ошибки. Более подробно об уровнях серьезности ошибок см. *ErrorClass*.

Примечание: Этот метод не поддерживает ссылочную целостность между связанными файлами. Поддержанием ссылочной целостности занимается класс RelationManager.

Тип результата: BYTE

См. Также: Update

Update (изменяет текущую запись)

Update, PROC

Метод **Update** изменяет (переписывает) текущую запись. Метод Update автоматически обрабатывает ошибки, возникающие в процессе обновления записи.

Метод TryUpdate является альтернативным методом для обновления записей. Отличие состоит в необходимости ручной обработки ошибок.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Если метод Update завершается успешно, возвращается Level:Benign (объявлен в файле ABERROR.INC). В противном случае возвращается уровень серьезности последней произошедшей ошибки. Более подробно об уровнях серьезности ошибок см. *ErrorClass*.

Примечание: Этот метод не поддерживает ссылочную целостность между связанными файлами. Поддержанием ссылочной целостности занимается класс RelationManager.

Тип результата: BYTE

См. Также: TryUpdate

UseFile (сигнализирует об использовании файла с отложенным открытием)

UseFile, PROC

Метод **UseFile** сообщает объектам библиотеки ABC, что файл, чье открытие отложено при помощи свойства LazyOpen, должен быть использован. UseFile возвращает результат, показывающий, готов ли файл к использованию. Значение, равное Level:Benign, означает готовность файла, любое другое значение означает проблему.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод UseFile возвращает значения, объявленные в файле ABERROR.INC. Более подробно об уровнях серьезности ошибок см. *ErrorClass*.

Тип результата: BYTE

См. Также: LazyOpen

ValidateField (проверяет поле)

ValidateField(*поле* **), VIRTUAL, PROC**

ValidateField Проверяет текущее значение указанного поля и возвращает индикатор успеха.

поле Численная константа, переменная, EQUATE или выражение, идентифицирующая проверяемое поле. Поле идентифицируется порядковым номером в структуре FILE. Значение, равное единице (1) означает первое поле, двум (2) - второе и т.д.

Метод **ValidateField** проверяет текущее значение указанного поля и возвращает индикатор успеха.

Рассматриваемый метод является виртуальным, следовательно другие методы базового класса могут непосредственно обращаться к этому методу в

порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод ValidateField просто возвращает ноль (0). По существующим соглашениям, возвращенный нулевой результат означает успех, любое другое значение - проблему. ABC-шаблоны порождают метод ValidateField для каждого конкретного файла, который реализует указанные в словаре БД проверки значений полей.

Методы ValidateFields и ValidateRecord вызывают метод ValidateField для каждого поля в рамках своей компетенции.

Тип результата: BYTE

Пример:

```
MyFile     FILE,DRIVER('TOPSPEED'),THREAD
Record     RECORD,PRE()
TGroup     GROUP                      !поле 1
Name       STRING(20)                 !поле 2
Name2      STRING(20)                 !поле 3
FirstName  STRING(10),OVER(Name2)     !поле 4
           END
Another    STRING(10)                 !поле 5
           END
           END
           CODE                       !код программы
Access:MyFile.ValidateField(4)        !проверить FirstName
```

См. Также: ValidateFields, ValidateRecord

ValidateFields (проверяет диапазон полей)

ValidateFields(*первое поле, последнее поле* [*,неверное поле*] **), VIRTUAL, PROTECTED, PROC**

ValidateField	Проверяет текущие значения полей из указанного диапазона.
первое поле	Численная константа, переменная, EQUATE или выражение, идентифицирующая первое проверяемое поле. Поле идентифицируется порядковым номером в структуре FILE. Значение, равное единице (1), означает первое поле, двум (2) - второе и т.д.
последнее поле	Численная константа, переменная, EQUATE или выражение, идентифицирующее последнее проверяемое поле. Поле

	идентифицируется порядковым номером в структуре FILE. Значение, равное единице (1), означает первое поле, двум (2) - второе и т.д.
неверное поле	Переменная типа SIGNED, получающая идентификатор первого неверного поля. Поле идентифицируется порядковым номером в структуре FILE. Значение, равное единице (1), означает первое поле, двум (2) - второе и т.д. Если параметр опущен, вызывающая процедура не имеет информации о том, какое поле не прошло проверку.

Метод **ValidateField** проверяет текущие значения полей из указанного диапазона, возвращает результат, по которому можно судить об успехе проверки, и может указать поле, не прошедшее проверку.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROTECTED, следовательно, может быть вызван только из методов класса FileManager или порожденного от него класса.

Этот метод имеет атрибут PROC, следовательно его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод ValidateFields вызывает метод ValidateField для каждого поля из диапазона *первое поле - последнее поле*.

Тип результата: BYTE
См. Также: ValidateField

ValidateRecord (проверяет все поля)

ValidateRecord([*неверное поле*] **), VIRTUAL**

ValidateRecord	Проверяет текущие значения всех полей записи и возвращает индикатор успеха.
неверное поле	Переменная типа SIGNED, получающая идентификатор первого неверного поля. Поле идентифицируется порядковым номером в структуре FILE. Значение, равное единице (1) означает первое поле, двум (2) - второе и т.д. Если параметр опущен, вызывающая процедура не имеет информации о том, какое поле не прошло проверку.

Метод **ValidateRecord** проверяет текущие значения всех полей записи, возвращает результат, по которому можно судить об успехе проверки и может

указать поле, не прошедшее проверку.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод ValidateRecord вызывает метод ValidateField для каждого поля записи.

Тип результата: BYTE

См. Также: ValidateFiel

Класс RelationManager

Обзор

Класс RelationManager объявляет диспетчер отношений - объект, выполняющий следующие функции.

• Полно и гибко определяет отношения между файлами базы данных, причем эти отношения не обязательно должны быть определены в словаре БД, их можно определять непосредственно (динамически) средствами класса RelationManager.

• С высокой надежностью обеспечивает выполнение ограничений ссылочной целостности (RI - referential integrity) между связанными файлами, причем эти ограничения не обязательно должны быть определены в словаре БД, их можно определять непосредственно (динамически) средствами класса RelationManager.

• Распространяет команды управления файлом на связанные с ним файлы. Например, когда объект открывает свой первичный файл, он открывает и все связынные с ним файлы.

Класс RelationManager предоставляет методы “настройки”, позволяющие вам описать отношения между файлами, поля связи этих отношений и ограничения ссылочной целостности. Кроме того, существует ряд методов, выполняющих каскадные или принудительные операции, такие, как открытие и закрытие файлов, удаление и изменение записей.

Концепции и соглашения класса RelationManager

Каскадирование команд и ссылочные ограничения

Отношения между файлами можно представить при помощи серии диспетчеров отношений (объектов RelationManager) - по одному объекту на каждый файл. Каждый объект описывает отношения между своим первичным файлом и всеми файлами, имеющими с ним *непосредственные* отношения. Однако, каждому объекту RelationManager также известно о диспетчерах отношений этих связанных файлов. Таким образом, объекту *опосредованно* известны также и вторичные отношения.

Например, рассмотрим три связанных файла: Customer <->> Order <->> Item, где <->> означает отношение один ко многим. Объекту RelationManager для файла

Customer известно об отношении между файлами Customer и Order; но также известно ему и о диспетчере отношений файла Order, а, следовательно, и об отношении между файлами Order и Item.

Преимущество подобной организации цепочки отношений состоит в том, что вы можете выполнять действия с любым файлом. При этом, если вы открываете или закрываете файл, будут открыты или закрыты все необходимые файлы, находящиеся в цепочке как выше, так и ниже файла, с которым вы работаете. Аналогично, при модификациях файла, будет обеспечена ссылочная целостность как вверх, так и вниз по цепочке отношений.

"Я" и "он"

Некоторые методы класса RelationManager ссылаются на свой первичный файл как на "MyFile" или "Me", а на связанные файлы как на "HisFile" или "Him". Более подробно см. *Свойства класса RelationManager*.

Левый и правый (и буфер)

Некоторые методы класса RelationManager ссылаются на буфер записи своего первичного файла как на "Left" (левый), буфер ассоциированной очереди - как на "Right" (правый) и на ассоциированную область промежуточного хранения как на "Buffer" (буфер). Более подробно см. *Классы обработки пар полей*.

Реализация ABC-шаблонов

ABC-шаблоны *порождают* от RelationManager классы для каждого файла, с которым работает программа. Порожденные классы называются Hide:Relate:*имя_файла,* но к ним можно обращаться и по имени Relate:*имя_файла*. Эти порожденные классы реализованы в генерируемых файлах *appna*BC0.CLW - *appna*BC9.CLW (их количество зависит от количества файлов, с которыми работает приложение). Методы порожденных классов отражают специфику обрабатываемых файлов, их отношений с другими файлами и ограничений ссылочной целостности, указанной в словаре БД.

ABC-шаблоны генерируют процедуры, которые инициализируют и завершают объекты RelationManager. Эти процедуры называются DctInit и DctKill.. Они генерируются в файле *appna*BC.CLW.

Порожденные классы RelationManager можно настраивать при помощи диалога **Global Properties** в генераторе приложений. Более подробно см. *Обзор шаблонов - Управление файлами* и *Управление классами* (*Template Overview—File Control Options*

и *Classes Options*).

Взаимоотношения с другими ABC-классами

Классы FileManager и BufferedPairsClass

Класс RelationManager значительную часть своей работы выполняет, основываясь на классах FileManager и BufferedPairsClass. Таким образом, если в вашей программе объявлены объекты RelationManager, следует также объявить классы FileManager и BufferedPairsClass. Это в основном делается автоматически, когда вы включаете заголовок RelationManager (файл ABFILE.INC) в секции объявления данных вашей программы. Более подробно см. *Концептуальный пример*, *Класс FileManager* и *Классы обработки пар полей*.

ViewManager

Возможно, более значительным является то, что RelationManager служит основой (или "мальчиком на посылках") для класса ViewManager. Если в вашей программе объявлены объекты ViewManager, следует также объявить класс RelationManager. Более подробно см. *Класс ViewManager*.

Исходные модули

Исходные модули класса RelationManager по умолчанию устанавливаются в каталог \CLARION4\LIBSRC. Конкретные файлы и соответствующие компоненты перечислены ниже:

ABFILE.INC	Объявления RelationManager
ABFILE.CLW	Определения методов RelationManager

Концептуальный пример

Приведеденный ниже пример иллюстрирует типичную последовательность операторов, объявляющих, создающих, инициализирующих, использующих и завершающих объекты класса FileManager.

Этот пример использует RelationManager для каскадного обновления ключевых полей в записях родительского и связанного дочернего файлов.

```
PROGRAM
INCLUDE('ABFILE.INC')
INCLUDE('ABREPORT.INC')
MAP
END
```

```
CUSTOMER
FILE,DRIVER('TOPSPEED'),NAME('CUSTOMER'),PRE(CUS),|

BINDABLE,CREATE,THREAD
BYNUMBER
KEY(CUS:CUSTNO),NOCASE,OPT,PRIMARY
Record     RECORD,PRE()
CUSTNO     LONG
NAME       STRING(30)
ZIP        DECIMAL(5)
           END
           END

PHONES   FILE,DRIVER('TOPSPEED'),NAME('PHONES'),PRE(PHO),|
BINDABLE,CREATE,THREAD
BYCUSTOMER KEY(PHO:CUSTNO,PHO:PHONE),DUP,NOCASE,OPT
Record     RECORD,PRE()
CUSTNO     LONG
PHONE      STRING(20)
TYPE       STRING(8)
           END
           END

GlobalErrors  ErrorClass

Access:CUSTOMER  CLASS(FileManager)
Init       PROCEDURE
           END
Relate:CUSTOMER  CLASS(RelationManager)
Init       PROCEDURE
END

Access:PHONES  CLASS(FileManager)
Init       PROCEDURE
END
Relate:PHONES  CLASS(RelationManager)
Init       PROCEDURE
END
```

```
RecordsPerCycle     LONG(25)
StartOfCycle        LONG,AUTO
PercentProgress  BYTE
ProgressMgr      StepLongClass
```

```
    CustView  VIEW(CUSTOMER)
      END
    Process   ProcessClass
    Progress:Bar     BYTE
    ProgressWindow  WINDOW('Processing...'),AT(,,142,59),|

 CENTER,TIMER(1),GRAY,DOUBLE
PROGRESS,USE(Progress:Bar),AT(15,15,111,12),RANGE(0,100)
STRING(''),AT(0,3,141,10),USE(?Progress:UserString),CENTER
            STRING(''),AT(0,30,141,10),USE(?Progress:Text),CENTER
            BUTTON('Cancel'),AT(45,42,50,15),USE(?Progress:Cancel)
                                        END
            CODE
            GlobalErrors.Init
            Relate:CUSTOMER.Init
            Relate:PHONES.Init
            ProgressMgr.Init(ScrollSort:AllowNumeric)
            Process.Init(CustView,Relate:CUSTOMER,|
            ?Progress:Text,Progress:Bar,|
            ProgressMgr,CUS:CUSTNO)
            Process.AddSortOrder( CUS:BYNUMBER )
            Relate:CUSTOMER.Open
            OPEN(ProgressWindow)
            ?Progress:Text{Prop:Text} = '0% Completed'
            ACCEPT
            CASE EVENT()
            OF Event:OpenWindow
            Process.Reset
            IF Process.Next()
            POST(Event:CloseWindow)
            CYCLE
            END
            OF Event:Timer
            StartOfCycle=Process.RecordsProcessed
LOOP WHILE Process.RecordsProcessed-StartOfCycle < |
                RecordsPerCycle
CUS:CUSTNO+=100                    !Изменить ключевое поле родительской
                                    !записи
IF Relate:CUSTOMER.Update()          !Каскадное изменение дочерних записей
            BREAK
            END
CASE Process.Next()
            OF Level:Notify
```

```
            ?Progress:Text{Prop:Text} = 'Process Completed'
DISPLAY(?Progress:Text)
POST(EVENT:CloseWindow)
            BREAK
OF Level:Fatal
POST(EVENT:CloseWindow)
BREAK
END
END
END
            CASE FIELD()
            OF ?Progress:Cancel
            CASE Event()
            OF Event:Accepted
            POST(Event:CloseWindow)
            END
            END
            END
            ProgressMgr.Kill
            Relate:CUSTOMER.Close
            Relate:CUSTOMER.Kill
            Relate:PHONES.Kill
            GlobalErrors.Kill

Access:CUSTOMER.Init  PROCEDURE
            CODE
            PARENT.Init(Customer, GlobalErrors)
            SELF.FileNameValue = 'CUSTOMER'
            SELF.Buffer &= CUS:Record
            SELF.AddKey(CUS:BYNUMBER,'CUS:BYNUMBER',1)

Relate:CUSTOMER.Init PROCEDURE
            CODE
            Access:CUSTOMER.Init
            PARENT.Init(Access:CUSTOMER,1)

SELF.AddRelation(Relate:PHONES,RI:Cascade,RI:Restrict,PHO:BYCUSTOMER)
            SELF.AddRelationLink(CUS:CUSTNO,PHO:CUSTNO)

Access:PHONES.Init  PROCEDURE
            CODE
            PARENT.Init(Phones, GlobalErrors)
            SELF.FileNameValue = 'PHONES'
```

```
            SELF.Buffer &= PHO:Record
            SELF.AddKey(PHO:BYCUSTOMER,'PHO:BYCUSTOMER')
Relate:PHONES.Init PROCEDURE
            CODE
            Access:PHONES.Init
            PARENT.Init(Access:PHONES,1)
            SELF.AddRelation( Relate:CUSTOMER )
```

Свойства класса RelationManager

Ниже описаны свойства класса RelationManager.

Me (объект класса FileManager для первичного файла)

Me &FileManager

Свойство **Me** представляет собой ссылку на объект класса FileManager для первичного файла объекта RelationManager. По определению, файл, на который ссылается этот объект, является первичным файлом объекта RelationManager. Свойство Me идентифицирует объект FileManager первичного файла для использования в различных методах RelationManager.

Реализация: Метод Init устанавливает значение свойства Me.
См. Также: Init

UseLogout (флаг использования транзакций)

UseLogout BYTE

Свойство **UseLogout** опредляет, должны ли каскадные обновления или удаления выполняться в рамках транзакции (LOGOUT/COMMIT). Нулевое значение (0) означает отсутствие транзакций; значение, равное единице (1), включает использование транзакций.

Реализация: Метод Init устанавливает значение свойства UseLogout.

ABC-шаблоны устанавливают значение свойства UseLogout на основании значения флага **Enclose RI code in transaction frame** в диалоге **Global Properties**.

См. Также: Init

Методы класса RelationManager

Функциональная организация - ожидаемое использование

В качестве помощи для понимания RelationManager, можно разделить многообразные методы этого класса на две большие категории в соответствии с их ожидаемым использованием - первичный интерфейс и виртуальные методы. Такое структурирование отражает то, что мы понимает под типичным использованием методов RelationManager.

Методы первичного интерфейса

Методы первичного интерфейса, которые вы, скорее всегоб явно вызываете в своих программах, могут в свою очередь быть разделены на три категории:

Разового применения:

Init	инициализирует объект RelationManager
AddRelation	устанавливает отношение
AddRelationLink	устанавливает поля связи для отношения
SetAlias	добавляет/устанавливает файл-псевдоним
Kill	завершает объект RelationManager

Основногопременения:

Open[V]	открывает файл и связанные с ним файлы
Save[V]	копирует текущую запись и все связанные записи
Update[V]	обновляет текущую запись с учетом ограничений ссылочной целостности
Delete[V]	удаляет текущую запись с учетом ограничений ссылочной целостности
Close[V]	закрывает файл и связанные с ним файлы

[V] Эти методы также являются виртуальными.

Эпизодического применения:

ListLinkingFields	планирует пары полей связи
SetQuickScan	разрешает QuickScan по связанным файлам

Виртуальные методы

Мы предполагаем, что у вас возникнет необходимость переопределить эти методы, и, поскольку они являются виртуальными, сделать это несложно. В

противном случае, и их стандартное поведение представляется разумным.

Open	открывает файл и связанные с ним файлы
CancelAutoInc	отменяет действия rimeAutoInc
Save	копирует текущую запись и все связанные записи
Update	обновляет текущую запись с учетом ограничений ссылочной целостности
Delete	удаляет текущую запись с учетом ограничений ссылочной целостности
Close	закрывает файл и связанные с ним файлы

AddRelation (устанавливает отношение между файлами)

AddRelation(*диспетчер связей* [,*режим обновления* ,*режим удаления* ,*ключ связи*] **), PROTECTED**

AddRelation	Описывает отношение между первичным файлом объекта (см. свойство *Me*) и другим файлом.
диспетчер связей	Метка объекта RelationManager для связанного файла.
режим обновления	Численная константа, переменная, EQUATE или выражение, указывающее какой вид поддержки ссылочной целостности должен осуществляться при изменениях полей связи первичного файла. Допускаются следующие виды: отсутствие контроля, очистка полей связи, запрещение изменения полей связи первичного файла и каскадное изменение. Если параметр опущен, параметры *режим удаления* и *ключ связи* также должны быть опущены. При этом ссылочная целостность не поддерживается.
режим удаления	Численная константа, переменная, EQUATE или выражение, указывающее, какой вид поддержки ссылочной целостности должен осуществляться при удалении записи первичного файла. Допускаются следующие виды: отсутствие контроля, очистка полей связи, запрещение изменения полей связи первичного файла и каскадное изменение. Если параметр опущен, параметры *режим обновления* и *ключ связи* также должны быть опущены. При этом ссылочная целостность не поддерживается.
ключ связи	Метка ключа связанного файла, по которому строится отношение. Если параметр присутствует, вызов метода AddRelation должен сопровождаться последующим вызовом метода AddRelationLink для каждого ключевого поля. Если параметр опущен, также должны быть опущены параметры *режим обновления* и *режим удаления*.

Метод **AddRelation** в совокупности с методом AddRelationLink описывает отношение между первичным файлом объекта RelationManager (см. свойство *Me*) и другим файлом, чтобы дать возможность другим методам выполнять каскадные операции над связанными файлами и обеспечивать ссылочную целостность.

Этот метод имеет атрибут PROTECTED, следовательно, может быть вызван только из методов класса RelationManager или порожденного от него класса.

Реализация: Как правило, метод AddRelation следует вызывать после вызова метода Init (или в порожденном методе Init).

Константы для параметров *режим обновления* и *режим удаления* описаны в файле FILE.INC следующим образом:

```
        ITEMIZE(0),PRE(RI)
None    EQUATE          !не предпринимается никаких
                        !действий
Clear   EQUATE          !поля связи в связанных файлах
                        !очищаются
Restrict        EQUATE  !операция запрещена при наличии
                        !записей в связанном файле
Cascade         EQUATE  !обновление полей связи в
                        !связанных файлах либо удаление
                        !записей в связанных файлах
                END
```

Пример:

```
Orders          FILE,DRIVER('TOPSPEED'),PRE(ORD),CREATE
ByCustomer          KEY(ORD:CustNo,ORD:OrderNo),DUP,NOCASE,OPT
Record            RECORD,PRE()
CustNo              LONG
OrderNo             LONG
OrderDate           LONG
Reference           STRING(24)
ShipTo              STRING(32)
Shipped             BYTE
Carrier             STRING(1)
                END
            END

Items           FILE,DRIVER('TOPSPEED'),PRE(ITEM),CREATE
AsEntered       KEY(ITEM:CustNo,ITEM:OrderNo,ITEM:LineNo),|
```

```
NOCASE,OPT,PRIMARY
Record          RECORD,PRE()
CustNo            LONG
OrderNo           LONG
LineNo           SHORT
ProdCode           SHORT
Quantity         SHORT
              END
            END
             CODE                          !код программы
Relate:Orders.Init PROCEDURE
 CODE
  SELF.AddRelation( Relate:Items,0,0, ITEM:AsEntered )
  SELF.AddRelationLink( ORD:CustNo, ITEM:CustNo )
  SELF.AddRelationLink( ORD:OrderNo, ITEM:OrderNo )
  SELF.AddRelation( Relate:Customer )
```

См. Также: AddRelationLink, Init

AddRelationLink (устанавливает поля связи для отношения)

AddRelationLink(*родитель, потомок* **), PROTECTED**

AddRelationLink	Идентифицирует поля связи для отношения между первичным файлом объекта (см. свойство *Me*) и другим файлом.
родитель	Метка поля связи первичного файла.
потомок	Метка поля связи другого файла.

Метод **AddRelationLink** в совокупности с методом AddRelation описывает отношение между первичным файлом объекта RelationManager (см. свойство *Me*) и другим файлом, чтобы дать возможность другим методам выполнять каскадные операции над связанными файлами и обеспечивать ссылочную целостность.

Вы должны вызывать метод AddRelationLink для каждой пары полей связи, причем вызовы должны следовать, начиная со старших полей ключа в сторону младших полей.

Этот метод имеет атрибут PROTECTED, следовательно, может быть вызван только из методов класса RelationManager или порожденного от него класса.

Реализация: Как правило, метод AddRelation следует вызывать после вызова метода Init (или в порожденном методе Init).

Пример:

```
Orders          FILE,DRIVER('TOPSPEED'),PRE(ORD),CREATE
ByCustomer          KEY(ORD:CustNo,ORD:OrderNo),DUP,NOCASE,OPT
Record            RECORD,PRE()
CustNo              LONG
OrderNo             LONG
OrderDate           LONG
Reference           STRING(24)
ShipTo             STRING(32)
Shipped             BYTE
Carrier            STRING(1)
                 END
               END
Items          FILE,DRIVER('TOPSPEED'),PRE(ITEM),CREATE
AsEntered        KEY(ITEM:CustNo,ITEM:OrderNo,ITEM:LineNo),|

NOCASE,OPT,PRIMARY
Record            RECORD,PRE()
CustNo              LONG
OrderNo             LONG
LineNo             SHORT
ProdCode             SHORT
Quantity           SHORT
                 END
               END
                CODE                          !код программы

Relate:Orders.Init PROCEDURE
 CODE
  SELF.AddRelation( Relate:Items,0,0, ITEM:AsEntered )
  SELF.AddRelationLink( ORD:CustNo, ITEM:CustNo )
  SELF.AddRelationLink( ORD:OrderNo, ITEM:OrderNo )
  SELF.AddRelation( Relate:Customer )
```

См. Также: AddRelation, Init

CancelAutoInc (отменяет автоинкремент)

CancelAutoInc, VIRTUAL, PROC

Метод **CancelAutoInc** восстанавливает обрабатываемый файл в состояние, в котором он находился до вызова метода PrimeAutoInc. Обычно это требуется при отмене операции добавления записи. CancelAutoInc возвращает результат, служащий иидикатором успеха. Нулевое значение (0 или Level:Benign) означает успешное

завершение, любое другое значение означает, что возникла проблема.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод CancelAutoInc вызывает метод FileManager.CancelAutoInc для основного файла, передавая SELF как параметр *relation manager*.

Return value EQUATEs are declared in ABERROR.INC as follows:

```
!Серьезность ошибки
Level:Benign          EQUATE(0)
Level:User            EQUATE(1)
Level:Program         EQUATE(2)
Level:Fatal           EQUATE(3)
Level:Cancel          EQUATE(4)
Level:Notify          EQUATE(5)
```

Тип результата: BYTE

Пример:

```
WindowManager.TakeCloseEvent PROCEDURE
 CODE
 IF SELF.Response <> RequestCompleted          !код процедуры
  IF SELF.OriginalRequest=InsertRecord |
            AND SELF.Response=RequestCancelled
   IF SELF.Primary.CancelAutoInc()             !отменить автоинкремент -
                                               !каскад
    SELECT(SELF.FirstField)
    RETURN Level:Notify
   END
  END
                                    !код процедуры
 END
 RETURN Level:Benign
```

См. Также: FileManager.CancelAutoInc, FileManager.PrimeAutoInc

Close (закрывает файл и все связанные файлы)

Close(*каскад* **), VIRTUAL, PROC**

Close	Закрывает первичный файл объекта (см. свойство *Me*) и все связанные файлы.
каскад	Численная константа, переменная, EQUATE или выражение, указывающее, был ли вызов этого метода рекурсивным. Нулевое (0) значение означает отсутствие рекурсии, единичное (1) - рекурсивный вызов. Это позволяет методу остановиться, после того, как он обработал все файлы в кольцевой цепочке отношений. Если параметр опущен, его значение принимается равным нулю (0). При вызове этого метода из ваших программ следует *всегда* опускать этот параметр.

Метод **Close** закрывает первичный фал объекта (см. свойство *Me*), если он не используется другими процедурами, а также все связанные файлы, и возвращает результат, показывающий, успешно ли завершилась операция.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод Close использует метод FileManager.Close для закрытия каждого файла. Возвращаемое значение есть результат выполнения метода FileManager.Close. Более подробно см. *Класс FileManager*.

Тип результата: BYTE

Пример:

```
Relate:Customer.Open                    !открыть файл Customer и связанные файлы
                                        !код программы
                                        !обработать фалы
        Relate:Customer.Close           !закрыть файл Customer и связанные файлы
```

См. Также: FileManager.Close

Delete (удаляет запись с поддержкой ссылочной целостности)

Delete([*подтверждение*]), VIRTUAL

Delete	Удаляет запись первичного файла с учетом ограничений ссылочной целостности
подтверждение	Целочисленная константа, переменная, EQUATE или выражение, указывающее, следует ли запрашивать у конечного пользователя подтверждение. Если значение параметра равно единице (1 или True), удаление происходит только с согласия пользователя. Если значение параметра равно нулю (0 или False), удаление происходит без запроса подтверждения. Если параметр опущен, его значение принимается равным единице (1).

Метод **Delete** удаляет текущую запись первичного файла (см. свойство *Me*), соблюдая ограничения ссылочной целостности в соответствии с их определениями, после чего возвращает результат, показывающий, успешно ли была завершена операция. Записи удаляются под одной транзакцией, если методу Init в качестве параметра *транзакция* была передана единица(1).

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Ограничения ссылочной целостности при удалении определяются методом AddRelation. Если имеется ограничение RI:Restrict, метод удаляет текущую запись только при условии, что она не имеет потомков в связанных файлах. Если установлено ограничение RI:Cascade, удаляются также все дочерние записи. Если ограничение имеет тип RI:None, безусловно удаляется только запись первичного файла. Если речь идет об ограничении RI:Clear, запись первичного файла, безусловно ,удаляется, а во всех дочерних записях зачищаются поля связи.

Метод Delete вызывает метод FileManager.Throw первичного файла для получения подтверждения от конечного пользователя.

Тип результата: BYTE

Пример:

```
DeleteCustomer PROCEDURE
        CODE
        Relate:Customer.Open        !открыть файл Customer и связанные файлы
        IF NOT GlobalErrors.Throw(Msg:ConfirmDelete)
                                    !получено подтверждение пользователя
        LOOP                        !повторение попыток удаления при неудаче
        IF Relate:Customer.Delete()
                                    !удалить с учетом ссылочной целостности
        IF NOT GlobalErrors.Throw(Msg:RetryDelete)
                                    !если не удалось удалить, предложим
                                    !повторить
        CYCLE                       !если пользователь согласен, попробуем
                                    !вновь
        END                         !в противном случае иде дальше
        END                         !если удалили или пользователь отказался -
        UNTIL 1                     !выход из цикла
        END
```

См. Также: AddRelation, Init

Init (инициализирует объект RelationManager)

Init(*диспетчер файла* [,*транзакция*] **)**

Init	Инициализирует объект RelationManager.
диспетчер файла	Метка объекта FileManager для первичного файла объекта RelationManager. По определению, файл, на который имеется ссылка через этот объект FileManager, является первичным файлом объекта RelationManager.
транзакция	Численная константа, переменная, EQUATE или выражение, определяющее, должны ли каскадные модификации и удаления производиться под транзакцией (LOGOUT/COMMIT). Нулевое (0) значение отключает использование транзакций, единичное (1) - включает. Если параметр опущен, его значение принимается равным нулю (0).

Метод **Init** инициализирует объект RelationManager. Чтобы стало возможным использование транзакций, все файлы в рамках транзакции должны иметь один и тот же драйвер, который должен поддерживать тразакции.

Реализация: Метод Init устанавливает значениея свойств Me и UseLogout.

ABC-шаблоны устанавливают параметр *транзакция* на основании значения переключателя **Enclose RI code in transaction frame** в диалоге **Global Properties**.

Пример:

```
        PROGRAM
        INCLUDE('FILE.INC')                    !объявить класс RelationManager
Access:Client CLASS(FileManager)               !объявить класс Access:Client
Init    PROCEDURE
END
                                               !объявить файл Client
Client  FILE,DRIVER('TOPSPEED'),PRE(CLI),THREAD
IDKey   KEY(CLI:ID),NOCASE,OPT,PRIMARY
Record  RECORD,PRE()
ID      LONG
Name    STRING(20)
StateCode STRING(2)
END
        END
 CODE
Access:Client.Init                             !инициализировать объект Access:Client
 Relate:Client.Init(Access:Client,1)           !инициализировать объект Relate:Client
                                               !с использованием транзакций
 Relate:Client.AddRelation( Relate:States )
                                               !установить отношение между файлами
                                               !Client и States
                                                !program code
 Relate:Client.Kill                            !завершить объект Relate:Client
 Access:Client.Kill                            !завершить объект Access:Client
```

См. Также: Me

Kill (завершает объект RelationManager)

Kill, VIRTUAL

Метод **Kill** освобождает всю память, выделенную за время жизни объекта и выполняет любой другой, необходимый при завершении код.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Пример:

```
        PROGRAM
        INCLUDE('FILE.INC')                    !объявить класс RelationManager
```

```
Access:Client CLASS(FileManager)      !объявить класс Access:Client
Init          PROCEDURE
              END
                                      !объявить файл Client
Client        FILE,DRIVER('TOPSPEED'),PRE(CLI),THREAD
IDKey         KEY(CLI:ID),NOCASE,OPT,PRIMARY
Record        RECORD,PRE()
ID            LONG
Name          STRING(20)
StateCode     STRING(2)
END
              END
 CODE
              Access:Client.Init              !инициализировать объект Access:Client
 Relate:Client.Init(Access:Client,1)          !инициализировать объект Relate:Client
                                              !с использованием транзакций
 Relate:Client.AddRelation( Relate:States )
                                      !установить отношение между файлами
                                      !Client и States
                                      !program code
 Relate:Client.Kill                   !завершить объект Relate:Client
 Access:Client.Kill                   !завершить объект Access:Client
```

ListLinkingFields (задает пары полей связи)

ListLinkingFields(*диспетчер связи, пары полей* [, *рекурсия*] **)**

ListLinkingFields	Задает пары полей связи между первичным и связанным файлом
диспетчер связи	Метка объекта RelationManager для связанного файла.
пары полей	Метка объекта FieldPairsClass, который будет хранить ссылки на поля связи.
рекурсия	Численная константа, переменная, EQUATE или выражение, которое показывает, что метод вызван рекурсивно. Нулевое значение (0) является индикатором нерекурсивного вызова, единичное (1) - рекурсивного. Это позволяет методу, при необходимости, получить список полей связи у *диспетчера связи* - поскольку лишь одна сторона в отношении ведет список полей связи. Если параметр опущен, его значение принимается равным нулю (0). Вам следует *всегда* опускать этот параметр при вызовах этого метода из своих программ.

Метод **ListLinkingFields** Задает пары полей связи между первичным и связанным файлом.

Реализация: Объект RelationManager не использует заданные пары полей связи, но предоставляет этот сервис для класса ViewManager и т.д..

Пример:

```
ViewManager.AddRange PROCEDURE( *? Field, RelationManager MyFile,|
        RelationManager HisFile)
        CODE                            !добавить проверку диапазона в просмотр
                                        !структуру
        SELF.Order.LimitType = Limit:File
                                            !установить тип диапазона - отношение
        MyFile.ListLinkingFields(HisFile,SELF.Order.RangeList)
                                            !получить поля связи
        ASSERT(RECORDS(SELF.Order.RangeList.List))
                                            !убедиться, что диапазон существует
        SELF.SetFreeElement             !установить свободный элемент ключа
```

См. Также:

Open (открывает файл и все связанные файлы)

Open(*каскад* **), VIRTUAL, PROC**

Open Открывает первичный файл (см свойство *Me*) и все связанные файлы.

каскад Численная константа, переменная, EQUATE или выражение, указывающее, был ли вызов этого метода рекурсивным. Нулевое (0) значение означает отсутствие рекурсии, единичное (1) - рекурсивный вызов. Это позволяет методу остановиться, после того, как он обработал все файлы в кольцевой цепочке отношений. Если параметр опущен, его значение принимается равным нулю (0). При вызове этого метода из ваших программ следует *всегда* опускать этот параметр.

Метод **Open** открывает первичный файл (см свойство *Me*) и все связанные файлы и возвращает результат, показывающий, успешно ли завершилась операция.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод Open использует метод FileManager.Open для открытия каждого файла. Возвращемый методом Open результат есть результат, возвращенный методом FileManager.Open. Более подробно см. *Класс FileManager*.

Тип результата: BYTE

Пример:

```
Relate:Customer.Open          !открыть файл Customer и связанные файлы
                              !код программы
                              !обработать фалы
Relate:Customer.Close         !закрыть файл Customer и связанные файлы
```

См. Также: FileManager.Open

Save (создает копии текущих записей первичного и всех связанных файлов)

Save, VIRTUAL

Метод Save создает копии текущих записей первичного и всех связанных файлов. Эти копии могут быть использованы для обнаружения последующих изменений текущих записей или для восстановления текущих записей в их предыдущее состояние.

Рассматриваемый метод является виртуальным, следовательно ,другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод Save использует метод BufferedPairsClass.AssignLeftToBuffer для сохранения каждой зааписи. Более подробно см. *Классы обработки пар полей*.

SetAlias (установить файл-псевдоним)

SetAlias(*диспетчер отношений* **)**

SetAlias Идентифицирует псевдоним для первичного файла объекта.
relationmanager Метка объекта RelationManager для файла-псевдонима.

Метод **SetAlias** идентифицирует псевдоним для первичного файла данного объекта, чтобы там, где это возможно, обрабатывать файл один раз. Например, если и первичный файл и псевдоним попадают в одну транзакцию (LOGOUT/COMMIT), RelationManager распознает псевдоним, и открывает транзакцию только на первичном файле.

Пример:

```
                                  !объявим файл Customer
Customer  FILE,DRIVER('TOPSPEED'),PRE(CLI),NAME('Customer')
```

```
IDKey     KEY(CLI:ID),NOCASE,OPT,PRIMARY
Record    RECORD,PRE()
ID        LONG
Name      STRING(20)
          END
          END
                                   !объявим файл-псевдоним Client
Client    FILE,DRIVER('TOPSPEED'),PRE(CUS),NAME('Customer')
IDKey     KEY(CUS:ID),NOCASE,OPT,PRIMARY
Record    RECORD,PRE()
ID        LONG
Name      STRING(20)
          END
          END
Relate:Customer.SetAlias( Relate:Client )      !Client = псевдоним файла Customer
```

SetQuickScan (управляет режимом ускоренного сканирования для первичного и связанных файлов)

SetQuickScan(*выключатель* [,*распространение*] **), VIRTUAL**

SetQuickScan	Управляет режимом ускоренного сканирования (QuickScan) для первичного файла и тех связанных файлов, которые заданы параметром *распространение*.
выключатель	Численная константа, переменная, EQUATE или выражение, разрешающее или запрещающее ускоренное сканирование. Нулевое (0) значение параметра запрещает ускоренное сканирование, единица (1) - разрешает.
распространение	Численная константа, переменная, EQUATE или выражение, которое указывает, на какие из связанных файлов распространяется действие данного метода. Допускаются следующие типы распространения: нет распространения, один ко многим, многие к одному, все. Если параметр опущен, распространение не производится.

Метод **SetQuickScan** управляет режимом ускоренного сканирования (QuickScan) для первичного файла и тех связанных файлов, которые заданы параметром *распространение*.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод SetQuickScan рассылает файловым драйверам (операция

SEND) управляющую строку с параметром QUICKSCAN для каждого указанного файла. Этот режим поддерживается драйверами ASCII, BASIC и DOS. Более подробно см. *Драйверы баз данных*.

Соответствующиие константы для режимов распространения объявлены в файле FILE.INC следующим образом:

```
ITEMIZE(0),PRE(Propagate)
None      EQUATE                !только первичный файл
OneMany   QUATE                 !только отношения 1:Много
ManyOne   EQUATE                !только отношения Много:1
All       EQUATE                !все связанные файлы
          END
```

Пример:

```
Relate:Customer.SetQuickScan(1,Propagate:OneMany)
                                !вкл. ускоренное сканирование для 1:Many
Relate:Orders.SetQuickScan(1)   !вкл. ускоренное сканирование для
                                !первичного файла
Relate:Orders.SetQuickScan(0)   !откл. ускоренное сканирование для
                                !первичного файла
```

Update (модифицирует запись с учетом ограничений ссылочной целостности)

Update(*из формы*), **VIRTUAL**

Update	Модифицирует запись первичного файла (см. свойство *Me*) с соблюдением ограничений ссылочной целостности.
из формы	Численная константа, переменная, EQUATE или выражение, которое указывает, был ли вызван данный метод из формы обновления, поддерживающей функцию хранения истории (восстановления) полей. Нулевое (0) значение означает отсутствие возможности восстановления, единичное (1) - ее присутствие. Это позволяет методу выводить соответствующее сообщение при возникновении ошибок обновления записи.

Метод **Update** модифицирует запись первичного файла (см. свойство *Me*), соблюдая ограничения ссылочной целостности, после чего возвращает результат, показывающий, насколько успешно завершилась операция.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Ограничения ссылочной целостности задаются методом AddRelation и касаются значений полей связи. При ограничении типа RI:Restrict обновления текущей записи не производится, если таковое обновление приведет к появлению записей-потомков, не имеющих родителя. При ограничении типа RI:Cascade метод модифицирует текущую запись, а также значения полей связи во всех дочерних записях. Если ограничение имеет тип RI:None, метод, безусловно, обновляет запись первичного файла. Если имеется ограничение типа RI:Clear, метод, безусловно, обновляет запись первичного файла и зачищает значение полей связи во всех дочерних записях.

Тип результата: BYTE

Пример:

```
ChangeOrder ROUTINE
        IF Relate:Orders.Update(0)      !обновить запись с соблюдением
                                        !ограничений ссылочной целостности
        MESSAGE('Update Failed')        !оповестить пользователя при ошибке
        ELSE                            !в противном случае
        POST(Event:CloseWindow)         !закрыть окно
        END
```

См. Также: AddRelation

Класс ViewManager - диспетчер просмотра

Обзор

Концепции класса ViewManager

Класс ViewManager обрабатывает структуры VIEW.

Обработка, которую выполняет класс ViewManager, включает определение различных порядков сортировки, диапазонов (ключевых фильтров), фильтров (неключевых) для структур VIEW, а также их применение. Кроме того, ViewManager открывает, буферирует, читает и закрывает VIEW. Обработка также включает в себя первоначальное заполнение и проверку записи первичного файла структуры VIEW при добавлении и обновлении записи.

Все вышеперечисленные сервисы, предоставляемые классом ViewManager, относятся к структуре VIEW, а не к файлу. Структура VIEW может заключать в себе некоторые или все поля одного или более связанных файлов. Концепция VIEW чрезвычайно мощна и является, пожалуй, основной в клиент-серверных средах с нормализованными данными. VIEW позволяет получить доступ к данным из различных файлов так же, как из одного и делает это весьма эффективно. Более подробно о структурах VIEW см. *VIEW* в *Описании языка*.

К тому же, ViewManager поддерживает буферирование (в отличие от некоторых драйверов), что позволяет процедурам типа “browse” демонстрировать мгновенное виртуальное быстродействие при выводе ранее прочитанных страниц данных. Буферирование (см. *BUFFER* в *Описании языка*) способно также оптимизировать производительность, если вы имеете дело с прямым драйвером какого-либо сервера БД (обычно основанного на SQL), поскольку такие драйверы способны оптимизировать обращения к серверу для минимизации сетевого трафика.

ViewManager предоставляет вам простой и надежный доступ ко всей мощи и скорости VIEW при помощи своих объектов. Таким образом, вы получаете эти скорость и мощь, не изобретая при этом колеса.

Отношения с другими ABC-классами

ViewManager выполняет большую часть своей работы на основе классов FieldPairsClass и RelationManager. Таким образом, если ваша программа создает объекты класса ViewManager, оно должна использовать и эти два класса. Это происходит автоматически, когда вы включаете заголовок класса ViewManager (ABFILE.INC) в секцию данных вашей программы. Более подробно см. *Классы обработки пар полей* и *Класс RelationManager*. См. также *Концептуальный пример*.

Возможно, более важным является то, что класс ViewManager служит основой классов BrowseClass и ProcessClass. То есть, оба этих класса порождаются от ViewManager.

BrowseClass—интерактивная VIEW

BrowseClass реализует интерактивную VIEW, которая включает возможности визуального промотра записей, прокрутки страниц, сортировки, поиска и обновления. Более подробно см. *Классы просмотра*.

ProcessClass—неинтерактивная VIEW

ProcessClass реализует поточную (неинтерактивую) VIEW с возможностями сортировки и обновления, однако без визуального просмотра, а, следовательно, без прокрутки страниц и поиска. Более подробно см. *ProcessClass*

Реализация ABC-шаблонов

Класс ViewManager служит основой для шаблонных процедур Browse, Report и Process, т. к. Все эти процедуры базируются на структуре VIEW.

Классы BrowseClass и ProcessClass порождаются от класса ViewManager, и ABC-шаблоны создают объекты этих классов, т. е. шаблоны не создают объекты класса ViewManager независимо от объектов BrowseClass и ProcessClass. Шаблонная процедура Browse создает объект BrowseClass, процедуры Process и Report - объекты ProcessClass.

Файлы исходного кода класса ViewManager

Файлы исходного кода класса ViewManager по умолчанию устанавливаются в каталог \CLARION4\LIBSRC. Ниже перечислены эти файлы и соответстующие компоненты:

ABFILE.INC	Объявления класса ViewManager
ABFILE.CLW	Определения методов класса ViewManager

Концептуальный пример

Следующий пример иллюстрирует типичную последовательность операторов, объявляющих, создающих, инициализирующих, использующих и завершающих объект класса ViewManager. Этот пример просто создает VIEW с определенным порядком сортировки, диапазоном и фильтром, затем обрабатывает результирующую выборку, которая удовлетворяет условиям диапазона и фильтра.

```
        PROGRAM
        INCLUDE('ABFILE.INC')      !объявить класс ViewManager
        MAP
        END

GlobalErrors        ErrorClass                  !объявить объект GlobalErrors
View:Customer       ViewManager                 !объявить объект View:Customer
                                                !объявить объект Access:Customer
Access:CUSTOMER  CLASS(FileManager)
Init        PROCEDURE
            END
                                                !объявить объект Relate:Customer
Relate:CUSTOMER    CLASS(RelationManager)
Init        PROCEDURE
            END

CUSTOMER
FILE,DRIVER('TOPSPEED'),PRE(CUS),THREAD,BINDABLE
BYNUMBER
KEY(CUS:CUSTNO),NOCASE,OPT,PRIMARY
Record      RECORD,PRE()
CUSTNO      LONG
NAME
STRING(30)
ZIP         DECIMAL(5)
            END
            END
Customer:View   VIEW(CUSTOMER)                  !объявить VIEW Customer
            END

Low         LONG                                !нижняя граница диапазона
```

```
High        LONG(1000)                  !верхняя граница диапазона
ProgressMsg    STRING(60)

ProgressWindow
WINDOW('Processing...'),AT(,,215,60),GRAY,TIMER(100)
STRING(@S60),AT(1,21,210,10),USE(ProgressMsg),CENTER
BUTTON('Cancel'),AT(87,37,45,14),USE(?Cancel)
END
            CODE
            GlobalErrors.Init           !инициализировать объект GlobalErrors
            Relate:CUSTOMER.Init        !инициализировать объект Relate:Customer
                                        !инициализировать объект View:Customer
            View:Customer.Init(Customer:View,Relate:CUSTOMER)
                                        !Добавить порядок сортировки BYNUMBER
View:Customer.AddSortOrder(CUS:BYNUMBER)
                                        !добавить вторичные порядки сортировки
View:Customer.AppendOrder('CUS:Name,CUS:ZIP')
                                        !добавить диапазон
View:Customer.AddRange(CUS:CUSTNO,Low,High)
                                        !добавить фильтр номер 1
View:Customer.SetFilter( 'CUS:ZIP=33066','1')
            Relate:CUSTOMER.Open        !открыть первичный и связанные файлы
            OPEN(ProgressWindow)        !открыть окно
            ProgressMsg='Processing...'
            ACCEPT
            CASE EVENT()
            OF Event:OpenWindow
                                    !открыть VIEW, применить диапазон и фильтр
View:Customer.Reset(1)
            OF Event:Timer
            CASE View:Customer.Next()   !прочесть следующую запись
            OF Level:Notify             !конец файла - стоп
            POST(EVENT:CloseWindow)
            BREAK
            OF Level:Fatal              !фатальная ошибка - стоп
            POST(EVENT:CloseWindow)
            BREAK
            END
            CUS:ZIP=33065               !обработать запись
            IF Relate:CUSTOMER.Update() !обновить первичный
                                        !и связанные файлы
            BREAK
            ELSE
```

```
ProgressMsg = CLIP(CUS:Name)&' zip changed to '&CUS:ZIP
DISPLAY(ProgressMsg)
END
END
IF FIELD() = ?Cancel                    !отказ пользователя - стоп
IF EVENT() = Event:Accepted
POST(Event:CloseWindow)
END
END
END
Relate:CUSTOMER.Close                   !закрыть первичный и связанные файлы
View:CUSTOMER.Kill                      !завершить объект View:Customer
Relate:CUSTOMER.Kill                    !завершить объект Relate:Customer
GlobalErrors.Kill                       !завершить объект GlobalErrors

Access:CUSTOMER.Init                PROCEDURE
        CODE
        PARENT.Init(Customer,GlobalErrors)
        SELF.FileNameValue = 'CUSTOMER.TPS'
        SELF.Buffer &= CUS:Record
        SELF.AddKey(CUS:BYNUMBER,'CUS:BYNUMBER',1)
        SELF.LazyOpen = False

Relate:CUSTOMER.Init                PROCEDURE
        CODE
        Access:CUSTOMER.Init
        PARENT.Init(Access:CUSTOMER,1)
```

Свойства класса ViewManager

Свойства класса ViewManager включают ссылки на обрабатываемую структуру VIEW, а также различные флаги и переключатели, которые указывают диспетчеру просмотра способ ее обработки.

Имеются ссылки на структуру VIEW, объект RelationManager для ее первичного файла и информацию о сортировке VIEW. Эти ссылки позволяют универсальному в прочих отношениях объекту ViewManager обрабатывать конкретную VIEW.

К переключателям относятся параметры буферирования, которые позволяют осуществлять асинхронное упреждающее чтение страниц и сохранение уже считанных страниц. Это буферирование обеспечивает быструю реакцию процедур страничного просмотра записей и может также минимизировать сетевой трафик для

клиент-серверных программ за счет уменьшения количества передаваемых пакетов.

Каждое из этих свойств полностью описано ниже.

Order (поярдок сортировки, диапазон, фильтр)

Order &SortOrder, PROTECTED

Свойство **Order** является ссылкой на структуру, которая содержит информацию о порядке сортировки, диапазоне и фильтре для обрабатываемой структуры VIEW. Методы класса ViewManager используют эту информацию для сортировки, выделения диапазона и фильтрации записей для результирующей выборки.

Это свойство имеет атрибут PROTECTED, следовательно, обращаться к нему можно только из методов класса ViewManager или порожденного от него класса.

Некоторые методы класса ViewManager могут изменять содержимое свойства Order. К таким методам относятся AddSortOrder, AddRange, AppendOrder и SetFilter. Метод SetOrder переопределяет указанный порядок сортировки, а метод SetSort определяет, какой из порядков сортировки является текущим для соответствующей VIEW.

Реализация: Свойство Order является ссылкой на QUEUE, объявленную в файле ABFILE.INC:

```
FilterQueue  QUEUE,TYPE
ID           STRING(30)                !отсортировано в порядке
                                       !приоритета
Filter       &STRING                   !выражение фильтра
             END

SortOrder    QUEUE,TYPE                !информация о сортировке и
                                       !фильтре
Filter       &FilterQueue              !список выражений фильтров
FreeElement  ANY                       !свободный элемент ключа
LimitType    BYTE                      !тип диапазона
MainKey      &KEY                      !главный ключ
Order        &STRING                   !список полей сортировки
RangeList    &BufferedPairsClass       !список полей в
                                       !диапазоне
             END
```

См. Также: AddSortOrder, AddRange, AppendOrder, SetFilter, SetOrder, SetSort

PagesAhead (буферизированные страницы записей)

PagesAhead USHORT

Свойство **PagesAhead** управляет автоматическим буферированием записей выборки для обрабатываемой структуры VIEW (см. *BUFFER* в *Описании языка*). Некоторые драйверы БД не поддерживают буферизацию. PagesAhead указывает количество дополнительных "страниц" записей, которое должно быть прочитано следом за текущей показываемой страницей.

Реализация: Метод Init устанавливает значение свойства PagesAhead равным нулю (0). Метод Open обеспечивает буферирование, параметры которого задаются свойствами PagesAhead, PagesBehind, PageSize и TimeOut.

См. Также: Init, Open, PagesBehind, PageSize, TimeOut

PagesBehind (буферизированные страницы записей)

PagesBehind USHORT

Свойство **PagesBehind** управляет автоматическим буферированием записей выборки для обрабатываемой структуры VIEW (см. *BUFFER* в *Описании языка*). Некоторые драйверы БД не поддерживают буферизацию. PagesBehind указывает количество "страниц" уже прочитанных записей, которое следует сохранить в памяти.

Реализация: Метод Init устанавливает значение свойства PagesBehind равным двум (2). Метод Open обеспечивает буферирование, параметры которого задаются свойствами PagesAhead, PagesBehind, PageSize и TimeOut.

См. Также: Init, Open, PagesAhead, PageSize, TimeOut

PageSize (размер страницы буфера)

PageSize USHORT

Свойство **PageSize** управляет автоматическим буферированием записей выборки для обрабатываемой структуры VIEW (см. *BUFFER* в *Описании языка*). Некоторые драйверы БД не поддерживают буферизацию. PageSize указывает количество записей в "странице" буфера.

Реализация: Метод Init устанавливает значение свойства PageSize равным двадцати (20). Метод Open обеспечивает буферирование, параметры которого задаются свойствами PagesAhead, PagesBehind, PageSize и TimeOut.

См. Также: Init, Open, PagesAhead, PagesBehind, TimeOut

Primary (объект RelationManager для первичного файла)

```
Primary     &RelationManager, PROTECTED
```

Свойство **Primary** представляет собой ссылку на объект RelationManager для первичного файла обрабатываемой VIEW. Методы класса ViewManager используют это свойство для поддержания ссылочной целостности между связанными файлами обрабатываемой структуры VIEW.

Это свойство имеет атрибут PROTECTED, следовательно, обращаться к нему можно только из методов класса ViewManager или порожденного от него класса.

Метод ViewManager.Init method устанавливает значение свойства Primary.
См. Также: Init

TimeOut (обновление буферированных страниц)

```
TimeOut     USHORT
```

Свойство **TimeOut** управляет автоматическим буферированием записей выборки для обрабатываемой структуры VIEW (см. *BUFFER* в *Описании языка*). Некоторые драйверы БД не поддерживают буферизацию.

Свойство TimeOut указывает число секунд, в течение которых буферированные записи считаются "достоверными" в сетевой среде. Если период, заданный значением свойства TimeOut истек, VIEW читает записи непосредственно из базы данных, а не из буфера.

Реализация: Метод Init устанавливает значение свойства TimeOiut равным шестидесяти (60). Метод Open обеспечивает буферирование, параметры которого задаются свойствами PagesAhead, PagesBehind, PageSize и TimeOut.
См. Также: Init, Open, PagesAhead, PagesBehind, PageSize

View (обрабатываемая VIEW)

```
View    &VIEW
```

Свойство **View** представляет собой ссылку на обрабатываемую структуру VIEW. Это свойство просто идентифицирует обрабатываемую VIEW для различных методов класса ViewManager.

Метод ViewManager.Init method устанавливает значение свойства View.
См. Также: Init

Методы ViewManager

Функциональная организация - ожидаемое использование

В качестве помощи для понимания ViewManager, можно разделить многообразные методы этого класса на две большие категории в соответствии с их ожидаемым использованием - первичный интерфейс и виртуальные методы. Такое структурирование отражает то, что мы понимает под типичным использованием методов ViewManager.

Методы первичного интерфейса

Методы первичного интерфейса, которые вы, скорее всего, явно вызываете в своих программах, могут, в свою очередь, быть разделены на три категории:

Разового применения:

Init	инициализирует объект ViewManager
AddRange	добавляет диапазон к активному порядку сортировки
AddSortOrder	добавляет порядок сортировки
AppendOrder	уточняет порядок сортировки
Kill[V]	завершает объект ViewManager

Основного применения:

Open[V]	открывает VIEW	
Next[V]	читает следующий элемент	
Previous[V]	читает предыдущий элемент	PrimeRecord подготавливает буфер записи при добавлении
	ValidateRecord[V]	проверяет текущий элемент
	SetFilter[V]	задает фильтр для активного порядка сортировки
	SetSort[V]	устанавливает активный порядок сортировки
	Close[V]	закрывает VIEW

Эпизодического применения:

	SetOrder[V]	заменяет активный порядок сортировки
	UseView	использует файлы с отложенным открытием

[V] Эти методы также являются виртуальными.

Виртуальные методы

Обычно эти методы не вызываются непосредственно - к ним обращаются методы первичного интерфейса. Однако мы предполагаем, что у вас возникнет необходимость переопределить эти методы, и, поскольку они являются виртуальными, сделать это несложно. В противном случае их стандартное поведение представляется разумным.

Open	открывает VIEW
Next	читает следующий элемент
Previous	читает предыдущий элемент
Reset	переустанавливает позицию VIEW
SetSort	устанавливает активный порядок сортировки
SetFilter	задает фильтр для активного порядка сортировки
SetOrder	аменяет активный порядок сортировки
ApplyFilter	ограничивает диапазон выборки и фильтрует ее
ApplyOrder	сортирует выборку
ApplyRange	ограничивает диапазон выборки и фильтрует ее
ValidateRecord	проверяет текущий элемент
GetFreeElementName	возвращает имя поля - свободногоэлемента ключа
GetFreeElementPosition	возвращает номер поля - свободного элемента ключа
Close	закрывает VIEW
Kill	завершает объект ViewManager

AddRange (добавляет диапазон)

AddRange(*поле* |[*,мин. знач.* [*,макс. знач.*]] |**)**
|*,первичный, родитель* |

AddRange	Задает границы диапазона для конкретного порядка сортировки
поле	Метка ограничиваемого поля. Поле не обязательно должно быть компонентом ключа или индекса, однако если оно является таковым, производительность VIEW существенно повышается
мин. знач.	Константа, переменная, EQUATE или выражение, задающая нижнюю границу диапазона значений *поля*. Если параметр опущен, *поле* ограничивается текущим значением.
макс. знач.	Константа, переменная, EQUATE или выражение, задающая верхнюю границу включающего диапазона значений *поля*. Нижняя граница включающего диапазона задается параметром *мин. знач.* Если параметр опущен, *поле* ограничивается значением, заданным параметром *мин. знач.*
первичный	Метка объекта RelationManager для первичного файла обрабатываемой VIEW. Таким образом, все поля связи ограничиваются значениями

соответствующих полей родительского файла.

родитель Метка объекта RelationManager для файла, являющегося родительским для первичного файла обрабатываемой VIEW. ViewManager использует этот объект при реляционном диапазоне, чтобы получить ограничивающие значения из родительского файла.

Метод **AddRange** задает границы диапазона для конкретного порядка сортировки. Этот диапазон может быть применен к VIEW, когда активным является соответствующий порядок сортировки. Когда ограничивающий диапазон "включен", в результирующую выборку попадают лишь те записи, в которых *поле* имеет значения, удовлетворяющие этому диапазону. Вы можете указать лишь по одному диапазону на каждый порядок сортировки.

Реализация: Метод AddSortOrder добавляет порядок сортировки. Метод ApplyRange "включает" диапазон активного порядка сортировки. Метод SetSort устанавливает активный порядок сортировки.

Метод AddRange игнорирует параметр *поле*, если присутствует паараметр *первичный*.

Пример:

```
MyView.AddSortOrder(ORD:ByCustomer)      !сортировка по номеру
                                         !покупателя
MyView.AddRange(ORD:CustNo,Relate:Orders,|
 Relate:Customer)                        !ограничить по родительскому
                                         !файлу
MyView.AddSortOrder(ORD:ByOrder)         !сортировка по номеру заказа
MyView.AddRange(ORD:OrderNo)             !ограничить по текущему
                                         !значению поля ORD:OrderNo
```

См. Также: AddSortOrder, ApplyRange, SetSort

AddSortOrder (добавляет порядок сортировки)

AddSortOrder([*ключ*] **), PROC**

AddSortOrder Задает порядок сортировки для объекта ViewManager.

ключ Метка ключа первичного файла. Если параметр опущен, принимается физический порядок записей.

Метод **AddSortOrder** задает порядок сортировки и возвращает число, идентифицирующее добавленный порядок сортировки.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Только один порядок сортировки активен в конкретный момент времени. Метод SetSort устанавливает активный порядок сортировки в терминах идентификатора, возвращаемого методом AddSortOrder.

Реализация: Вы можете задать несколько порядков сортировки, для чего следует несколько раз вызвать метод AddSortOrder. При первом вызове будет возвращена единица (1), при втором - два (2) и т.д.

Тип результата: BYTE

Пример:

```
CustSort = MyView.AddSortOrder(ORD:ByCustomer)      !сортировка по
                                                    !номеру покупателя

MyView.AddRange(ORD:CustNo,Relate:Orders,|
 Relate:Customer)                                    !ограничить по
                                                    !родительскому файлу

OrderSort =MyView.AddSortOrder(ORD:ByOrder)         !сортировка по
                                                    !номеру заказа

MyView.AddRange(ORD:OrderNo)                        !ограничить по текущему
                                                    !значению поля ORD:OrderNo
                                                    !код программы

IF MyView.SetSort(CustSort)                         !установить активный
                                                    !порядок  сортировки

DISPLAY                                             !если изменился -
                                                    !перепоказать

        END
```

См. Также: SetSort

AppendOrder (уточняет порядок сортировки)

AppendOrder(*выражение-список* **)**

AppendOrder	Уточняет порядок сортировки для объекта ViewManager
выражение-список	Строковая константа, переменная, EQUATE или выражение, содержащее список сортировки для атрибута ORDER Более подробно см. *ORDER* в *Описании языка*.

Метод **AppendOrder** уточняет или расширяет активный порядок сортировки.

Метод SetSort устанавливает активный порядок сортировки.

Реализация: ViewManager реализует сортировку при помощи атрибута ORDER структур VIEW. Метод AppendOrder добавляет *выражение-список* в конец выражения-списка активного порядка сортировки. Нет необходимости начинать

выражение-список с запятой или другого разделителя.

Пример:

```
MyView.AddSortOrder(ORD:ByCustomer)      !сортировка по  номеру
                                         !покупателя
MyView.AppendOrder('CUST:CustName')      !и по его имени
```

См. Также: AddSortOrder, SetSort

ApplyFilter (ограничивает диапазон и фильтрует выборку)

ApplyFilter, VIRTUAL

Метод **ApplyFilter** "включает" ограничивающий диапазон и фильтр для активного порядка сортировки обрабатываемой VIEW. Фильтрация вступает в силу при следующей операции чтения.

Методы AddSortOrder и SetSort устанавливают активный порядок сортировки. Метод SetFilter устанавливает выражение фильтра.

Метод ApplyFilter является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: ViewManager реализует диапазоны и фильтры при помощи атрибута FILTER структур VIEW. Более подробно см. *FILTER* в *Описании языка*.

Пример:

```
MyView.AddSortOrder(ORD:ByCustomer)          !сортировка по номеру
                                             !покупателя
MyView.AddRange(ORD:CustNo,Relate:Orders,|   !ограничить по
Relate:Customer)                             !родительскому файлу
MyView.SetFilter( '(CUST:Name>''T'')' )      !установить фильтр
                                             !по имени покупателя
                                             !код программы
    MyView.ApplyFilter                       !"включить фильтр"
  MyView.Next()                              !прочесть следующую запись,
                                             !удовлетворяющую условим
                                             !фильтра
```

См. Также: SetFilter, SetSort

ApplyOrder (сортирует выборку)

ApplyOrder, VIRTUAL

Метод **ApplyOrder** применяет активный порядок сортировки к обрабатываемой VIEW. Сортировка вступает в силу с момента следующей операции чтения VIEW.

Метод ApplyOrder устанавливает допустимые порядки сортировки. Метод SetSort устанавливает активный порядок сортировки.

Метод ApplyOrder является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: ViewManager реализует сортировку при помощи атрибута ORDER структур VIEW. Более подробно см. *ORDER* в *Описании языка*.

Пример:

```
MyView.AddSortOrder(ORD:ByCustomer)  !сортировка по номеру
                                     !покупателя
                                     !код программы
MyView.ApplyOrder                    !применить сортировку
MyView.Next()                        !прочесть следующую запись в
                                     !требуемой последовательности
```

См. Также: AddSortOrder, SetSort

ApplyRange (ограничивает диапазон и фильтрует выборку)

ApplyRange, VIRTUAL, PROC

Метод **ApplyRange** "включает" ограничивающий диапазон и вызывает метод ApplyFilter, если границы диапазона изменились. ApplyRange возвращает значение, показывающее, произошло ли изменение. Единичный (1) результат означает, что изменение было, нулевой (0) результат означает отсутствие изменений.

Метод AddRange задает границы диапазона для объекта ViewManager. Метод SetSort устанавливает активный порядок сортировки.

Метод ApplyRange является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно, его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод **ApplyRange** "включает" ограничивающий диапазон и фильтр при помощи метода ApplyFilter.

Тип результата: BYTE

Пример:

```
MyView.AddSortOrder(ORD:ByCustomer)          !сортировка по номеру
                                             !покупателя
MyView.AddRange(ORD:CustNo,Relate:Orders,|   !ограничить по
 Relate:Customer)                            !родительскоу файлу
                                             !код программы
        MyView.ApplyRange                    "включить" диапазон
        MyView.Next()
                                             !прочесть следующую запись в
                                             !диапазоне
```

См. Также: AddRange, ApplyFilter, SetSort

Close (закрывает VIEW)

Close, VIRTUAL

Метод **Close** закрывает обрабатываемую VIEW.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Пример:

```
MyView.AddSortOrder(ORD:ByCustomer)          !сортировка по номеру
                                             !покупателя
MyView.AddRange(ORD:CustNo,Relate:Orders,|   !ограничить по
 Relate:Customer)                            !родительскоу файлу
MyView.Open                                  !открыть VIEW
                                             !код программы
        MyView.Close                         !закрыть VIEW
```

GetFreeElementName (возвращает имя свободного элемента ключа)

GetFreeElementName

Метод **GetFreeElementName** возвращает полное имя первого поля активной сортировки, которое не ограничено единственным значением при текущем ограничивающем диапазоне. Например, рассмотрим VIEW, отсортированную по полям Customer, Order и Item, причем поле Customer ограничено текущим

значением. Свободным элементом в данном случае является Order. Но если "отключить" ограничения по диапазону, свободным элементом станет поле Customer.

Метод AddSortOrder устанавливает ключ / порядок сортировки для VIEW. Метод SetSort устанавливает активную сортировку. Метод AddRange добавляет ограничивающий диапазон.

Реализация: Класс FilterLocatorClass использует метод GetFreeElementName для обновления окна.

Тип результата: STRING

Пример:

```
BuildFilter PROCEDURE(STRING filter)
FieldName CSTRING(100)
          CODE
FieldName = MyView.GetFreeElementName()          !получить фильтруемое
                                                 !поле
MyView.SetFilter(FieldName&'[1] = '''&filter[1]&'''')  !установить
                                                 !выражение фильтра
MyView.ApplyFilter()                             !"включить" фильтр
```

См. Также: AddRange, AddSortOrder, SetSort

GetFreeElementPosition (возвращает номер свободного элемента ключа)

GetFreeElementPosition, PROTECTED, VIRTUAL

Метод **GetFreeElementPosition** возвращает номер первого поля активной сортировки, которое не ограничено единственным значением при текущем ограничивающем диапазоне. Например, рассмотрим VIEW, отсортированную по полям Customer, Order и Item, причем поле Customer ограничено текущим значением. Свободным элементом в данном случае является Order. Но если "отключить" ограничения по диапазону, свободным элементом станет поле Customer.

Метод AddSortOrder устанавливает ключ / порядок сортировки для VIEW. Метод SetSort устанавливает активную сортировку. Метод AddRange добавляет ограничивающий диапазон.

Этот метод имеет атрибут PROTECTED, следовательно, может быть вызван только из методов класса ViewManager или порожденного от него класса.

Метод GetFreeElementPosition является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод BrowseClass.TakeKey использует метод GetFreeElementPosition для репозиционирования VIEW, базирующейся на фиксированных элементах ключа. Метод GetFreeElementName использует GetFreeElementPosition для определения имени свободного элемента.

Тип результата: BYTE

Пример:

```
BrowseClass.TakeKey  PROCEDURE    !код метода
IF SELF.Sort.Locator.TakeKey()
        Handled = 1
        SELF.Reset(SELF.GetFreeElementPosition())
        SELF.ResetQueue(Reset:Done)
        ELSE
        SELF.ListControl{PROP:SelStart} = SELF.CurrentChoice
        END
```

См. Также: GetFreeElementName, BrowseClass.TakeKey

Init (инициализирует объект ViewManager)

Init(*view, первичный* [, *сортировка*] **)**

Init	Инициализирует объект ViewManager.
view	Метка обрабатываемой структуры VIEW.
первичный	Метка объекта RelationManager для первичного файла *view*.
сортировка	Структура, содержащая информацию о сортировке, диапазоне и фильтре для обрабатываемой VIEW. Если параметр опущен, метод Init создает пустую структуру SortOrder, которую можно настроить при помощи методов AddSortOrder, AppendOrder, SetOrder, AddRange и SetFilter.

Метод **Init** инициализирует объект ViewManager.

Реализация: Метод Init устанавливает значения свойств Order, PagesAhead, PagesBehind, PageSize, Primary и View.

Параметр *сортировка* позволяет порожденным классам, таким как BrowseClass, добавлять дополнительную информацию о сортировках своих структур VIEW.

Передавая в качестве параметра *сортировка* свойство Order из другого объекта ViewManager или свойство Sort из объекта BrowseClass, вы можете создать несколько

объектов со сходными сортировками, диапазонами и фильтрами.

Пример:

```
MyView.Init(OrderView,Relate:Order)!инициализировать ViewManager
MyView.Open                                !открыть VIEW
                                           !код программы
MyView.Close                               !закрыть VIEW
MyView.Kill                                !завершить ViewManager
```

См. Также: Order, Primary, View, PagesAhead, PagesBehind, PageSize

Kill (shut down the ViewManager object)

Kill, VIRTUAL

Метод **Kill** завершает объект ViewManager, освобождая всю память, выделенную в течение жизни объекта и выполняя любой необходимый код завершения.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Пример:

```
MyView.Init(OrderView,Relate:Order)        !инициализировать ViewManager
MyView.AddSortOrder(ORD:ByCustomer)        !сортировка по номеру
                                           !покупателя
MyView.AddRange(ORD:CustNo,Relate:Orders,| !ограничить по
Relate:Customer)                           !родительскому файлу
MyView.Open                                !открыть VIEW
                                           !код программы
MyView.Close                               !закрыть VIEW
MyView.Kill                                !завершить ViewManager
```

Next (читает следующий элемент)

Next, VIRTUAL

Метод **Next** читает следующий элемент VIEW в соответствии с текущей сортировкой, диапазоном и фильтром, затем возвращает результат, показывающий, успешно ли завершилась операция.

Если Next завершился успешно, возвращается Level:Benign (объявлен в файле ABERROR.INC). Если завершение было неудачным, возвращается Level:Notify или Level:Fatal, в зависимости от произошедшей ошибки. Более подробно об уровнях серьезности ошибок см. *ErrorClass - обработчик ошибок*.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод Next использует метод ValidateRecord для проверки неотфильтрованных записей.

Тип результата: BYTE

Пример:

```
CASE MyView.Next()                      !попробуем прочесть следующую запись
OF Level:Benign                         !контроль ошибок
                                        !обработка записи
     OF Level:Notify                    !неудача
                                        !протоколирование ошибки
     OF Level:Fatal                     !фатальная ошибка
     POST(Event:CloseWindow)
     BREAK
     END
```

См. Также: ValidateRecord

Open (открывает VIEW)

Open, VIRTUAL

Метод **Open** открывает обрабатываемую VIEW.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод Open открывает VIEW *и* "включает" активную сортировку и фильтр при помощи методов ApplyOrder и ApplyFilter, а также буферирование, заданное свойствами PagesAhead, PagesBehind, PageSize и TimeOut.

Пример:

```
MyView.AddSortOrder(ORD:ByCustomer)          !сортировка по номеру
                                             !покупателя
MyView.AddRange(ORD:CustNo,Relate:Orders,|   !ограничить по
Relate:Customer)                             !родительскому файлу
MyView.Open                                  !открыть VIEW
                                             !код программы
     MyView.Close                            !закрыть VIEW
```

См. Также: ApplyFilter, ApplyOrder, PagesAhead, PagesBehind, PageSize, TimeOut

Previous (читает предыдущий элемент)

Previous, VIRTUAL

Метод **Previous** читает предыдущий элемент VIEW в соответствии с текущей сортировкой, диапазоном и фильтром, затем возвращает результат, показывающий, успешно ли завершилась операция.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Если Previous завершился успешно, возвращается Level:Benign (объявлен в файле ABERROR.INC). Если завершение было неудачным, возвращается Level:Notify или Level:Fatal, в зависимости от произошедшей ошибки. Более подробно об уровнях серьезности ошибок см. *ErrorClass - обработчик ошибок*.

Метод Previous использует метод ValidateRecord для проверки неотфильтрованных записей.

Тип результата: BYTE

Пример:

```
CASE MyView.Previous()                  !попробуем прочесть предыдущую запись
        OF Level:Benign                 !контроль ошибок
                                        !обработка записи
        OF Level:Notify                 !неудача
                                        !протоколирование ошибки
        OF Level:Fatal                  !фатальная ошибка
        POST(Event:CloseWindow)
        BREAK
        END
```

См. Также: ValidateRecord

PrimeRecord (готовит запись для добавления)

PrimeRecord([*не очищать*] **), VIRTUAL**

PrimeRecord Подготавливает буфер записи первичного файла VIEW к добавлению.

не очищать Целочисленная константа, переменная, EQUATE или выражение,

указывающая, очищать ли буфер записи. Нулевое значение (0 или False) очищает буфер, единичное (1 или True) - не очищает. Если параметр опущен, его значение принимается равным нулю (0).

Метод **PrimeRecord** заполняет поля буфера записи первичного файла VIEW начальными значениями перед добавлением новой записи.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Пример:

```
CASE FIELD()
OF ?InsertButton
CASE EVENT()
OF EVENT:Accepted                    !если нажата кнопка добавления
MyView.PrimeRecord                   !подготовить запись
                                     !добавить новую запись
        END
        END
```

См. Также: FileManager.PrimeRecord

Reset (репозиционирует VIEW)

Reset([*количество*] **), VIRTUAL**

Reset Репозиционирует VIEW.

количество Целочисленная константа, переменная, EQUATE или выражение, задающее начальную позицию на основании значений указанного *количества* старших компонентов текущей сортировки (атрибута ORDER). Если параметр опущен, VIEW позиционируется на первый элемент выборки.

Метод **Reset** переустанавливает позицию VIEW на начало результирующей выборки, заданной активным порядком сортировки, диапазоном и фильтром. Параметр *количество* служит для уточнения позиции по значениям указанного *количества* выражений в активной сортировке.

Например, рассмотрим VIEW, отсортированную по полю Customer, где значение этого поля равно десяти (10). Если *количество* опущено, Reset позиционируется на элемент с наименьшим значением поля Customer, независимо от его значения. Однако, если *количество* равно единице(1), Reset позиционируется на первый элемент, у которого значение поля равно десяти (10).

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод Reset вызвает метод Open и выполняет оператор SET для обрабатываемой VIEW. Более подробно см. *Описание языка - SET*.

Пример:

```
View:Customer.Init(Customer:View,Relate:CUSTOMER)
                                    !инициализировать объект View:Customer
View:Customer.AddSortOrder( CUS:BYNUMBER )
                                    !добавить сортировку BYNUMBER
View:Customer.AddRange(CUS:CUSTNO,Low,High)
                                    !добавить диапазон
View:Customer.SetFilter( 'CUS:ZIP=33064','1')
                                    !добавить фильтр номер 1
Relate:CUSTOMER.Open                !открыть файл Customer и связанные с ним
View:Customer.Reset                 !открыть VIEW с диапазоном и фильтром
IF View:Customer.Next()             !прочесть первую запись
        HALT                        !если записей нет - стоп
        END
```

См. Также: Open

SetFilter (добавляет, изменяет или удаляет активный фильтр)

SetFilter(*выражение* [, *идентификатор*] **), VIRTUAL**

SetFilter Задает фильтр для активной сортировки.

выражение Строковая константа, переменная, EQUATE или выражение, содержащая выражение фильтра. Более подробно см. *FILTER* в *Описании языка*. Если *выражение* пусто (''), SetFilter удаляет все существующие фильтры с соответствующим *идентификатором*.

идентификатор Строковая константа, переменная, EQUATE или выражение, которая уникально идентифицирует фильтр (и задает его приоритет). Таким образом, вы можете использовать несколько условий фильтрации, а также заменять или удалять условия фильтрации при помощи последовательных вызовов метода SetFilter. Если параметр опуущен, фильтр приобретает идентификатор по умолчанию, так что последующие вызовы SetFilter без *идентификатора* заменяют *выражения* фильтра, установленные предыдущими вызовами SetFilter без *идентификатора*.

Метод **SetFilter** задет фильтр для активного порядка сортировки. Когда фильтр вступает в силу, в результирующую выборку попадают только те элементы, для которых *выражение* вычисляется в истинное значение.

Параметр *идентификатор* позволяет вам задать несколько *выражений* фильтра или заменить *выражение* с определенным *идентификатором*. Если вы задали несколько *выражений*, каждое со своим уникальным идентификатором, все эти *выражения* должны вычисляться в истинное значение для того, чтобы элемент был включен в результирующую выборку.

ViewManager вычисляет *выражения* в порядке их идентификаторов, поэтому с точки зрения эффективности целесообразно задать *выражениям* приоритеты таким образом, чтобы в начале находились *выражения* с наиболее трудновыполнимыми условиями, например:

```
MyView.SetFilter('TaxPayer=True','9Tax')
                                    !выражение с низким приоритетом
MyView.SetFilter('LotteryWinner=True','1Lot')
                                    !выражение с высоким приоритетом
                                    !эквивалентно (LotteryWinner=True) AND
                                    (TaxPayer=True)
```

Методы ApplyFilter и ApplyRange "включают" фильтрацию для активной сортировки. Метод SetSort устанавливает активную сортировку.

Метод SetFilter является виртуальным, следовательно другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Пример:

```
MyView.AddSortOrder(ORD:ByOrder)      !сортировка по номеру заказа (1)
MyView.SetFilter('(ORD:OrdNo=CUST:OrdNo)','1OrderNo')
                                    !фильтр по номеру заказа
MyView.SetFilter('(ORD:Date='&TODAY()&')','1Date')
                                    !и по дате.
        !Вначале проверяется дата, т.к. идентификатор этого      фильтра
        !меньше идентификатора фильтра по номеру заказа
MyView.AddSortOrder(ORD:ByName)
                                    !сортировка по имени покупателя (2)
MyView.SetFilter('CUST:Name[1]=''A''')
                                    !фильтр по имени покупателя
                                    !код программы
MyView.SetSort(2)                   !сортируем по имени
MyView.SetFilter('CUST:Name[1]=''J''')  !новый фильтр по имени
                                    !заменяет старый
```

См. Также: AddSortOrder, Order

SetOrder (заменяет порядок сортировки)

SetOrder(*выражение-список* **), VIRTUAL**

SetOrder	Заменяет активную сортировку.
выражение-список	Строковая константа, переменная, EQUATE или выражение, содержащая выражение-список для атрибута ORDER. Более подробно см. *Описание языка - ORDER*.

Метод **SetOrder** заменяет активную сортировку для объекта ViewManager.

Метод SetSort устанавливает активную сортировку.

Метод SetOrder является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: ViewManager реализует сортировки при помощи атрибута ORDER структур VIEW. Метод SetOrder заменяет выражение-список активной сортировки на *выражение-список*, заданное параметром.

Пример:

```
MyView.AddSortOrder(ORD:ByCustomer)  !сортировка по номеру
                                     !program code
MyView.SetOrder(CUST:CustName)       !ссортировка по имени
```

См. Также: SetSort

SetSort (устанавливает активный порядок сортировки)

SetSort(*номер сортировки* **), VIRTUAL**

SetSort	Устанавливает активный порядок сортировки.
номер сортировки	Целочисленная константа, переменная, EQUATE или выражение, указывающая, какой порядок сортировки должен быть использован. Поорядки сортировки пронумерованы в той последовательности, в которой они добавлялись методом AddSortOrder.

Метод **SetSort** устанавливает активный порядок сортировки и возвращает результат, показывающий, изменился ли активный порядок сортировки (*номер сортировки*).

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: SetSort возвращает единицу (1), если *номер сортировки*

изменился и ноль (0) в противном случае.

Тип результата: BYTE

Пример:

```
CustSort = MyView.AddSortOrder(ORD:ByCustomer)
                                        !сортировка по номеру покупателя
MyView.AddRange(ORD:CustNo,Relate:Orders,Relate:Customer)
                                        !ограничение по родительскому файлу
OrderSort = MyView.AddSortOrder(ORD:ByOrder)
                                        !сортировка по номеру заказа
MyView.AddRange(ORD:OrderNo)            !ограничение диапазона
                                        !по текущему значению
ORD:OrderNo                             !код программы
IF MyView.SetSort(CustSort)             !установить активную сортировку
MESSAGE('New Sort Order')               !оповестить о новой сортировке
        END
```

См. Также: AddSortOrder

UseView (использует файлы с отложенным открытием)

UseView, PROTECTED

Метод **UseView** сообщает объектам библиотеки ABC, что файлы обрабатываемой структуры VIEW, открытие которых отложено свойством LazyOpen, будут использоваться в ближайшее время.

Этот метод имеет атрибут PROTECTED, следовательно, может быть вызван только из методов класса ViewManager или порожденного от него класса.

Реализация: Методы Init и Open вызывают метод UseView, который, в свою очередь, вызовет метод FileManager.UseFile для каждого файла в обрабатываемой VIEW.

См. Также: Init, Open, FileManager.LazyOpen, FileManager.UseFile

ValidateRecord (проверяет элемент)

ValidateRecord, VIRTUAL

Метод **ValidateRecord** проверяет текущий элемент VIEW и возвращает результат, показывающий, верны ли данные. Нулевой (0) результат говорит о том, что данные верны, любое другое значение свидетельствует о том, что элемент содержит недопустимые данные.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу в порожденных классах. Это позволяет вам легко реализовать собственные варианты

данного метода.

Реализация: ValidateRecord является "виртуальным служащим" для методов порожденных классов.

Методы Next и Previous вызывают метод ValidateRecord.

Возвращаемые значения объявлены в ABFILE.INC следующим образом:

```
 ITEMIZE(0),PRE(Record)
OK          EQUATE                  !Запись удовлетворяет
                                    !ограничениям диапазона и
                                    !условиям фильтра
OutOfRange  EQUATE                  !Запись не удовлетворяет
                                    !ограничениям диапазона
Filtered    EQUATE                  !Запись не удовлетворяет
                                    !условиям фильтра
 END
```

Тип результата: BYTE

Пример:

```
ViewManager.Next PROCEDURE
 CODE
            LOOP
            NEXT(SELF.View)
            IF ERRORCODE()
            IF ERRORCODE() = BadRecErr
            RETURN Level:Notify
            ELSE
SELF.Primary.Me.Throw(Msg:AbortReading)
            RETURN Level:Fatal
            END
            ELSE
            CASE SELF.ValidateRecord()
            OF Record:OK
            RETURN Level:Benign
            OF Record:OutOfRange
            RETURN Level:Notify
            END
            END
            END
```

См. Также: Next, Previous

Классы Browse

Обзор

Концепции BrowseClass

BrowseClass представляет собой ViewManager с пользовательским интерфейсом для навигации по результирующей выборке соответствующей VIEW-структуры.

BrowseClass использует несколько связанных классов, что обеспечивает стандартную функциональность Browse – это постраничное заполнение списков с автоматическим скроллингом, поиск, выделение диапазона записей, фильтрацию, сброс, условные цвета, условные пиктограммы и т.д. (эти классы могут также использоваться для реализации других компонентов и функциональных возможностей).

В дополнение к стандартному набору функций существует редактирование по месту, то есть обновление первичного файла может быть выполнено непосредственным вводом в список browse. Для этого не требуется отдельной модифицирующей процедуры. Изменения выполняются с учетом необходимых автоинкрементов полей, поддержки ссылочной целостности и проверки значений полей.

Ниже приведены классы, обеспечивающие функциональность browse, с указанием их назначения:

BrowseClass	Класс “администратор” Browse
StepClass	Базовый класс индикаторов прокрутки/прогресса
LongStepClass	Целочисленное распределение на этапе выполнения
RealStepClass	Численное распределение на этапе выполнения
StringStepClass	Символьное временное распределение
CustomStepClass	Настраиваемое специальное распределение
LocatorClass	Базовый класс локатора
StepLocatorClass	Пошаговый локатор
EntryLocatorClass	Вводной локатор
IncrementalLocatorClass	Инкрементный (наращиваемый) локатор
FilterLocatorClass	Фильтрующий локатор
EditClass	Редактирование по месту

Класс BrowseClass полностью описан в оставшейся части данной главы. См. *Step Classes* и *Locator Classes* для более полной информации об этих классах.

Концепции класса EditClass

Класс EditClass дает Вам возможность динамического редактирования по месту каждого столбца (поля) первичного файла в browse. При этом поддерживаются автоинкрементные ключи, диапазоны и ссылочная целостность. В настоящее время проверка значений полей выполняется при сохранении записи, и изменен может быть только первичный файл.

Метод BrowseClass.AskRecord – инструмент для выполнения функции редактирования по месту. Этот метод динамически создает экранные элементы управления по запросу (добавление, изменение или удаление записи) конечного пользователя. Когда пользователь покидает отредактированную запись (нажатием клавиши или щелчком мышью на другом элементе окна), этот метод сохраняет или удаляет запись и ликвидирует экранный созданный ранее элемент управления.

Для осуществления редактирования по месту вам необходимо лишь вызвать метод BrowseClass.Ask. Никаких прямых ссылок на класс EditClass не требуется.

Взаимоотношение с другими ABC-классами

Класс BrowseClass тесно взаимосвязан с некоторыми другими объектами библиотеки ABC – в частности, с объектами WindowManager ? ToolbarClass. Эти объекты регистрируют присутствие друг друга, устанавливают значения свойств друг друга и при необходимости вызывают методы друг друга для достижения своих целей.

Класс BrowseClass порождается от класса ViewManager, а также опирается на другие ABC-классы (RelationManager, FieldPairsClass, ToolbarClass, PopupClass и т.д.) для выполнения своих задач. Таким образом, если в вашей программе объявлены объекты BrowseClass, в ней также должны быть объявлены и эти другие классы. В значительной степени это делается автоматически, когда вы включаете (INCLUDE) заголовок класса BrowseClass (ABBROWSE.INC) в раздел данных своей программы. См. *Концептуальный пример*.

Реализация ABC -шаблонов

ABC-шаблоны автоматически включают все классы и генерируют весь код, необходимый для поддержки функциональности, указанной в шаблонах Browse Procedure ? BrowseBox Control вашего приложения.

Шаблоны *порождают* класс от BrowseClass и создают объект для *каждого* шаблонного элемента управления BrowseBox в приложении. Порожденный класс называется Browse*Filename*:BRW#, где # - номер экземпляра шаблона элемента управления BrowseBox. Этот объект порожденного класса поддерживает всю функциональность, заданную в шаблоне BrowseBox.

Порожденный класс является локальным по отношению к процедуре, специфическим для данного элемента управления BrowseBox и опирается на глобальные объекты RelationManager и FileManager для показываемого файла.

Редактирование по месту

Один из переключателей в окне шаблона элемента управления BrowseUpdateButtons разрешает поддержку редактирования по месту для данного browse – любая связанная форма обновления теперь становится совершенно излишней.

Генерируемый код для процедуры Edit-in-place Browse лишь немного отличается от традиционного Browse с процедурой Form, поэтому Вы можете отложить выбор методов модифицирования на последнюю минуту.

Исходные модули класса Browse Class

Исходные модули BrowseClass по умолчанию устанавливаются в каталог \CLARION4\LIBSRC. Ниже перечислены файлы BrowseClass и соответствующие компоненты:

ABBROWSE.INC	Объявления класса BrowseClass Объявления класса EditClass
ABBROWSE.CLW	Определение методов класса BrowseClass Определение методов класса EditClass
ABBROWSE.TRN	Перевод строковых констант, относящихся к Browse

Концептуальный пример

Следующий пример иллюстрирует типичную последовательность операторов, объявляющих, активизирующих, инициализирующих, использующих и завершающих объект BrowseClass и связанные объекты. Пример инициализирует и постранично загружает список LIST, затем обрабатывает ряд событий, включая поиск, скроллинг и обновление. Когда объекты BrowseClass и WindowManager должным образом инициализированы, они выполняют большинство действий (стандартную обработку событий) самостоятельно.

```
PROGRAM
 INCLUDE('ABWINDOW.INC')          !объявить класс WindowManager
INCLUDE('ABBROWSE.INC')           !объявить класс BrowseClass
          MAP
          END

State
FILE,DRIVER('TOPSPEED'),PRE(ST),THREAD
StateCodeKey KEY(ST:STATECODE),NOCASE,OPT
Record    RECORD,PRE()
STATECODE STRING(2)
STATENAME STRING(20)
END
END
StView    VIEW(State)             !объявить обрабатываемую VIEW
END
StateQ    QUEUE                   !объявить очередь просмотра
ST:STATECODE LIKE(ST:STATECODE)
ST:STATENAME LIKE(ST:STATENAME)
ViewPosition  STRING(512)
          END
GlobalErrors ErrorClass           !объявить объект GlobalErrors
Access:State CLASS(FileManager)   !объявить объект Access:State
Init PROCEDURE
END
Relate:State CLASS(RelationManager)
!объявить объект Relate:State
Init      PROCEDURE
END
VCRRequest  LONG(0),THREAD
StWindow  WINDOW('Browse States'),AT(,,123,152),IMM,SYSTEM,GRAY
LIST,AT(8,5,108,124),USE(?StList),IMM,HVSCROLL,FROM(StateQ),|
FORMAT('27L(2)|M~CODE~@s2@80L(2)|M~STATENAME~@s20@')
BUTTON('&Insert'),AT(8,133),USE(?Insert)
BUTTON('&Change'),AT(43,133),USE(?Change),DEFAULT
BUTTON('&Delete'),AT(83,133),USE(?Delete)
END
ThisWindow  CLASS(WindowManager)  !объявить объект ThisWindow
Init
PROCEDURE(),BYTE,PROC,VIRTUAL
Kill
PROCEDURE(),BYTE,PROC,VIRTUAL
END
```

```
BrowseSt  CLASS(BrowseClass)                    !объявить объект BrowseSt
Q         &StateQ
END
StLocator StepLocatorClass                      !объявить объект StLocator
StStep    StepStringClass                       !объявить объект StStep
          CODE
          ThisWindow.Run()                      !запуск процедуры обработки окна
ThisWindow.Init PROCEDURE()                     !инициализация
ReturnValue  BYTE,AUTO
          CODE
ReturnValue = PARENT.Init()                     !вызов метода инициализации
                                                !базового     класса
IF ReturnValue THEN RETURN ReturnValue.
GlobalErrors.Init                               !инициализация объекта
                                                !GlobalErrors
Relate:State.Init                               !инициализация объекта Relate:State
SELF.FirstField = ?StList                       !установить FirstField для ThisWindow
SELF.VCRRequest &= VCRRequest                   !VCRRequest не используется
SELF.Errors &= GlobalErrors                     !установить обработчик ошибок для
                                                !ThisWindow
Relate:State.Open                               !открыть State и связанные файлы
                                                !Инициализировать объект BrowseSt с
                                                !указанием его
                                                !LIST,VIEW,Q,RelationManager &
                                                !WindowManager
BrowseSt.Init(?StList,StateQ.ViewPosition,StView,StateQ,Relate:State,SELF)
          OPEN(StWindow)
          SELF.Opened=True
          BrowseSt.Q &= StateQ                  !установить ссылку на очередь просмотра
StStep.Init(+ScrollSort:AllowAlpha,ScrollBy:Runtime)! инициализация объекта StStep
BrowseSt.AddSortOrder(StStep,ST:StateCodeKey)
                                                !установить порядок сортировки
BrowseSt.AddLocator(StLocator)                  !включить локатор
StLocator.Init(,ST:STATECODE,1,BrowseSt)
                                                !инициализировать локатор
BrowseSt.AddField(ST:STATECODE,BrowseSt.Q.ST:STATECODE)
                                                !добавить поле просмотра
BrowseSt.AddField(ST:STATENAME,BrowseSt.Q.ST:STATENAME)
                                                ! добавить поле просмотра
BrowseSt.InsertControl=?Insert                  !установить элемент управления для
                                                !добавления записей
BrowseSt.ChangeControl=?Change                  !установить элемент управления для
                                                !изменения записей
```

```
BrowseSt.DeleteControl=?Delete          !установить элемент управления для удаления
                                        !записей
SELF.SetAlerts()                        !назначить горячие клавиши для ThisWindow
RETURN ReturnValue
ThisWindow.Kill  PROCEDURE()            !завершение
ReturnValue  BYTE,AUTO
             CODE
ReturnValue = PARENT.Kill()             !вызов завершающего метода базового
                                        !класса
IF ReturnValue THEN RETURN ReturnValue.
Relate:State.Close                      !закрыть State и связанные файлы
Relate:State.Kill                       !завершить объект Relate:State
GlobalErrors.Kill                       !завершить объект GlobalErrors
RETURN ReturnValue
Access:State.Init  PROCEDURE
CODE
PARENT.Init(State,GlobalErrors)
SELF.FileNameValue = 'State'
SELF.Buffer &= ST:Record
SELF.AddKey(ST:StateCodeKey,'ST:StateCodeKey',0)
Relate:State.Init  PROCEDURE
CODE
Access:State.Init
PARENT.Init(Access:State,1)
```

Свойства BrowseClass

Класс BrowseClass наследует все свойства класса ViewManager, от которого он порожден. Более подробно см. *Свойства ViewManager*.

Кроме унаследованных свойств, BrowseClass содержит свойства, перечисленные ниже.

ActiveInvisible (поведение скрытого списка browse)

ActiveInvisible	BYTE

Свойство **ActiveInvisible** показывает, следует ли заполнять или обновлять очередь browse, когда список LIST невидим, поскольку находится на неактивной закладке (TAB) или скрыт иным способом. Единичное значение (1) обновляет очередь, когда список LIST невидим, нулевое значение (0) запрещает обновление.

Установка ActiveInvisible в нулевое значение (0) величивает производительность процедур с "невидимыми" списками; однако, не следует полагаться на содержимое буфера, т. к. для невидимых списков оно может не соответствовать текущему состоянию файла.

Реализация: Метод ResetQueue ведет себя в соответствии со значением свойства ActiveInvisible.

См. также: ResetQueue

AllowUnfilled (отображение заполненного списка)

AllowUnfilled BYTE

Свойство **AllowUnfilled** указывает, всегда ли отображать "полный" список или разрешать изображение частично заполненного списка, когда результирующая выборка "заканчивается" посередине списка. Единичное значение (1) отображает частично заполненный список и увеличивает производительность за счет подавления дополнительного чтения, требующегося для заполнения списка, нулевое значение (0) всегда отображает заполненный список.

Установка в единичное значение (1) улучшает производительность для списков browse, особенно для тех, которые используют данные SQL.

Реализация: Метод ResetQueue ведет себя в соответствии со значением свойства AllowUnfilled.

См. также: ResetQueue

ArrowAction (действие при редактировании по месту при нажатии клавишей со стрелками)

ArrowAction BYTE

Свойство **ArrowAction** указывает действие, которое будет выполняться, когда пользователь нажимает клавиши "стрелка вверх" или "стрелка вниз" во время выполнения редактирования по месту. Далее представлены три типа действий, которыми управляет ArrowAction:

√ что делать с любыми изменениями (стандартное действие, сохранение, отмена или запрос пользователя)

√ какой режим использовать далее (продолжение редактирования или возврат к режиму просмотра записей)

√ какой столбец редактировать следующим (текущий столбец или первый

доступный для редактирования столбец)

Указанные действия осуществляются методом Ask. Выбор действий следует осуществлять присваиванием свойству ArrowAction комбинаций следующих констант EQUATE. Эти значения EQUATE находятся в файле ABBROWSE.INC:

```
 ITEMIZE,PRE(EIPAction)
Default     EQUATE(0)          !сохранять в соответствии с методом Ask
Always      EQUATE(1)          !всегда сохранять изменения
Never       EQUATE(2)          !никогда не сохранять изменения
Prompted    EQUATE(4)          !запрашивать о сохранении изменений
Remain      EQUATE(8)          !продолжать редактирование
RetainColumn  EQUATE(16)       !сохранять позицию столбца в новой строке
            END
```

Пример:

```
BRW1.ArrowAction = EIPAction:Prompted
                                   !предложить сохранить изменения
BRW1.ArrowAction = EIPAction:Prompted+EIPAction:Remain
                                   !предложить сохранить изменения,
                                   !продолжить редактирование
                                   !первого допустимого для
                                   !редактирования столбца
BRW1.ArrowAction = EIPAction:Remain+EIPAction:RetainColumn
                                   !сохранить по умолчанию,
                                   !продолжить редактирование
                                   !того же столбца
```

См. также: Ask

AskProcedure (процедура обновления)

AskProcedure USHORT

Свойство **AskProcedure** идентифицирует процедуру обновления элемента данных browse. Нулевое значение (0) использует для изменений метод AskRecord объекта класса BrowseClass. Любое другое значение использует отдельную процедуру, зарегистрированную с объектом WindowManager.

Реализация: Обычно объект WindowManager (метод Init) устанавливает должным образом значение свойства AskProcedure, когда необходима отдельная модифицирующая процедура. Метод Ask передает значение AskProcedure в WindowManager. Метод Run указывает, какая процедура должна быть выполнена.

См. также: Ask, AskRecord, WindowManager.Run

ChangeControl (номер элемента управления изменением/обновлением)

ChangeControl SIGNED

Свойство **ChangeControl** содержит номер элемента управления изменением/модификацией. Обычно он приравнивается метке соответствия поля для кнопки Change. Методы BrowseClass используют это значение для разрешения и запрещения элемента управления в соответствующих случаях, при передаче событий элементу управления, при отражении режима изменения в соответствующем контекстном меню и т.д.

Реализация: Метод BrowseClass.Init не инициализирует свойства ChangeControl. Вы должны инициализировать эти свойства после вызова метода BrowseClass.Init. См. *Концептуальный пример.*

См. также: UpdateToolbarButtons

DeleteControl (номер элемента управления удалением)

DeleteControl SIGNED

Свойство **DeleteControl** содержит номер элемента управления удалением. Обычно он приравнивается метке соответствия поля для кнопки Delete. Методы BrowseClass используют это значение для разрешения и запрещения элемента управления в соответствующих случаях, при передаче событий элементу управления, при отражении режима удаления в соответствующем контекстном меню и т.д.

Реализация: Метод BrowseClass.Init не инициализирует свойства DeleteControl. Вы должны инициализировать эти свойства после вызова метода BrowseClass.Init. См. *Концептуальный пример.*

См. также: UpdateToolbarButtons

EditList (список элементов управления редактированием по месту)

EditList &BrowseEditQueue, PROTECTED

Свойство **EditList** представляет собой ссылку на структуру, содержащую список классов редактирования по месту для работы с списком столбцов конкретного browse.

Метод AddEditControl добавляет новые классы редактирования по месту и соответствующий им список столбцов в свойство EditList.

Это свойство имеет атрибут PROTECTED, следовательно обращаться к нему можно только из методов класса BrowseClass или порожденного от него класса.

Реализация: Нет необходимости инициализировать это свойство для реализации стандартных элементов редактирования по месту. Свойство EditList поддерживает индивидуальные элементы редактирования по месту.

Свойство EditList является ссылкой на QUEUE, объявленной в файле ABBROWSE.INC таким образом:

```
BrowseEditQueue  QUEUE,TYPE
Field        UNSIGNED
FreeUp       BYTE
Control      &EditClass
END
```

См. также: AddEditControl

EnterAction (действия при редактировании по месту по нажатию клавиши Enter)

EnterAction BYTE

Свойство **EnterAction** указывает, какое действие следует выполнить, когда конечный пользователь нажимает клавишу ENTER в процессе редактирования по месту. Существует два типа действий, управляемых свойством EnterAction:

- что делать с любыми изменениями (стандартное действие, сохранение, отмена или запрос пользователя)
- какой режим использовать далее (продолжение редактирования или возврат к режиму просмотра записей)

Указанные действия реализуются методом Ask. Выбор действий следует осуществлять присваиванием свойству EnterAction комбинаций следующих констант EQUATE. Эти значения EQUATE находятся в ABBROWSE.INC:

```
ITEMIZE,PRE(EIPAction)
Default    EQUATE(0)      !сохранять в соответствии с методом Ask
Always     EQUATE(1)      !всегда сохранять изменения
Never      EQUATE(2)      !никогда не сохранять изменения
Prompted   EQUATE(4)      !запрашивать о сохранении изменений
Remain     EQUATE(8)      !продолжать редактирование
END
```

Пример:

```
BRW1.EnterAction = EIPAction:Prompted
                                    !предложить сохранить изменения
BRW1.EnterAction = EIPAction:Prompted+EIPAction:Remain
                                    !предложить сохранить изменения,
                                    !продолжить редактирование
```

См. также: Ask

FocusLossAction (действия редактирования по месту при потере фокуса)

FocusLossAction BYTE

Свойство показывает какие действия следует выполнить в зависимости от того, какие изменения не сохранены, когда редактируемое поле теряет фокус в процессе редактирования по месту.

Указанные действия осуществляются методом Ask. Выбор действий следует осуществлять присваиванием свойству FocusLossAction комбинаций следующих констант EQUATE. Эти значения EQUATE находятся в ABBROWSE.INC:

```
ITEMIZE,PRE(EIPAction)
Default   EQUATE(0)     !сохранять в соответствии с методом Ask
Always    EQUATE(1)     !всегда сохранять изменения
Never     EQUATE(2)     !никогда не сохранять изменения
Prompted  EQUATE(4)     !запрашивать о сохранении изменений
          END
```

Пример:

```
BRW1.FocusLossAction = EIPAction:Prompted
                                    !предложить сохранить изменения
```

См. также: Ask

HideSelect (прячет кнопку выбора)

HideSelect BYTE

Свойство **HideSelect** указывает, прятать (HIDE) ли кнопку Select (указанную свойством SelectControl), когда browse вызывается для целей обновления (указанных свойством Selecting). Единичное значение (1) прячет кнопку Select, нулевое значение (0) всегда показывает кнопку Select.

Реализация: Метод ResetQueue реализует поведение, заданное свойством HideSelect.

См. также: ResetQueue, SelectControl, Selecting

InsertControl (номер элемента управления добавлением)

InsertControl SIGNED

Свойство **InsertControl** содержит номер элемента управления добавлением. Обычно он приравнивается метке соответствия поля для кнопки Insert. Методы BrowseClass используют это значение для разрешения и запрещения элемента управления в соответствующих случаях, при передаче событий элементу управления, при отражении режима добавления в соответствующем контекстном меню и т.д.

Ревлизация: Метод BrowseClass.Init не инициализирует свойства InsertControl. Вы должны инициализировать эти свойства после вызова метода BrowseClass.Init. См. *Концептуальный пример*.

См. также: UpdateToolbarButtons

ListControl (элемент управления LIST для browse)

ListControl SIGNED

Свойство **ListControl** содержит номер элемента управления LIST, который отображает данные browse.

Метод BrowseClass инициализирует свойство ListControl. См. *Концептуальный пример*.

См. также: Init

ListQueue (ссылка на очередь, содержащую данные для browse)

ListQueue &QUEUE

Свойство **ListQueue** является ссылкой на структуру, которая является источником элементов данных, отображаемых в списке.

Метод BrowseClass инициализирует свойство ListQueue. См. *Концептуальный пример*.

См. также: Init

Loaded (флаг заполненной очереди browse)

Loaded BYTE, PROTECTED

Свойство **Loaded** содержит значение, указывающее пытался или нет объект класса BrowseClass заполнить очередь просмотра browse. BrowseClass использует это свойство для гарантии того, что очередь заполнилась и во избежание лишнего

повторного заполнения.

Это свойство имеет атрибут PROTECTED, следовательно обращаться к нему можно только из методов класса BrowseClass или порожденного от него класса.

Popup (ссылка на контекстное меню browse)

Popup&PopupClass

Свойство **Popup** является ссылкой на объект класса PopupClass, используемый данным объектом BrowseClass.

Реализация: Поскольку это свойство непосредственно ссылается на PopupClass, заголовок BrowseClass включает (INCLUDE) заголовок PopupClass. Поэтому реализация PopupClass классом BrowseClass происходит автоматически.Вам не нужно предпринимать никаких действий.

Метод BrowseClass.Init активизирует объект PopupClass, на который ссылается свойство Popup. См. *Концептуальный пример*.

См. также: Init

QuickScan (флаг буферированного чтения)

QuickScan BYTE

Свойство **QuickScan** содержит значение, которое указывает объекту BrowseClass использовать ли буферированный просмотр при постраничной загрузке списочной очереди browse. Буферированный просмотр оказывает влияние только на те файловые системы, которые поддерживают буферы из нескольких записей. Подробнее см. *Драйверы баз данных*.

Нулевое значение (0) запрещает буферированный просмотр, ненулевое - разрешает. Буферирование – это естественный путь чтения записей для browse. Однако, повторное чтение буфера может повысить целостность данных в многопользовательских системах за счет более медленного чтения.

Реализация: Метод BrowseClass.Fetch реализует быстрое чтение только в процессе постраничной загрузки и только если свойство QuickScan имеет ненулевое значение. Метод BrowseClass.Fetch устанавливает (оператор SEND) строку драйвера ‘QUICKSCAN=ON’ для поддерживающих этот режим драйверов при помощи метода RelationManager.SetQuickScan.

Внимание: Метод RelationManager.SetQuickScan не устанавливает

свойство BrowseClass.QuickScan. Однако, если вы установили свойство BrowseClass.QuickScan в единичное значение (1), BrowseClass использует метод RelationManager.SetQuickScan для установки (оператор SEND) строки драйвера QUICKSCAN соответствующим файлам.

См. также: Fetch, RelationManager.SetQuickScan.

RetainRow (поведение засветки при обновлении окна)

RetainRow BYTE

Свойство **RetainRow** указывает, следует ли объекту класса BrowseClass пытаться установить засветку на той же строке списка в случае изменения порядка сортировки, модификации или другого обновляющего действия browse. Единичное значение (1) сохраняет засветку на текущей строке, нулевое значение (0) передвигает засветку на первую строку.

Задание данному свойству единичного значения может вызвать снижение производительности приложений, работающих с нынешней версией TopSpeed ODBC- драйвера.

Реализация: Метод Init устанавливает свойство RetainRow в единичное значение(1).Метод ResetQueue реализует поведение, заданное свойством RetainRow.

См. также: Init, ResetQueue

SelectControl (номер элемента управления выбором в browse)

SelectControl SIGNED

Свойство **SelectControl** содержит номер элемента управления выбором. Обычно он приравнивается метке соответствия поля для кнопки Select. Методы BrowseClass используют это значение для разрешения и запрещения элемента управления в соответствующих случаях, при передаче событий элементу управления, при отражении режима выбора в соответствующем контекстном меню и т.д.

Реализация: Метод BrowseClass.Init не инициализирует свойства SelectControl. Вы должны инициализировать эти свойства после вызова метода BrowseClass.Init. См. *Концептуальный пример*.

См. также: UpdateToolbarButtons

Selecting (флаг режима выбора)

Selecting BYTE

The **Selecting** property indicates whether the BrowseClass object selects a browse item or updates browse items. A value of zero (0) sets update mode; a value of one (1) sets select only mode.

Свойство **Selecting** переключает режимы выбора и модификации элементов данных browse. Нулевое значение (0) устанавливает режим обновления; единичное значение (1) устанавливает режим только выбора.

Свойство HideSelect оказывает влияние только в тогда, когда включен режим модификации.

Реализация: В режиме выбора двойной клик или клавиша ENTER служат для выбора элемента данных; в режиме модификации – для обновления элемента данных.

См. также: HideSelect

Sort (информация о порядках сортировки browse)

Sort &BrowseSortOrder

Свойство **Sort** является ссылкой на структуру, содержащую всю информацию о порядках сортировки для данного объекта BrowseClass. Методы используют данное свойство для реализации множественных порядков сортировки, диапазонов, фильтров и локаторов для одного и того же списка.

Implementation: Свойство BrowseClass.Sort имитирует или затеняет унаследованное свойство ViewManager.Order. Свойство Sort является ссылкой на QUEUE, объявленную в файле ABBROWSE.INC следующим образом:

```
BrowseSortOrder QUEUE(SortOrder),TYPE!информация о порядках сортировки
                                     !browse
Locator     &LocatorControl          !локатор для данного порядка
                                     !сортировки
Resets  &FieldPairsClass             !"сторожевые" поля для данного
                                     !порядка сортировки
Thumb       &ThumbClass              !ThumbClass для данного порядка
                                     !сортировки
END
```

Обратите внимание, что очередь BrowseSortOrder содержит все поля очереди SortOrder, определяемой в файле ABFILE.INC следующим образом:

```
SortOrder  QUEUE,TYPE              !информация о сортировке VIEW
Filter     &FilterQueue            !выражения фильтра, объединяемые по И
FreeElement   ANY                  !свободный элемент ключа
LimitType  BYTE                    !тип используемого диапазона
MainKey    &KEY                    !ключ
Order      &STRING                 !выражение ORDER
RangeList  &FieldPairsClass        !ограничиваемые по диапазону поля
           END
```

Очередь же SortOrder содержит ссылку на очередь FilterQueue, объявленную в файле ABFILE.INC так:

```
FilterQueue  QUEUE,TYPE            !информация о сортировке VIEW
ID
STRING(30)                         !идентификатор фильтра
Filter     &STRING                 !выражение фильтра
           END
```

Таким образом, очередь BrowseSortOrder, помимо прочего, является очередью очередей.

Метод AddSortOrder определяет порядок сортировки для browse. Метод SetSort применяет или активизирует порядок сортировки для browse. В каждый момент времени активен только один порядок сортировки.

См также: AddSortOrder, SetSort

StartAtCurrent (начальная позиция browse)

StartAtCurrent BYTE

Свойство **StartAtCurrent** указывает объекту способ исходного позиционирования - на первый элемент данных в порядке сортировки или на элемент данных, определенного содержимым буфера VIEW. Нулевое значение (0 или False) позиционирует на первый элемент данных; единичное значение (1 или True) позиционирует на элемент данных, определенный содержимым буфера VIEW.

Реализация: Метод SetSort устанавливает начальную позицию в соответствии со значением свойства StartAtCurrent. Метод SetSort позиционирует список, на основании на содержимого полей активного порядка сортировки, включая свободные элементы ключа.

Пример:

```
BRW1.StartAtCurrent = True
        !начальное позиционирование по текущим значениям
ST:StateCode = 'K'
        !установить значение компонента ключа
BrowseSt.Init(?StList,StateQ.ViewPosition,StView,StateQ,Relate:State,SELF)
```

См. также: SetSort

TabAction (действия при редактировании по месту по нажатию клавиши tab)

TabAction BYTE

Свойство **TabAction** указывает, какое действие следует выполнить, когда конечный пользователь нажимает клавишу TAB в процессе редактирования по месту. Существует два типа действий, управляемых свойством TabAction:

- что делать с любыми изменениями (стандартное действие, сохранение, отмена или запрос пользователя)
- какой режим использовать далее (продолжение редактирования или возврат к режиму просмотра записей)

Указанные действия осуществляются методом Ask. Выбор действий следует осуществлять присваиванием свойству TabAction комбинаций следующих констант EQUATE. Эти значения EQUATE находятся в ABBROWSE.INC:

```
ITEMIZE,PRE(EIPAction)
Default   EQUATE(0)          !сохранять в соответствии с методом Ask
Always    EQUATE(1)          !всегда сохранять изменения
Never     EQUATE(2)          !никогда не сохранять изменения
Prompted  EQUATE(4)          !запрашивать о сохранении изменений
Remain    EQUATE(8)          !продолжать редактирование
          END
```

Пример:

```
BRW1.TabAction = EIPAction:Prompted
                             !предложить сохранить изменения
BRW1.TabAction = EIPAction:Prompted+EIPAction:Remain
                             !предложить сохранить изменения,
                             !продолжить редактирование
```

См. также: Ask

Toolbar (объект Toolbar для browse)

Toolbar &ToolbarClass

Свойство **Toolbar** является ссылкой на объект класса ToolbarClass для данного объекта BrowseClass. Объект ToolbarClass собирает события для инструментальной панели и передает их активному объекту ToolbarTarget для обработки.

Метод AddToolbarTarget регистрирует ToolbarTarget, так же как и объект ToolbarListBoxClass в качестве потенциального адресата объекта ToolbarClass.

Метод ToolbarClass.SetTarget устанавливает активного адресата для объекта ToolbarClass.

Реализация: Объект ToolbarClass для browse является объектом, который обнаруживает события инструментальной панели, такие как, scroll down и page down, и передает их *активному* объекту ToolbarListBoxClass (ToolbarTarget). В стандартной реализации шаблона существует единственная глобальная инструментальная панель и объект ToolbarClass, который может управлять несколькими различными browse и form, каждая из которых является ToolbarTarget. В текущий момент времени может быть активен только один ToolbarTarget.

См. также: ToolbarItem, AddToolbarTarget, ToolbarClass.SetTarget

ToolbarItem (объект ToolbarTarget для browse)

ToolbarItem &ToolbarListBoxClass

Свойство **ToolbarItem** является ссылкой на класс ToolbarListBoxClass для данного объекта BrowseClass. Объект ToolbarListBoxClass (ToolbarTarget) получает события инструментальной панели (от объекта ToolbarClass) и обрабатывает их.

Метод AddToolbarTarget регистрирует ToolbarTarget, так же как и объект ToolbarListBoxClass в качестве потенциального адресата объекта ToolbarClass.

Метод ToolbarClass.SetTarget устанавливает активного адресата для объекта ToolbarClass.

Реализация: Объект ToolbarClass для browse является объектом, который обнаруживает события инструментальной панели, такие как, scroll down и page down, и передает их *активному* объекту ToolbarListBoxClass (ToolbarTarget). В стандартной реализации шаблона существует единственная глобальная инструментальная панель

и объект ToolbarClass, который может управлять несколькими различными browse и form, каждая из которых является ToolbarTarget. В текущий момент времени может быть активен только один ToolbarTarget.

См. также: Toolbar, AddToolbarTarget, ToolbarClass.SetTarget

Toolbar (объект Toolbar для browse)

Toolbar &ToolbarClass

Свойство **Toolbar** является ссылкой на класс ToolbarClass для данного объекта BrowseClass. Объект ToolbarClass собирает события для инструментальной панели и передает их активному объекту ToolbarTarget для обработки.

Метод AddToolbarTarget регистрирует ToolbarTarget, так же как и объект ToolbarListBoxClass в качестве потенциального адресата объекта ToolbarClass.

Метод ToolbarClass.SetTarget устанавливает активного адресата для объекта ToolbarClass.

Реализация: Объект ToolbarClass для browse является объектом, который обнаруживает события инструментальной панели, такие как, scroll down и page down, и передает их *активному* объекту ToolbarListBoxClass (ToolbarTarget). В стандартной реализации шаблона существует единственная глобальная инструментальная панель и объект ToolbarClass, который может управлять несколькими различными browse и form, каждая из которых является ToolbarTarget. В текущий момент времени может быть активен только один ToolbarTarget.

См. также: ToolbarItem, AddToolbarTarget, ToolbarClass.SetTarget

Window (объект WindowManager)

Window &WindowManager

Свойство **Window** является ссылкой на объект WindowManager для данного объекта BrowseClass. Объект WindowManager передает события активному объекту BrowseClass для обработки.

Метод WindowManager.AddItem регистрирует объект BrowseClass в объекте WindowManager, поэтому объект WindowManager может передавать события.

Метод Init устанавливает значение свойства Window.

Реализация: Объект WindowManager вызывает метод BrowseClass.TakeEvent, поэтому при необходимости объект BrowseClass может

обрабатывать события.

См. также: Init, WindowManager.AddItem

Методы BrowseClass

Класс BrowseClass наследует все методы класса ViewManager, от которого он порожден. Более подробно см. *Методы класса ViewManager*.

Помимо унаследованных методов (или вместо них), класс BrowseClass содержит методы, перечисленные ниже.

Функциональная организация - ожидаемое использование

В качестве помощи для понимания BrowseClass, можно разделить многообразные методы этого класса на две большие категории в соответствии с их ожидаемым использованием - первичный интерфейс и виртуальные методы. Такое структурирование отражает то, что мы понимаем под типичным использованием методов BrowseClass.

Методы первичного интерфейса

Init	инициализирует объект класса BrowseClass
AddEditControl	задает индивидуальное редактирование по месту для поля browse
AddField	определяет соответствие полей FILE и QUEUE
AddLocator	связывает локатор с его порядком сортировки
AddResetField	задает “сторожевое” поле (при изменении которого обновляется список browse)
AddSortOrder	добавляет порядок сортировки списку browse
AddToolbarTarget	связывает список browse с объектом инструментальной панели
SetAlerts	взводит клавиши для списка, локатора и экранных элементов редактирования
Kill [V]	закрывает объект BrowseClass

Основного применения:

Next[V]	последовательно читает следующую запись VIEW
Previous[V]	последовательно читает предыдущую запись VIEW
Ask	модифицирует выбранный элемент данных
TakeEvent[V]	обрабатывает текущее событие ACCEPT- цикла
TakeNewSelection[V]	обрабатывает выбор нового элемента данных

[V] Эти методы также являются виртуальными.

Эпизодического применения:

ApplyRange	обновляет список browse в соответствии с заданным диапазоном
AskRecord	редактирует по месту выбранный элемент данных
PostNewSelection	посылает событие EVENT:NewSelection списку browse
Records	возвращает количество записей в списке browse
ResetResets	запоминает текущие значения “сторожевых” полей
ResetThumbLimits	переустанавливает границы указателя вертикальной прокрутки в соответствии с результирующей выборкой
TakeAcceptedLocator	применяет введенное значение локатора
UpdateResets	копирует “сторожевые” поля в буфер файла
UpdateThumb	позиционирует указатель вертикальной прокрутки
UpdateThumbFixed	позиционирует фиксированный указатель вертикальной прокрутки
UpdateWindow	выполняет скроллирование, поиск, ограничение по диапазону и т.д.

Виртуальные методы

Обычно эти методы не вызываются непосредственно – их вызывают методы первичного интерфейса. Однако мы предполагаем, что у вас возникнет необходимость переопределить эти методы, и, поскольку они являются виртуальными, сделать это несложно. В противном случае, и их стандартное поведение представляется разумным.

ApplyRange	условно ограничивает диапазон записей и фильтрует их
Fetch	загружает страницу элементов данных в список browse
Kill	закрывает объект BrowseClass
Next	получает следующую запись из VIEW browse
Previous	получает предыдущую запись из VIEW browse
Reset[1]	reset the view position репозиционирует VIEW
ResetQueue	заполняет или перестраивает очередь browse
ScrollEnd	прокручивает к первому или последнему элементу данных
ScrollOne	прокручивает вверх или вниз на один элемент данных
ScrollPage	прокручивает вверх или вниз на одну страницу данных
SetQueueRecord	копирует данные из буфера файла в буфер очереди
SetSort	применяет порядок сортировки для browse
ResetSort	применяет порядок сортировки для browse
TakeKey	обрабатывает нажатие горячих клавиш
TakeEvent	обрабатывает текущее событие ACCEPT-цикла
TakeNewSelection	обрабатывает выбор нового элемента данных из списка browse
TakeScroll	обрабатывает событие прокрутки

TakeVCRScroll	обрабатывает событие прокрутки при помощи кнопок VCR
UpdateBuffer	Копирует данные из буфера очереди в буфер файла
UpdateViewRecord	копирует выбранный элемент данных в соответствующие буферы файлов

Совет: Используйте комбинацию методов ResetSort и UpdateWindow для обновления и перепоказа ваших ABC Browse. Можно также вызвать метод WindowManager.Reset.

AddEditControl (описывает пользовательский класс редактирования по месту)

AddEditControl([*editclass*], *столбец* [, *автоликвидация*] **)**

AddEditControl	Описывает пользовательский класс редактирования по месту для поля browse.
editclass	Метка объекта класса EditClass. Если параметр опущен, указанный *столбец* не может редактироваться.
столбец	Численная костанта, переменная EQUATE, или выражение, которое указывает столбец browse, который будет редактироваться при помощи описанного объекта *editclass*.Единичное значение (1) указывает первый столбец; двойка (2) указывает второй столбец и т.д.
автоликвидация	Численная костанта, переменная EQUATE, или выражение, которое указывает, должен ли метод BrowseClass.Kill ликвидировать объект *editclass* оператором DISPOSE. Нулевое значение (0) оставляет объект нетронутым. Ненулевое значение ликвидирует объект. Если параметр опущен, его значение принимается равным нулю (0).

Метод **AddEditControl** описывает *editclass*, который определяет элемент управления редактированием по месту для *столбца* browse. Используйте *автоликвидацию* с осторожностью: освобождать при помощи оператора DISPOSE следует только память, выделенную при помощи оператора NEW. Более подробно об операторах NEW и DISPOSE см. *Описание языка*.

Реализация: Нет необходимости вызывать данный метод для использования стандартного значения параметра *editclass*. Если Вы не вызываете метод AddEditControl для столбца списка browse, BrowseClass автоматически активизирует для данного столбца объект EditClass, объявленный в файле ABBROWSE.INC.

Значение параметра *автоликвидация* по умолчанию равно нулю (0). Метод BrowseClass.Kill посредством оператора DISPOSE освобождает объекты *editclass* только если параметр *автоликвидация* содержит ненулевое значение.

Если необходимо, метод BrowseClass.Ask активизирует обьекты *editclass*, затем создает и уничтожает элементы управления редактированием по месту в соответствии с запросом конечного пользователя на добавление или изменение.

Пример:

```
INCLUDE('ABBROWSE.INC')          !объявить browse и связанные классы
INCLUDE('MYCOMBO.INC')           объявить собственный класс управления
                                 !редактированием по месту
CODE
MyBrowse.AddEditControl(,1)      !нередактируемый первый столбец
MyBrowse.AddEditControl(ComboClass,2)
                                 !столбец № 2 редактируется при помощи
                                 !COMBO
```

См. также: Ask

AddField (определяет пару полей файла и очереди)

AddField(*поле файла, поле очереди* **)**

AddField	Определяет соответствие полей файла и очереди для столбца списка browse.
Поле файла	Полная метка поля файла или переменной памяти. *???? ?????* является первоисточником данных списка LIST.
Поле очереди	Полная метка соответствующего поля очереди. *Поле очереди* загружается из *поля файла* и является непосредственным источником данных, отображаемых в экранном списке (LIST).

Метод **AddField** определяет соответствие полей файла и очереди для столбца списка browse. Вы должны вызвать AddField для каждого столбца, показываемого в списке browse.

Также Вы можете использовать метод AddField для образования пар переменных памяти с полями очереди, указывая метку переменной в качестве *поля файла*.

Реализация: Для списков browse, поддерживающих редактирование по месту, вы должны добавить поля (вызовая метод AddField) в той же последовательности, в которой были объявлены поля очереди browse.

Пример:

```
INCLUDE('ABBROWSE.INC')                  !объявить browse и связанные классы
States    FILE,DRIVER('TOPSPEED'),PRE(StFile)    !объявить файл States
```

```
ByCode      KEY(StFile:Code),NOCASE,OPT
Record      RECORD,PRE()
Code        STRING(2)
Name        STRING(20)
StQType     QUEUE,TYPE                          !бъявить тип St QUEUE
Code        LIKE(StFile:Code)
Name        LIKE(StFile:Name)
Position    STRING(512)
            END
BrowseStClassCLASS(BrowseClass),TYPE            !объявить класс BrowseSt
Q           &StQType
        END
StQ     StQType                 !объявить реальную очередь StQ QUEUE
BrowseSt    BrowseStClass       !объявить объект BrowseSt
        CODE
BrowseSt.AddField(StFile:Code,BrowseSt.Q.Code)  !объединить в пары поля in
BrowseSt.AddField(StFile:Name,BrowseSt.Q.Name)  !файла и очереди
```

AddLocator (назначает локатор)

AddLocator(*локатор***)**

AddLocator Назначает объект локатора для определенного порядка сортировки.
локатор Метка объекта локатора.

Метод **AddLocator** назначает объект локатора для порядка сортировки, определенного предшествующим вызовом метода AddSortOrder или метода SetSort. Обычно, метод AddLocator вызывается непосредственно после метода AddSortOrder.

Реализация: Установленный *локатор* связан с конкретным порядком сортировки – он доступен только тогда, когда активен этот порядок сортировки. Метод SetSort применяет или активизирует порядок сортировки для browse. В текущий момент времени может быть активен только один порядок сортировки.

Пример:

```
BrowseSt.AddSortOrder(BrowseSt:Step,StFile:ByCode)
                                    !добавить порядок сортировки
BrowseSt.AddLocator(BrowseSt:Locator)
                                    !и связанный с ним локатор
BrowseSt:Locator.Init(?Loc,StFile:StCode,1,BrowseSt)
                                    !инициализировать объект
                                    !локатора
```

См. также: AddSortOrder, LocatorClass, SetSort

AddResetField (устанавливает поле для отслеживания изменений – "сторожевое" поле)

AddResetField(*"сторожевое" поле***)**

AddResetField	Задает поле, при изменении значения которого, список browse должен быть перепоказан.
"сторожевое" поле	Метка поля, отслеживающего изменения.

Для активного порядка сортировки (определенного предшествующим вызовом метода AddSortOrder или метода SetSort) метод **AddResetField** задает поле, по которому объект browse отслеживает изменения, затем, когда содержимое поля изменено, обновляет список browse. Обычно, метод AddResetField вызывается непосредственно после метода AddSortOrder.

Для задания нескольких "сторожевых" полей для порядка сортировки, метод AddResetField может вызываться несколько раз.

Реализация: Заданное "сторожевое" поле связан с конкретным порядком сортировки – оно доступно только тогда, когда активен этот порядок сортировки. Метод SetSort устанавливает активный порядок сортировки для browse. Метод SetSort также вызывает метод ApplyRange, применяющий заданные ранее "сторожевые поля", а метод SetSort обновляет browse, когда происходит изменение.

Метод WindowManager.Reset при необходимости также проводит оценку "сторожевых" полей и последующее обновление browse для любых объектов browse, зарегистрированных WindowManager.

Пример:

```
BrowseSt.AddSortOrder(BrowseSt:Step,StFile:ByCode)
                                    !добавить порядок сортировки
BrowseSt.AddLocator(BrowseSt:Locator)
                                    !с соответствующим локатором
BrowseSt.AddResetField(Local:StFilter)
                                    !и "сторожевым" полем
```

См. также: AddSortOrder, SetSort, WindowManager.Reset

AddSortOrder (задает порядок сортировки для browse)

AddSortOrder([*индикатор*] [, *ключ*]**), PROC**

AddSortOrder	Задает дополнительный порядок сортировки для списка browse.

индикатор Метка объекта StepClass, который управляет линейкой вертикальной прокрутки и поведением ее индикатора. Если параметр опущен, применяется фиксированный индикатор. О поведении указателя см. подробнее *Control Templates—BrowseBox*.

ключ Метка ключа, по которому проходит сортировка. Если параметр опущен, список browse не сортируется – элементы данных выводятся в физическом порядке или в порядке, заданном унаследованным методом AppendOrder.

Метод **AddSortOrder** задает доплнительный порядок сортировки для списка browse и возвращает его номер, который используется методом SetSort. Метод AddSortOrder должен быть вызван для каждого порядка сортировки, применяемого для browse.

Метод AddLocator добавляет локатор для порядка сортировки, заданного предшествующим вызовом AddSortOrder.

Метод AddResetField добавляет “сторожевое” поле для порядка сортировки, определенного предшествующим вызовом AddSortOrder. Можно добавлять несколько “сторожевых” полей для каждого порядка сортировки многократным вызовом AddResetField.

Унаследованный метод AddRange добавляет диапазон для порядка сортировки, определенного предшествующим вызовом AddSortOrder.

Этот метод имеет атрибут PROC, следовательно его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод AddSortOrder добавляет один элемент в свойство Sort.
Возвращаемый тип данных: BYTE

Пример:

```
BrowseSt.AddSortOrder(BrowseSt:Step,StFile:ByCode)
                                        !добавить порядок сортировки
BrowseSt.AddLocator(BrowseSt:Locator)
                                        !со связанным локатором
BrowseSt.AddResetField(Local:StFilter)
                                        !и “сторожевым” полем
```

См. также: AddLocator, AddResetField, Sort, StepClass, SetSort, ViewManager.AddRange, ViewManager.AppendOrder

AddToolbarTarget (устанавливает инструментальную панель для browse)

AddToolbarTarget(*инструментальная панель* **)**

AddToolbarTarget	Регистрирует browse в качестве потенциального адресата указанной инструментальной панели.
инструментальная панель	Метка объекта ToolbarClass, который направляет события инструментальной панели данному объекту BrowseClass.

Метод ToolbarClass.SetTarget устанавливает активного адресата для объекта ToolbarClass.

Реализация: Объект ToolbarClass для browse является объектом, который обнаруживает события инструментальной панели, такие как, scroll down и page down, и посылает их на *активный* объект ToolbarTarget. В стандартной реализации шаблона существует единственная глобальная инструментальная панель, и объект ToolbarClass, который может управлять несколькими различными browse и form, каждая из которых является ToolbarTarget. В текущий момент времени может быть активен только один ToolbarTarget.

Пример:

```
BrowseSt.AddToolbarTarget(Browse:Toolbar)
                                    !связать объект BrowseSt с обьъектом
                                    !Toolbar
BrowseZIP.AddToolbarTarget(Browse:Toolbar)
                                    !связать объект BrowseZIP с обьъектом
                                    !Toolbar
!код программы
        Browse:Toolbar.SetTarget(?StList)
                                    !текущим адресатом  панели инструментов
                                    !является список штатов
!код программы
        Browse:Toolbar.SetTarget(?ZIPList)
                                    !текущим адресатом  панели инструментов
                                    !является список ZIP-кодов
```

См. также: Toolbar, ToolbarItem, ToolbarClass.SetTarget

ApplyRange (обновление browse на основе "сторожевых" полей и значений диапазонов)

ApplyRange, VIRTUAL, PROC

Метод ApplyRange проверяет текущий статус "сторожевых" полей и значений диапазонов и, при необходимости, обновляет список browse. Затем он возвращает значение, указывающее, требуется ли обновление экрана.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно его можно вызывать как процедуру, игнорируя возвращаемый результат.

Унаследованный метод AddRange добавляет диапазон для каждого порядка сортировки. Метод AddResetField устанавливает "сторожевые" поля для каждого порядка сортировки в browse.

Реализация: Метод ApplyRange возвращает единичное значение (1), если требуется обновить экран, или нулевое значение (0), если этого не требуется.

Возвращаемый тип данных: BYTE

Пример:

```
IF BrowseSt.ApplyRange()
        !обновить очередь browse если произошли изменения
DISPLAY(?StList)
        !перерисовать LIST если очередь обновлена
        END
```

См. также: AddResetField, ViewManager.AddRange

Ask (модифицирует выбранный элемент данных browse)

Ask(*запрос***), VIRTUAL, PROC**

Ask	Модифицирует выбранную запись browse.
запрос	Численная константа, переменная, EQUATE или выражение, указывающее запрошенное действие модификации. Допустимыми действиями являются вставка, изменение и удаление (Insert, Change ? Delete).

Метод **Ask** модифицирует выбранную browse запись и возвращает значение, указывающее завершена или отменена запрошенная модификация.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Depending on the value of the AskProcedure property, the Ask method either calls the WindowManager.Run method to execute a specific update procedure, or it calls the AskRecord method to do an edit-in-place update.

В зависимости от значения свойства AskProcedure, метод Ask вызывает либо метод WindowManager.Run для выполнения конкретной модифицирующей процедуры, либо метод AskRecord для выполнения редактирования по месту.

Метод Ask полагает, что ранее был вызван метод UpdateViewRecord, что гарантирует корректность буфера записи.

Метод TakeEvent вызывает метод Ask.

Возвращаемые значения EQUATE объявлены в файле \LIBSRC\TPLEQU.CLW:

```
RequestCompleted    EQUATE (1)                    !обновление завершено
RequestCancelled    EQUATE (2)                    !обновление прервано
```

Константы для запроса объявлены в файле \LIBSRC\TPLEQU.CLW:

```
InsertRecord        EQUATE (1)          !добавить запись  в таблицу
ChangeRecord        EQUATE (2)          !изменить текущую запись
DeleteRecord        EQUATE (3)          !удалить текушую запись
```

Возвращаемый тип данных: BYTE

Пример:

```
BrowseClass.TakeEvent PROCEDURE
!procedure data
 CODE      !procedure code
CASE ACCEPTED()
OF 0
OF SELF.DeleteControl
            SELF.Window.Update()
        SELF.Ask(DeleteRecord)                  !удалить элемент
        OF SELF.ChangeControl
```

```
            SELF.Window.Update()
            SELF.Ask(ChangeRecord)              !изменить элемент
            OF SELF.InsertControl
            SELF.Window.Update()
            SELF.Ask(InsertRecord)              !добавить элемент
            OF SELF.SelectControl
SELF.Window.Response = RequestCompleted
POST(EVENT:CloseWindow)
ELSE
SELF.TakeAcceptedLocator
            END
```

См. также: AskProcedure, AskRecord, TakeEvent

AskRecord (редактирование по месту выбранного browse элемента данных)

AskRecord(запрос), VIRTUAL, PROC

AskRecord Выполняет редактирование по месту выбранной записи browse.
запрос Численная константа, переменная, EQUATE или выражение, указывающее запрошенное действие редактирования по месту. Допустимыми действиями являются вставка, изменение и удаление (Insert, Change ? Delete).

Метод **AskRecord** выполняет редактирование по месту выбранных строки и столбца browse и возвращает значение, указывающее завершено или прервано запрошенное редактирование.

Метод AddEditControl описывает EditClass для определенного столбца.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: The AskRecord method assumes the UpdateViewRecord method has been called to ensure correct record buffer contents.

Метод AskRecord полагает, что ранее был вызван метод UpdateViewRecord, что гарантирует корректность буфера записи.

За методом AskRecord **должен** следовать метод ResetFromAsk.

Метод Ask вызывает метод AskRecord.

Возвращаемые значения EQUATE объявлены в файле \LIBSRC\TPLEQU.CLW:

```
RequestCompleted        EQUATE (1)          !обновление завершено
RequestCancelled        EQUATE (2)          !обновление прервано
```

Константы для запроса объявлены в файле \LIBSRC\TPLEQU.CLW:

```
InsertRecord        EQUATE (1)          !добавить запись  в таблицу
ChangeRecord        EQUATE (2)          !изменить текущую запись
DeleteRecord        EQUATE (3)          !удалить текушую запись
```

Возвращаемый тип данных: BYTE

Пример:

```
BrowseClass.Ask PROCEDURE(BYTE Req)
Response BYTE
 CODE
 LOOP
  SELF.Window.VCRRequest = VCR:None
  IF Req=InsertRecord THEN
   SELF.PrimeRecord
  END
  IF SELF.AskProcedure
   Response = SELF.Window.Run(SELF.AskProcedure,Req)
                                        !обновление с редактированием
                                        !по месту
   SELF.ResetFromAsk(Req,Response)
  ELSE
   Response = SELF.AskRecord(Req)
  END
 UNTIL SELF.Window.VCRRequest = VCR:None
 RETURN Response
```

См. также: AddEditControl, Ask, ResetFromAsk

Fetch (читает страницу элементов данных browse)

Fetch(направление), VIRTUAL, PROTECTED

Fetch Загружает страницу элементов данных в очередь списка browse.

направление Численная константа, переменная, EQUATE или выражение, указывающее, считывать следующий или предыдущий набор элементов данных.

Метод **Fetch** загружает следующую или предыдущую страницу элементов данных в очередь списка browse.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROTECTED, следовательно может быть вызван только из методов класса BrowseClass или порожденного от него класса.

Реализация: Метод Fetch вызывается методами ResetQueue, ScrollOne, ScrollPage, и ScrollEnd. Страница вмещает столько элементов данных, сколько и экранный элемент управления LIST.

Константы *направления* BrowseClass.Fetch описаны в файле ABBROWSE.INC следующим образом:

```
FillBackward        EQUATE(1)       !назад
FillForward         EQUATE(2)       !вперед
```

Пример:

```
ScrollOne PROCEDURE(SIGNED Event)
            CODE
 IF Event = Event:ScrollUp AND CurrentChoice > 1
  CurrentChoice -= 1
 ELSIF Event = Event:ScrollDown AND CurrentChoice < RECORDS(ListQueue)
  CurrentChoice += 1
 ELSE
  ItemsToFill = 1
  MyBrowse.Fetch( CHOOSE( Event = EVENT:ScrollUp, FillForward, FillBackward ) )
 END
```

См. также: ResetQueue, ScrollOne, ScrollPage, ScrollEnd

Init (инициализирует объект BrowseClass)

Init(*экранный список, позиция view, view, очередь, relationmanager , windowmanager* **)**

Init	Инициализирует объект класса BrowseClass.
экранный список	Численная константа, переменная, EQUATE или выражение, содержащее номер управляющего элемента LIST списка browse.
позиция view	Метка строковой переменной в структуре *очереди,* содержащая позицию (POSITION) VIEW.
view	Метка VIEW, показываемой в списке browse.
очередь	Метка QUEUE для *экранного списка*.
relationmanager	Метка объекта RelationManager первичного файла списка browse. Более подробно см. *Relation Manager.*

windowmanager Метка объекта WindowManager. Более подробно см. *Window Manager*.

Метод **Init** инициализирует объект класса BrowseClass.

Реализация: Помимо прочего, метод Init вызывает метод PARENT.Init (ViewManager.Init) для инициализации объекта ViewManager списка browse. Более подробно см. ViewManager.

Метод Init активизирует объект PopupClass для browse.

Метод Init вызывает метод WindowManager.AddItem для регистрации в WindowManager присутствия объекта BrowseClass.

Пример:

```
CODE                                    !настройка объекта BrowseClass:
BrowseState.Init(?StateList,|
                                        ! идентифицировать экранный списокi(LIST),
StateQ.Position,|                       ! позицию VIEW,
StateView, |                            ! собственно VIEW,
StateQ,    |                            ! показываемую QUEUE,
Relate:State|                           ! RelationManager первичного файла,
ThisWindow)                             ! WindowManager
```

См. также: ViewManager.Init, PopupClass, WindowManager.AddItem

Kill (закрывает объект BrowseClass)

Kill, VIRTUAL

Метод **Kill** закрывает объект BrowseClass.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Помимо прочего, метод BrowseClass.Kill вызывает метод PARENT.Kill (ViewManager.Kill) для закрытия объекта ViewManager. Более подробно см. *View Manager*.

Пример:

```
CODE                                    !настройка объекта BrowseClass:
BrowseState.Init(?StateList,|           ! идентифицировать экранный списокi(LIST),
StateQ.Position,|                       ! позицию VIEW,
StateView, |                            ! собственно VIEW,
StateQ,    |                            ! показываемую QUEUE,
```

```
Relate:State|                          ! RelationManager первичного файла,
ThisWindow)                            ! WindowManager
!код программы
BrowseState.Kill                       !закрыть объект BrowseClass
```

См. также: ViewManager.Kill

Next (читает следующий элемент данных browse)

Next, VIRTUAL

Метод **Next** читает следующую запись VIEW и возвращает значение, указывающее на успешный или неудачный результат.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Next возвращает значение Level:Benign в случае успешного завершения, значение Level:Notify в случае достижения конца файла и значение Level:Fatal в случае обнаружения фатальной ошибки.

Реализация: Соответствующие константы возвращаемых значений описаны в файле ABERROR.INC. Более подробно об уровнях серьезности??. ? *Error Class.*

```
Level:Benign        EQUATE(0)
Level:User          EQUATE(1)
Level:Program       EQUATE(2)
Level:Fatal         EQUATE(3)
Level:Cancel        EQUATE(4)
Level:Notify        EQUATE(5)
```

Метод Next вызывается методами Fetch и ResetThumbLimits. Помимо прочего, метод Next вызывает метод PARENT.Next (ViewManager.Next). Более подробно см. *ViewManage* .

Возвращаемый тип данных: BYTE

Пример:

```
CASE MyBrowse.Next()                   !прочесть следующую запись
OF Level:Benign                        !если успешно – продолжить
OF Level:Fatal                         !если фатальная ошибка
                                       RETURN
```

```
                                    !завершить рпроцедуру
OF Level:Notify                     !если достигнут конец файла
MESSAGE('Reached end of file.')     !сообщить об этомм
        END
```

См. также: Fetch, ResetThumbLimits

PostNewSelection (посылает событие EVENT:NewSelection списку browse)

PostNewSelection

Метод **PostNewSelection** посылает событие EVENT:NewSelection списку browse для поддержки скроллинга, добавлений, удалений и других изменений позиции в пределах списка browse.

Реализация: Константы событий объявлены в файле EQUATES.CLW.

Пример:

```
UpdateMyBrowse  ROUTINE             !код обновления
MyBrowse.ResetFromFile              !после добавления или изменения –
                                    !перестроить очередь по файлу
MyBrowse.PostNewSelection           !после обновления послать событие
                                    !Event:NewSelection
                                    !для корректного обновления окна
```

Previous (get the previous browse item)

Previous, VIRTUAL

Метод **Previous** читает предыдущую запись VIEW и возвращает значение, указывающее успешный или неудачый результат.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Implementation: Previous возвращает значение Level:Benign в случае успешного завершения, значение Level:Notify в случае достижения начала файла и значение Level:Fatal в случае обнаружения фатальной ошибки. Соответствующие константы возвращаемых значений описаны в файле ABERROR.INC. Более подробно об уровнях серьезностисм. в *Error Class*.

```
Level:Benign        EQUATE(0)
```

```
Level:User          EQUATE(1)
Level:Program       EQUATE(2)
Level:Fatal         EQUATE(3)
Level:Cancel        EQUATE(4)
Level:Notify        EQUATE(5)
```

Метод Previous вызывается методами Fetch и ResetThumbLimits. Помимо прочего, метод Previous вызывает метод PARENT.Next (ViewManager.Previous). Более подробно см. *ViewManager* .

Возвращаемый тип данных: BYTE

Пример:

```
CASE MyBrowse.Previous()                !прочесть предыдущую запись
OF Level:Benign                         !если успешно – продолжиить
OF Level:Fatal                          !если фатальная ошибка
RETURN                                  !завершить процедуру
OF Level:Notify                         !если достигнуто начало файла
MESSAGE('Reached beginning of file.')!сообщить об этом
        END
```

См. также: Fetch, ResetThumbLimits

Records (возвращает количество элементов данных очереди browse)

Records, PROC

Метод **Records** возвращает количество записей в очереди browse *и* запрещает соответствующие элементы управления, если количество записей равно нулю.

Этот метод имеет атрибут PROC, следовательно его можно вызывать как процедуру, игнорируя возвращаемый результат.

Возвращаемый тип данных: LONG

Пример:

```
DeleteMyBrowse    ROUTINE       !код удаления
MyBrowse.Records()              !запретить кнопку удаления (и пункт меню)
                                !при отсутствии записей
```

ResetFromAsk (обновляет browse после модификации)

ResetFromAsk(*запрос, ответ*), VIRTUAL, PROTECTED

ResetFromAsk Обновляет browse после модификациии.

запрос Численная константа, переменная, EQUATE или выражение, указывающее тип запрошенной модификации. Допустимыми модификациями являются добавление, изменение и удаление.

ответ Численная константа, переменная, EQUATE или выражение, указывающее, была ли завершена или прервана запрашиваемая модификация.

Метод **ResetFromAsk** обновляет browse после модификациии элемента данных методом Ask или AskRecord.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROTECTED, следовательно может быть вызван только из методов класса BrowseClass или порожденного от него класса.

Реализация: При необходимости методы Ask и AskRecord вызывают метод ResetFromAsk для обновления объекта browse.

При необходимости метод ResetFromAsk сбрасывает кэш VIEW объекта BrowseClass на диск (FLUSH), вызывает соответствующий метод обновления (ResetQueue, ResetFromFile, или ResetFromView) для перестроения очереди, затем завершает любой незавершенный запрос на прокрутку, сделанный одновременно с модификацией. См. *WindowManager.VCRRequest.*

Константы для *запроса* объявлены в файле \LIBSRC\TPLEQU.CLW:

```
InsertRecord    EQUATE (1)    !добавить запись  в таблицу
ChangeRecord    EQUATE (2)    !изменить текущую запись
DeleteRecord    EQUATE (3)    !удалить текущую запись
```

Константы для *ответа* описаны в файле \LIBSRC\TPLEQU.CLW следующим образом:

```
RequestCompleted    EQUATE (1)    !обновление завершено
RequestCancelled    EQUATE (2)    !обновление превано
```

Пример:

```
BrowseClass.Ask PROCEDURE(BYTE Req)
Response BYTE
 CODE
 LOOP
  SELF.Window.VCRRequest = VCR:None
  IF Req=InsertRecord THEN
   SELF.PrimeRecord
  END
  IF SELF.AskProcedure
   Response = SELF.Window.Run(SELF.AskProcedure,Req)
   SELF.ResetFromAsk(Req,Response)     !обновить browse после модификации
  ELSE
   Response = SELF.AskRecord(Req)
  END
 UNTIL SELF.Window.VCRRequest = VCR:None
 RETURN Response
```

См. также: Ask, AskRecord, ResetQueue, ResetFromFile, ResetFromView, WindowManager.VCRRequest

ResetFromBuffer (заполнение очереди, начиная с буфера записи)

ResetFromBuffer, VIRTUAL

Метод **ResetFromBuffer** заполняет или перестраивает очередь browse начиная с буфера записи первичного файла (и буферов вторичных файлов, если такие имеются). Если запись найдена, ResetFromBuffer заполняет очередь browse начиная с данной записи. Если же запись не найдена ResetFromBuffer, заполняет очередь browse ближайшей совпадающей записи.

Если активный порядок сортировки (ключ) разрешает дублирование записей и такие записи существуют, ResetFromBuffer заполняет очередь browse, начиная с *первой* совпадающей записи.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Совет: используйте ResetFromBuffer, когда позиции и значения первичного и вторичного файлов допустимы, но результирующая выборка больше не соответствует значениям буферов. Например, после поиска или перемещения указателя прокрутки.

Реализация: ResetFromBuffer завершается успешно, даже если не существует точно совпадающей записи, и обычно используется для поиска подходящей записи после передвижения указателя вертикальной прокрутки.

Метод ResetFromBuffer вызывает метод ViewManager.Reset для позиционирования, а затем вызывает метод ResetQueue для заполнения очереди browse.

Пример:

```
IF EVENT() = EVENT:ScrollDrag
        !передвинут указатель вертикальной прокрутки
IF ?MyList{PROP:VScrollPos} <= 1
        !обработать прокрутку на первую страницу
POST(Event:ScrollTop, ?MyList)
ELSIF ?MyList{PROP:VScrollPos} = 100
        !обработать прокрутку на последнюю страницу
POST(Event:ScrollBottom, ?MyList)
ELSE    !обработать промежуточную прокрутку
MyBrowse.Sort.FreeElement =
MyBrowse.Sort.Step.GetValue(?MyList{PROP:VScrollPos})
MyBrowse.ResetFromBuffer
        !и перестроить очередь начиная с этой точки
        END
        END
```

См. также: ViewManager.Reset, ResetQueue

ResetFromFile (заполняет очередь, начиная с позиции (POSITION) файла)

ResetFromFile, VIRTUAL

Метод **ResetFromFile** заполняет или дополняет перестраивает очередь browse, начиная с текущей позиции (POSITION) первичного файла. Если позиция (POSITION) не установлена, ResetFromFile заполняет очередь browse, начиная с начала файла.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Совет: используйте ResetFromFile, когда позиция первичного файла имеет допустимое значение, но вторичных записей и их содержимого может не быть. Например при возврате

из режима модификации.

Реализация: Метод завершается успешно, даже если буфер записи пуст, и обычно используется для чтения текущей записи после модификации.

Пример:

```
MyBrowseClass.ResetFromAsk PROCEDURE(*BYTE Request,*BYTE Response)
 CODE
 IF Response = RequestCompleted
  FLUSH(SELF.View)
  IF Request = DeleteRecord
   DELETE(SELF.ListQueue)
   SELF.ResetQueue(Reset:Queue)     !перестроить очередь после удаления
  ELSE
   SELF.ResetFromFile               !перестроить очередь после добавления или
                                    !изменения
  END
 ELSE
  SELF.ResetQueue(Reset:Queue)
 END
```

ResetFromView (перенастройка browse по текущему результирующей выборке)

ResetFromView, VIRTUAL

The **ResetFromView** method resets the BrowseClass object to conform to the current result set.

Метод **ResetFromView** перенастраивает объект BrowseClass для соответствия текущей результирующей выборке.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Совет: Используйте ResetFromVIew, когда нужна перенастройка для любых изменений, которые могут произойти с выборкой, таких как добавление новых или удаление записей другими рабочими станциями.

Реализация: Метод SetSort вызывает метод ResetFromView.

При необходимости метод ResetFromView перенастраивает указатель

прокрутки. ABC – шаблоны переопределяют метод BrowseClass.ResetFromView для пересчета итогов, если это необходимо.

Пример:

```
BRW1.ResetFromView                PROCEDURE
ForceRefresh:Cnt                    LONG
        CODE
        SETCURSOR(Cursor:Wait)
        SELF.Reset
        LOOP
        CASE SELF.Next()
        OF Level:Notify
        BREAK
        OF Level:Fatal
        RETURN
        END
        SELF.SetQueueRecord
        ForceRefresh:Cnt += 1
        END
        ForceRefresh = ForceRefresh:Cnt
        SETCURSOR()
```

ResetQueue (заполнение или перестроение очереди)

ResetQueue(*режим*), VIRTUAL

ResetQueue Заполняет или перестраивает экранную очередь списка browse.

режим Численная константа, переменная, EQUATE или выражение, которая показывает, как метод ResetQueue определит выделенную запись после заполнения очереди. Значение Reset:Queue выделяет текущий выбранный элемент данных. Значение Reset:Done высвечивает запись на основе текущей позиции VIEW и других факторов, таких как свойство RetainRow.

Метод **ResetQueue** заполняет или перестраивает очередь browse и соответственно делает доступными или недоступными элементы управления изменением, удалением и выбором. Процесс заполнения зависит от значения параметра *режим* и некоторых других свойств BrowseClass, включая свойства ActiveInvisible, AllowUnfilled, RetainRow и т.д.

Значение Reset:Queue параметра *режим* обычно обеспечивает более эффективное заполнение очереди, чем значение Reset:Done.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод ResetQueue вызывает метод Fetch для заполнения очереди.

Константы для *режима* объявлены в файле ABBROWSE.INC следующим образом:

```
        ITEMIZE,PRE(Reset)
Queue           EQUATE
Done            EQUATE
                    END
```

Пример:

```
DeleteMyBrowse  ROUTINE                 !код удаления
MyBrowse.ResetQueue(Reset:Queue)
                                        !после удаления обновим очередь,
MyBrowse.PostNewSelection               !отправим событие Event:NeweSelection
                                        !для корректного обновления окна
```

См. также: ActiveInvisible, AllowUnfilled, RetainRow, ChangeControl, DeleteControl, SelectControl, Fetch

ResetResets (копирует “сторожевые” поля)

ResetResets, PROTECTED

Метод **ResetResets** копирует текущие значения “сторожевых” полей, таким образом любые последующие изменения значений этих полей могут быть обнаружены.

Этот метод имеет атрибут PROTECTED, следовательно может быть вызван только из методов класса BrowseClass или порожденного от него класса.

Метод AddResetField добавляет “сторожевое” поле для порядка сортировки, определенному предыдущим вызовом AddSortOrder. Можно добавлять несколько “сторожевых” полей для каждого порядка сортировки многократным вызовом метода AddResetField.

Пример:

```
MyBrowse.CheckReset  PROCEDURE
IF NOT SELF.Sort.Resets.Equal()                 !если изменилось
```

```
                                    !“сторожевое” поле,
SELF.ResetQueue(Reset:Queue)        !обновить очередь
SELF.ResetResets

                                    !запомнить новые значения “сторожевых”
                                    !полей
        END
```

См. также: AddResetField

ResetSort (применяет порядок сортировки к browse)

ResetSort(*обязательно*), VIRTUAL, PROC

ResetSort Повторно применяет активный порядок сортировки для списка browse.

обязательно Численная константа, переменная, EQUATE или выражение, которое показывает, условно или безусловно перенастраивать browse. Единичное значение (1 или True) перенастраивает безусловно; нулевое значение (0 или False) перенастраивает только если требуют обстоятельства (изменение порядка сортировки, изменение “сторожевых” полей, первая загрузка и т.д.).

Метод **ResetSort** повторно применяет активный порядок сортировки к списку browse и возвращает единичное (1) значение, если порядок сортировки изменился; если порядок сортировки не изменялся, возвращается ноль (0). Любые диапазоны, локаторы или “сторожевые” поля, связанные с порядком сортировки являются доступными.

Совет: Используйте UpdateWindow вслед за ResetSort для обновления и перерисовки ваших ABC BrowseBoxes. Или используйте метод WindowManager.Reset.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод ResetSort вызывает метод SetSort для применения текущего порядка сортировки. Шаблоны ABC переопределяют метод ResetSort для применения порядка сортировки на основе выбранной закладки.

Возвращаемый тип данных: BYTE

Пример:

```
BRW1.ResetSort  FUNCTION(BYTE Force)          !применить подходящую сортировку
CODE
IF CHOICE(?CurrentTab) = 1                    !выбрана 1-я закладка
RETURN SELF.SetSort(1,Force)                  !применить первыый порядок
                                              !сортировки
ELSE                                          !в противном случае
RETURN SELF.SetSort(2,Force)
                                    !применить второй порядок сортировки
END
```

См. также: AddRange, AddResetField, AddSortOrder, SetSort, UpdateWindow

ScrollEnd (прокручивает к первому или последнему элементу данных)

ScrollEnd(*событие*), VIRTUAL, PROTECTED

ScrollEnd Прокручивает browse к первому или последнему элементу.

событие Численная константа, переменная, EQUATE или выражение, которое показывает требуемое действие прокрутки. Допустимыми действиями прокрутки для данного метода являются прокрутка к началу или концу списка.

Метод **ScrollEnd** прокручивает browse к первому или последнему элементу.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROTECTED, следовательно может быть вызван только из методов класса BrowseClass или порожденного от него класса.

Реализация: Метод BrowseClass.TakeScroll вызывает метод ScrollEnd.

Шестнадцатеричное значение EVENT:ScrollTop параметра *событие* прокручивает browse к первой записи. Значение EVENT:ScrollBottom прокручивает browse к последней записи. Соответствующие событиям прокрутки константы объявлены в файле EQUATES.CLW:

```
EVENT:ScrollTop       EQUATE (07H)
EVENT:ScrollBottom  EQUATE (08H)
```

Пример:

```
BrowseClass.TakeScroll PROCEDURE( SIGNED Event )
CODE
IF RECORDS(SELF.ListQueue)
        CASE Event
        OF Event:ScrollUp OROF Event:ScrollDown
        SELF.ScrollOne( Event )
        OF Event:PageUp OROF Event:PageDown
        SELF.ScrollPage( Event )
        OF Event:ScrollTop OROF Event:ScrollBottom
        SELF.ScrollEnd( Event )
        END
        END
```

См. также: TakeScroll

ScrollOne (прокручивает вверх или вниз на один элемент)

ScrollOne(*событие*), VIRTUAL, PROTECTED

ScrollOne	Прокручивает browse вверх или вниз на один элемент.
событие	Численная константа, переменная, EQUATE или выражение, которое показывает требуемое действие прокрутки. Допустимыми действиями прокрутки для данного метода являются прокрутка вверх или вниз на один элемент списка.

Метод **ScrollOne** прокручивает browse вверх или вниз на один элемент.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROTECTED, следовательно может быть вызван только из методов класса BrowseClass или порожденного от него класса.

Реализация: Метод BrowseClass.TakeScroll вызывает метод ScrollOne.

Шестнадцатиричное значение EVENT:ScrollUp параметра *событие* прокручивает browse на один элемент вверх. Значение EVENT:ScrollDown прокручивает browse на один элемент вниз. Соответствующие событиям прокрутки константы объявлены в файле EQUATES.CLW:

```
EVENT:ScrollUp      EQUATE (03H)
EVENT:ScrollDown    EQUATE (04H)
```

Пример:

```
BrowseClass.TakeScroll PROCEDURE( SIGNED Event )
CODE
IF RECORDS(SELF.ListQueue)
        CASE Event
        OF Event:ScrollUp OROF Event:ScrollDown
        SELF.ScrollOne( Event )
        OF Event:PageUp OROF Event:PageDown
        SELF.ScrollPage( Event )
        OF Event:ScrollTop OROF Event:ScrollBottom
        SELF.ScrollEnd( Event )
        END
        END
```

См. также: TakeScroll

ScrollPage (прокручивает вверх или вниз на одну страницу)

ScrollPage(*событие*), VIRTUAL, PROTECTED

ScrollPage Прокручивает browse вверх или вниз на одну страницу.

событие Численная константа, переменная, EQUATE или выражение, которое показывает требуемое действие прокрутки. Допустимыми действиями прокрутки для данного метода являются прокрутка browse вверх или вниз на одну страницу.

Метод **ScrollPage** Прокручивает browse вверх или вниз на одну страницу.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROTECTED, следовательно может быть вызван только из методов класса BrowseClass или порожденного от него класса.

Реализация: Метод BrowseClass.TakeScroll вызывает метод ScrollPage.

Шестнадцатиричное значение EVENT:PageUp параметра *scrollevent* прокручивает browse на одну страницу вверх. Значение EVENT:PageDown прокручивает browse на одну страницу вниз. Соответствующие событиям прокрутки константы объявлены в файле EQUATES.CLW:

```
EVENT:PageUp    EQUATE (05H)
EVENT:PageDown  EQUATE (06H)
```

Пример:

```
BrowseClass.TakeScroll PROCEDURE( SIGNED Event )
        CODE
        IF RECORDS(SELF.ListQueue)
                                    CASE Event
                                    OF Event:ScrollUp OROF Event:ScrollDown
                                       SELF.ScrollOne( Event )
                                    OF Event:PageUp OROF Event:PageDown
                                       SELF.ScrollPage( Event )
                                    OF Event:ScrollTop OROF Event:ScrollBottom
                                       SELF.ScrollEnd( Event )
                                    END
        END
```

См. также: TakeScroll

SetAlerts (задает горячие клавиши для элементов управления списка и локатора)

SetAlerts

Метод **SetAlerts** определяет стандартные горячие клавиши для экранного списка browse и для любых связанных с ним элементов управления локатором.

Метод BrowseClass.TakeKey обрабатывает определенные горячие клавиши.

Реализация: The BrowseClass.SetAlerts method alerts the mouse DOUBLE-CLICK, the INSERT, DELETE and CTRL+ENTER keys for the browse's list control and calls the LocaorClass.SetAlerts method for each associated locator control. Corresponding keycode EQUATEs are declared in KEYCODES.CLW.

Метод BrowseClass.SetAlerts определяет двойное нажатие левой кнопки мыши, клавиши INSERT, DELETE и CTRL+ENTER для экранного списка и вызывает метод LocaorClass.SetAlerts для каждого связанного элемента управления локатором. Соответствующие кодам клавиш константы объявлены в файле KEYCODES.CLW.

Метод BrowseClass.SetAlerts также определяет контекстное меню для каждого списка browse, которое имитирует поведение любых кнопок управления (insert, change, delete, select).

Пример:

```
PrepareStateBrowse  ROUTINE          !настроить объект BrowseClass:
BrowseState.Init(?StateList,|        ! задать экранный список,
```

```
StateQ.Position,|                    ! строку, хранящую позицию VIEW,
StateView, |                         ! собственно VIEW,
StateQ,    |                         ! показываемую очередь,
Relate:State)                        ! RelationManager для первичного файла
BrowseState.SetAlerts                !установить горячие клавиши List-а и локатора
```

См. также: TakeKey

SetQueueRecord (копирует данные из буфера файла в буфер очереди)

SetQueueRecord, VIRTUAL

Метод **SetQueueRecord** копирует соответствующие данные из полей *поле файла* в поля *поле очереди*, описанные методом AddField. Обычно это поля буфера файла и поля буфера очереди списка browse, поэтому буфер очереди соответствует буферам файла.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Методы BrowseClass.Fetch и BrowseClass.Ask вызывают метод SetQueueRecord method.

Пример:

```
MyBrowseClass.SetQueueRecord PROCEDURE
CODE
SELF.Fields.AssignLeftToRight
        !скопировать данные из файла в буфер очереди
SELF.ViewPosition = POSITION( SELF.View )        !установить позицию VIEW
```

См. также: Ask, AddField, Fetch

SetSort (применяет порядок сортировки для browse)

SetSort(*порядок сортировки*, *обязательно перенастраивать* **), VIRTUAL, PROC**

SetSort Применяет заданный порядок сортировки для списка browse.

порядок сортировки Численная константа, переменная, EQUATE или выражение, которое описывает применяемый порядок сортировки.

обязательно перенастраивать

Численная константа, переменная, EQUATE или выражение, которое сообщает методу, обязательно ли перенастраивать browse. Нулевое значение (0 или False) перенастраивает browse только если

обстоятельства требуют этого (изменен порядок сортировки, изменены "сторожевые" поля, при первой загрузке); единичное значение (1 или True) переенастраивает browse безусловно.

Метод **SetSort** применяет заданный *порядок* сортировки к списку browse и возвращает единичное значение (1), если порядок сортировки изменен; и возвращает нулевое значение (0), если порядок сортировки не изменялся.

Обычно значение параметра *порядок сортировки* - это значение, возвращаемое методом AddSortOrder, который идентифицирует конкретный порядок сортировки. Поскольку метод AddSortOrder возвращает номера последовательностей сортировки, единичное значение (1) применяет порядок сортировки, заданный первым вызовом метода AddSortOrder; двойка (2) применяет порядок сортировки, заданный следующим вызовом метода AddSortOrder; и т.д. Нулевое значение применяет порядок сортировки по умолчанию.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод ResetSort вызывает метод SetSort.

Возвращаемый тип данных: BYTE

Пример:

```
IF FIELD() = ?FirstTab                  !выбрана 1-я закладка
IF MyBrowse.SetSort(1,0)                !применить первыый порядок сортировки
MyBrowse.ResetThumbLimits               !если порядок сортировки изменен,
                                        !переустановить
                                        !границы указателя вертикальной прокрутки
END
MyBrowse.UpdateBuffer                   !обновить буфер по текущему элементу
        END
```

См. также: AddRange, AddResetField, AddSortOrder, ResetSort

TakeAcceptedLocator (применяет введенное значение локатора)

TakeAcceptedLocator

Метод **TakeAcceptedLocator** применяет введенное значение локатора к списку browse – объект BrowseClass прокручивает список к запрошенному элементу данных.

Локаторы с вводными полями – это локаторы, значение которых вводится непосредственно. Другие типы локаторов работают по-другому, например, при помощи горячих клавиш. Значения локаторов считаются введенными, когда конечный пользователь уходит с поля при помощи клавиши TAB или переносит фокус с вводного поля локатора иным способом.

Метод AddLocator устанавливает локаторы для browse.

Реализация: Метод TakeAcceptedLocator вызывает соответствующий метод LocatorClass.TakeAccepted.

Пример:

```
IF FIELD() = ?MyLocator                !фокус на поле локатора
IF EVENT() = EVENT:Accepted            !ввод завершен
MyBrowse.TakeAcceptedLocator           !объект BrowseClass обрабатывает его
        END
        END
```

См. также: AddLocator

TakeEvent (обрабатывает текущее событие ACCEPT- цикла)

TakeEvent, VIRTUAL

Метод **TakeEvent** обрабатывает текущее событие ACCEPT- цикла для объекта BrowseClass. Метод TakeEvent обрабатывает все события, связанные со списком browse, исключая событие EVENT:NewSelection. Это событие обрабатывается методом TakeNewSelection.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод WindowManager.TakeEvent вызывает метод TakeEvent. Метод TakeEvent вызывает методы TakeScroll или TakeKey в зависимости от обстоятельств.

Пример:

```
MyWindowManager.TakeEvent PROCEDURE
RVal BYTE(Level:Benign)
I   USHORT,AUTO
 CODE
LOOP I = 1 TO RECORDS(SELF.Browses)
GET(SELF.Browses,I)
SELF.Browses.Browse.TakeEvent
```

```
        END
        LOOP i=1 TO RECORDS(SELF.FileDrops)
                                    GET(SELF.FileDrops,i)
                                    ASSERT(~ERRORCODE())
                                    SELF.FileDrops.FileDrop.TakeEvent
        END
        RETURN RVal
```

См.также: TakeKey, TakeNewSelection, TakeScroll, WindowManager.TakeEvent

TakeKey (обрабатывает нажатия горячих клавиш)

TakeKey, VIRTUAL, PROC

Метод **TakeKey** обрабатывает нажатия горячих клавиш для объекта BrowseClass, включая DOUBLE-CLICK, INSERT, CTRLENTER или DELETE, и возвращает значение, указывающее, какое действие было выполнено.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод TakeKey возвращает единичное значение (1), если было выполнено какое-нибудь действие; и возвращает нулевое значение (0) в противном случае.

Метод TakeEvent вызывает в случае необходимости метод TakeKey. Метод BrowseClass.TakeKey вызывает в случае необходимости метод Locator.TakeKey.

Возвращаемый тип данных: BYTE

Пример:

```
IF FIELD() = ?MyBrowseList          !фокус на списке browse
IF EVENT() EVENT:AlertKey           !нажата горячая клавиша
MyBrowse.TakeKey                    !объект BrowseClass обрабатывает ее
        END
        END
```

См.также: TakeEvent

TakeNewSelection (обрабатывает событие EVENT:NewSelection)

TakeNewSelection, VIRTUAL, PROC

Метод **TakeNewSelection** обрабатывает событие EVENT:NewSelection списка browse и возвращает значение, указывающее, нужно ли обновлять окно.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод TakeNewSelection возвращает единичное значение (1), если необходимо обновить окно, и нулевое значение (0) в противном случае.

Метод TakeEvent вызывает в случае необходимости метод TakeNewSelection. Метод BrowseClass.TakeNewSelection вызывает в случае необходимости метод Locator.TakeNewSelection.

Возвращаемый тип данных: BYTE

Пример:

```
IF FIELD() = ?MyBrowse                !фокус на списке browse
IF EVENT() = EVENT:NewSelection       !выбран новый элемент
MyBrowse.TakeNewSelection()           !объект BrowseClass обрабатывает это
                                      !событие
ELSE                                  !другое событие
MyBrowse.TakeEvent                    !объект BrowseClass обрабатывает его
        END
        END
```

TakeScroll (обрабатывает событие прокрутки)

TakeScroll([*событие*] **), VIRTUAL**

TakeScroll Обрабатывает событие прокрутки для списка browse.

событие Численная константа, переменная, EQUATE или выражение, которое указывает событие прокрутки. Допустимыми событиями прокрутки являются: вверх на один элемент, вниз на один элемент, вверх на одну страницу, вниз на одну страницу, вверх к первому элементу, вниз к последнему элементу. Если параметр опущен, прокрутка не осуществляется.

Метод **TakeScroll** обрабатывает событие прокрутки для списка browse

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Значение EVENT:ScrollUp параметра *событие* прокручивает browse на один элемент вверх; значение EVENT:ScrollDown прокручивает browse на один элемент вниз; значение EVENT:PageUp прокручивает browse на одну страницу вверх; значение EVENT:PageDown прокручивает browse на одну страницу вниз; значение EVENT:ScrollTop прокручивает browse к первому элементу списка; значение EVENT:ScrollBottom прокручивает browse к последнему элементу списка. Соответствующие параметру *событие* константы объявлены в файле EQUATES.CLW.

```
EVENT:ScrollUp      EQUATE (03H)
EVENT:ScrollDown    EQUATE (04H)
EVENT:PageUp        EQUATE (05H)
EVENT:PageDown      EQUATE (06H)
EVENT:ScrollTop     EQUATE (07H)
EVENT:ScrollBottom  EQUATE (08H)
```

Метод TakeScroll вызывает ScrollEnd, ScrollOne или при необходимости метод ScrollPage.

Пример:

```
IF FIELD() = ?MyBrowse                  !фокус на списке browse
CASE EVENT()                            !событие скроллинга
OF EVENT:ScrollUp
OROF EVENT:ScrollDown
OROF EVENT:PageUp
OROF EVENT:PageDown
OROF EVENT:ScrollTop
OROF EVENT:ScrollBottom
MyBrowse.TakeScroll                     !объект BrowseClass обрабатывает его
        END
        END
```

См. также: ScrollEnd, ScrollOne, ScrollPage

TakeVCRScroll (обрабатывает событие VCR-прокрутки)

TakeVCRScroll([*событие*]), VIRTUAL

TakeVCRScroll Обрабатывает событие VCR-прокрутки для списка browse.
vcrevent Численная константа, переменная, EQUATE или выражение, которое описывает событие прокрутки. Допустимыми событиями прокрутки являются: вверх на один элемент, вниз на один элемент, вверх на одну страницу, вниз на одну страницу, вверх к первому элементу, вниз к последнему элементу. Если параметр пропущен, прокрутка не осуществляется.

Метод **TakeVCRSroll** обрабатывает событие VCR-прокрутки для списка browse.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Значение VCR:Forward параметра *событие* прокручивает browse на один элемент вниз; значение VCR:Backward прокручивает browse на один элемент вверх; значение VCR:PageForward прокручивает browse на одну страницу вниз; значение VCR:PageBackward прокручивает browse на одну страницу вверх; значение VCR:Last прокручивает browse к последнему элементу; значение VCR:First scrolls прокручивает browse к первому элементу. Соответствующие параметру *событие* константы объявлены в файле \LIBSRC\ABTOOLBA.INC.

```
                    ITEMIZE,PRE(VCR)
Forward             EQUATE(Toolbar:Down)
Backward            EQUATE(Toolbar:Up)
PageForward         EQUATE(Toolbar:PageDown)
PageBackward EQUATE(Toolbar:PageUp)
First               EQUATE(Toolbar:Top)
Last                EQUATE(Toolbar:Bottom)
Insert              EQUATE(Toolbar:Insert)
None                EQUATE(0)
    END
    END
```

Метод TakeVCRScroll вызывает метод TakeScroll, преобразовывая событие VCR-прокрутки в соответствующее событие прокрутки.

Пример:

```
    LOOP
```

```
        !обработка повторяющихся ссобытий скроллинга
IF VCRRequest = VCR:None                    !больше нет
BREAK                                       !конец цикла
ELSE                                        !иначе
MyBrowse.TakeVCRScroll( VCRRequest )  !объект BrowseClass обрабатывает его
        END
        END
```

См.также: TakeScroll

UpdateBuffer (копирует выбранный элемент данных из буфера очереди в буфер файла)

UpdateBuffer, VIRTUAL

Метод **UpdateBuffer** копирует соответствующие данные из полей *поле очереди* в поля *поле файла*, заданные методом AddField, для текущего выбранного элемента данных. Обычно это поля буфера очереди списка browse и поля буфера файла, поэтому буферы файлов соответствуют текущему выбранному элементу данных списка browse.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Многие методы BrowseClass вызывают метод UpdateBuffer.

Пример:

```
IF FIELD() = ?FirstTab                  !выбрана первая закладка
IF MyBrowse.SetSort(1,0)                !применитьпервый порядок сортировки
MyBrowse.ResetThumbLimits               !если порядок сортировки изменился,
                                        !переустановить
                                        !границы указателя  вертикальной прокрутки
        END
MyBrowse.UpdateBuffer                   !обновить буфер файла по текущему элементу
MyBrowse.UpdateResets                   !обновить буфер файла по “сторожевым”
                                            !полям
        END
```

См. также: AddField

UpdateResets (копирует “сторожевые” поля в буфер файла)

UpdateResets, PROTECTED

Метод **UpdateResets** копирует значения “сторожевых” полей в соответствующие поля буфера файла.

Этот метод имеет атрибут PROTECTED, следовательно может быть вызван только из методов класса BrowseClass или порожденного от него класса.

Метод AddResetField определяет "сторожевые" поля для объекта BrowseClass.

Реализация: Методы BrowseClass.Next и BrowseClass.Previous вызывают метод UpdateResets.

Пример:

```
MyBrowseClass.Next PROCEDURE     !метод класса, производного от BrowseClass
CODE
IF Level:Fatal = PARENT.Next()   !выполнить родительский метод
POST(EVENT:CloseWindow)          !если неудача – завершить
        ELSE                     !иначе
SELF.UpdateResets                обновить буфер файла по "сторожевым"
полям
        END
```

См. также: AddResetField, Next, Previous

UpdateThumb (позиционирует указатель прокрутки)

UpdateThumb

Метод **UpdateThumb** позиционирует указатель прокрутки и управляет доступностью линейки вертикальной прокрутки в зависимости от количества элементов данных в списке browse, текущего выбранного элемента данных и активного метода пошагового распределения. Более подробно о поведении указателя прокрутки см. *Control Templates—BrowseBox*.

Реализация: Метод AddSortOrder устанавливает методы пошагового распределения для объекта BrowseClass.

Пример:

```
IF FIELD() = ?MyBrowse                !фокус на списке browse
IF EVENT() = EVENT:NewSelection       !если выбран новый элемент
IF MyBrowse.TakeNewSelection()        !объект BrowseClass обрабатывает это
MyBrowse.UdateThumb                   !переместить указатель прокрутки
        END
        END
        END
```

UpdateThumbFixed (позиционирует фиксированный указатель прокрутки)

UpdateThumbFixed, PROTECTED

Метод **UpdateThumbFixed** позиционирует фиксированный указатель прокрутки и управляет доступностью линейки вертикальной прокрутки в зависимости от количества элементов данных в списке browse, текущего выбранного элемента данных и активного метода пошагового распределения. Более подробно о поведении указателя прокрутки см. *Control Templates—BrowseBox*.

Этот метод имеет атрибут PROTECTED, следовательно может быть вызван только из методов класса BrowseClass или порожденного от него класса.

Реализация: Метод AddSortOrder устанавливает методы пошагового распределения для объекта BrowseClass.

Пример:

```
MyBrowseClass.UpdateThumb PROCEDURE
CODE
IF SELF.Sort.Thumb &= NULL          !если нет объекта step
SELF.UpdateThumbFixed               !репозиционировать указатель как
                                    !фиксированный
        ELSE                        !репозиционировать указатель в соответствии
                                    !методом распределения
        END
```

UpdateViewRecord (читает данные VIEW для выбранного элемента данных)

UpdateViewRecord, VIRTUAL

Метод **UpdateViewRecord** перечитывает запись VIEW для выбранного элемента данных списка browse, поэтому запись VIEW может быть записана на диск. Метод UpdateViewRecord включает автоматическую обработку оптимистического конкурентного доступа, при котором операция записи (PUT) на диск возвращает ошибку, если другой пользователь изменил данные с тех пор, как они были получены методом UpdateViewRecord.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод UpdateViewRecord использует операторы WATCH и REGET для выполнения автоматической обработки оптимистического конкурентного доступа; более подробно см. *Описание языка*.

Пример:

```
IF FIELD() = ?ChangeButton                !по кнопке “изменить”
IF EVENT() = EVENT:Accepted               !кнопка нажата
MyBrowse.UpdateViewRecord                 !обновить буферы и взвести WATCH
DO MyBrowse:ButtonChange                  !вызвать подпрограмму обновления.
          END
          END
```

UpdateWindow (модифицирует экранные переменные в соответствии с browse)

UpdateWindow

Метод **UpdateWindow** модифицирует экранные переменные для приведения их в соответствие текущему состоянию списка browse.

Совет: Используйте UpdateWindow вслед за ResetSort для обновления и перерисовки Ваших ABC BrowseBoxes. Или используйте метод WindowManager.Reset.

Реализация: Метод BrowseClass.UpdateWindow вызывает соответствующий метод LocatorClass.UpdateWindow, который гарантирует, что поле локатора содержит текущее значение поиска.

Пример:

```
IF FIELD() = ?MyBrowse                    !фокус на списке browse
IF EVENT) = EVENT:NewSelection            !выбран новый элемент
IF MyBrowse.TakeNewSelection()            !объект BrowseClass обрабатывает это
MyBrowse.SetSort(0,1)                     !повторно примеить порядок сортировки
MyBrowse.UpdateBuffer                     !обновить буфер файла по текущему
                                          !элементу
MyBrowse.UpdateWindow                     !обновить экранные переменные (локатор)
DISPLAY()                                 !и перерисовать окно
          .         .         .
```

Свойства EditClass

Класс EditClass дает Вам возможность динамического редактирования по месту каждого столбца (поля) первичного файла в browse. При этом поддерживаются автоинкрементные ключи, диапазоны и ссылочная целостность. В настоящее время проверка значений полей выполняется при сохранении записи, и изменен может быть только первичный файл.

Метод BrowseClass.AskRecord – инструмент для выполнения функции редактирования по месту. Этот метод динамически создает экранные элементы управления по запросу (добавление, изменение или удаление записи) конечного пользователя. Когда пользователь покидает отредактированную запись (нажатием клавиши или щелчком мышью на другом элементе окна), этот метод сохраняет или удаляет запись и ликвидирует экранный созданный ранее элемент управления.

Для осуществления редактирования по месту вам необходимо лишь вызвать метод BrowseClass.Ask. Никаких прямых ссылок на класс EditClass не требуется.

Класс EditClass имеет единственное свойство - FEQ. Это свойство описано ниже.

FEQ (номер элемента управления редактированием по месту)

FEQ UNSIGNED

Свойство **FEQ** содержит номер элемента управления редактированием по месту.

Метод Init устанавливает значение свойства FEQ во время создания элемента управления.

См. также: Init

Методы EditClass

Init (инициализирует объект класса EditClass)

Init(*столбец, экранный список, редактируемое поле***), VIRTUAL**

Init	Инициализирует объект класса EditClass.
столбец	Численная константа, переменная, EQUATE или выражение, которое устанавливает номер редактируемого столбца *экранного списка*.
экранный список	Численная константа, переменная, EQUATE или выражение, которое устанавливает номер экранного элемента управления LIST, в котором показываются записи файла.
редактируемое поле	Полная метка редактируемого поля.

Метод **Init** инициализирует объект класса EditClass.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод EditClass.Init создает элемент управления редактированием по месту, загружает его выбранным элементом данных списка и взводит соответствующие навигационные клавиши редактирования по месту.

Пример:

```
MyEditClass.Init(1,?MyList,StateFile:StateCode)     !иинициализация объекта
EditClass                                           !только для 1-го столбца
                                                    !код программы
MyEditClass.Kill                                    закрыть объект EditClass
```

Kill (закрывает объект класса EditClass)

Kill, VIRTUAL

Метод **Kill** закрывает объект класса EditClass.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод EditClass.Kill ликвидирует элемент управления редактированием по месту, созданный методом EdiClass.Init.

Пример:

```
MyEditClass.Init(1,?MyList,StateFile:StateCode)     !иинициализация объекта
                                                    !EditClass
                                                    !только для 1-го столбца
!код программы
MyEditClass.Kill                                    !закрыть объект EditClass
```

TakeEvent (обрабатывает событие редактирования по месту)

TakeEvent(*событие*), VIRTUAL

TakeEvent	Обрабатывает событие для элемента управления редактированием по месту.
событие	An integer constant, variable, EQUATE, or expression that resolves to the edited column number of the *listbox*.

****

В оригинале, по-видимому опечатка, т. к. из приведенного ниже примера следует, что в качестве параметра данному методу передается **именно событие**. Поэтому предлагается нижеследующий перевод. [прим. переводчика]

****

Численная константа, переменная, EQUATE или выражение, имеющее значение произошедшего события.

Метод **TakeEvent** обрабатывает событие для элемента управления редактированием по месту и возвращает значение, указывающее запрошенное пользователем действие. Допустимыми значениями являются бездействие, следующая запись, предыдущая запись, завершение или OK, отмена, следующее поле и предыдущее поле.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод EditClass.TakeEvent обрабатывает событие EVENT:AlertKey для элемента управления редактированием по месту и возвращает значение, указывающее запрошенное пользователем действие. Соответствующие константы для возможных действий по редактированию по месту объявлены в файле ABBROWSE.INC:

```
EditAction    ITEMIZE(0),PRE
None          EQUATE                ! бездействие
Forward       EQUATE                ! следующее поле
Backward      EQUATE                ! предыдущее поле
Complete      EQUATE                ! OK
Cancel        EQUATE                ! отмена
Next          EQUATE                ! следующая запись
Previous      EQUATE                ! предыдущая запись
END
```

Возвращаемый тип данных: BYTE

Пример:

```
EditClassAction ROUTINE
 CASE SELF.EditList.Control.TakeEvent(EVENT()) !Здесь передано СОБЫТИЕ
 OF EditAction:Forward
          !обработка табуляции вперед (новое поле, та же запись)
 OF EditAction:Backward
          ! обработка табуляции назад (новое поле, та же запись)
```

```
OF EditAction:Next
        !обработка клавиши “стрелка вниз” (новая запись, предложить сохранить
        !отредактированную запись)
OF EditAction:Previous
        !обработка клавиши “стрелка вверх” (новая запись, предложить
        !сохранить отредактированную запись)
OF EditAction:Complete
        !обработка OK или клавиши ENTER (сохранить запись)
OF EditAction:Cancel
        !обработка отмены или клавиши ESC (восстановить оригинальную
        !запись)
END
```

Классы File Drop (выпадающие списки записей файлов)

Обзор

Развитие классов File Drop

Текущая реализация классов File Drop является самостоятельной. В дальнейшем эти классы или те, что придут им на смену, будут порождаться от класса BrowseClass.

Концепции классов File Drop

FileDropClass

Класс FileDropClass представляет собой ViewManager, поддерживающий скроллируемый список в окне, заполняемый записями файлов. FileDrop служит списком выбора для конечного пользователя. Список выбора – это ограниченный список взаимоисключающих или альтернативных вариантов. Конечный пользователь может выбрать лишь один из нескольких пунктов, однако ему не требуется запоминать возможные варианты, так как все они показываются на экране.

На основании выбора конечного пользователя вы можете присвоить значения одному или более полям. Вы можете показать на экране одно поле (например, описание), но присвоить значение другому полю (например, коду) из выбранного элемента списка.

FileDropClass также поддерживает фильтры, диапазоны, использование цветов, пиктограмм, а также выбор нескольких элементов. Более подробно о реализации этих возможностей в шаблонах см. *Шаблоны элементов управления – Выпадающие списки (FileDrop)*.

FileDropComboClass

Класс FileDropComboClass представляет собой FileDropClass, основанный на элементе управления COMBO вместо LISTa. Таким образом, он поддерживает не только выбор из существующего набора вариантов, но так же и *ввод значений, отсутствующих в списке*, и опциональное *добавление новых значений в список*. Более подробно о реализации этих возможностей в шаблонах см. *Шаблоны элементов управления – Комбинированные списки (FileDropCombo)*.

Взаимодействие с другими классами

Как FileDropClass так и FileDropComboClass тесно интегрированы с классом WindowManager. Эти объекты регистрируют присутствие друг друга, устанавливают значения свойств друг друга и, при необходимости, вызывают методы друг друга для достижения своих целей.

Класс FileDropClass порождается от класса ViewManager, а также опирается на другие ABC-классы для выполнения некоторых задач. Таким образом, если в вашей программе объявлены объекты FileDropClass, в ней также должны быть объявлены и эти другие классы. В значительной степени это делается автоматически, когда вы включаете (INCLUDE) заголовок класса FileDropClass (ABDROPS.INC) в секцию данных своей программы. См. *Концептуальный пример*.

Класс FileDropComboClass порожден от класса FileDropClass, поэтому он опирается на те же ABC-классы, а также на ErrorClass. Использование этих классов автоматизируется при включении (INCLUDE) заголовка класса FileDropClass (ABDROPS.INC) в секцию данных своей программы. См. *Концептуальный пример*.

Реализация ABC-шаблонов

ABC-шаблоны автоматически включают все классы и генерируют весь код, необходимый для поддержки функциональности, указанной в шаблонах FileDrop и FileDropCombo вашего приложения.

FileDropClass

Шаблоны *порождают* класс от FileDropClass и создают объект для *каждого* шаблонного элемента управления FileDrop в приложении. Порожденный класс называется *процедура*:FDB#, где *процедура* – имя процедуры, а # - номер экземпляра шаблона элемента управления FileDrop. Поскольку шаблоны предоставляют порожденный класс, вы можете использовать вкладку **Classes** в окне настройки элементов управления FileDrop в целях модификации поведения каждого экземпляра.

Объект порожденного класса называется FDB#. Этот объект поддерживает всю функциональность, указанную в шаблоне FileDrop.

Порожденный класс является локальным по отношению к процедуре, специфическим для данного элемента управления, и опирается на глобальные

объекты RelationManager и FileManager для показываемого файла.

FileDropComboClass

Шаблоны *порождают* класс от FileDropComboClass и создают объект для *каждого* шаблонного элемента управления FileDropCombo в приложении. Порожденный класс называется *процедура*:FDCB#, где *процедура* – имя процедуры, а # - номер экземпляра шаблона элемента управления FileDropCombo. Поскольку шаблоны предоставляют порожденный класс, вы можете использовать вкладку **Classes** в окне настройки элементов управления FileDropCombo в целях модификации поведения каждого экземпляра.

Объект порожденного класса называется FDCB#. Этот объект поддерживает всю функциональность, указанную в шаблоне FileDropCombo.

Порожденный класс является локальным по отношению к процедуре, специфическим для данного элемента управления, и опирается на глобальный объект ErrorClass и объекты RelationManager и FileManager для показываемого файла.

Исходные модули классов FileDrop

Исходные модули FileDropClass по умолчанию устанавливаются в каталог \CLARION4\LIBSRC. Ниже перечислены файлы FileDropClass и соответствующие компоненты:

ABDROPS.INC	объявления FileDropClass объявления FileDropComboClass
ABDROPS.CLW	определения методов FileDropClass определения методов FileDropComboClass

Концептуальный пример

Следующий пример иллюстрирует типичную последовательность операторов, объявляющих, активизирующих, инициализирующих, использующих и завершающих объект FileDropClass.

В этом примере объект FileDropClass используется, чтобы предоставить конечному пользователю возможность выбрать верный код штата для данного клиента. Штат должен выбираться из соответствующего файла.

```
PROGRAM

INCLUDE('ABWINDOW.INC')
INCLUDE('ABDROPS.INC')
MAP
```

```
            END

State
FILE,DRIVER('TOPSPEED'),PRE(ST),THREAD
StateCodeKey KEY(ST:STATECODE),NOCASE,OPT
Record      RECORD,PRE()
StateCode  STRING(2)
StateName STRING(20)
            END
            END

Customer
FILE,DRIVER('TOPSPEED'),PRE(CUS),CREATE,THREAD
BYNUMBER
KEY(CUS:CUSTNO),NOCASE,OPT,PRIMARY
Record      RECORD,PRE()
CUSTNO    LONG
Name        STRING(30)
State       STRING(2)
            END
            END

GlobalErrors  ErrorClass
VCRRequest  LONG(0),THREAD
Access:State   CLASS(FileManager)
Init            PROCEDURE
            END
Relate:State  CLASS(RelationManager)
Init   PROCEDURE
END
Access:Customer CLASS(FileManager)
Init            PROCEDURE
END
Relate:Customer   CLASS(RelationManager)
Init            PROCEDURE
END
StateQ      QUEUE
ST:STATECODE   LIKE(ST:STATECODE)
ViewPosition   STRING(512)
END

StateView  VIEW(State)
END
```

```
CusWindow WINDOW('Add Customer'),AT(,,157,58),IMM,SYSTEM,GRAY
PROMPT('Customer:'),AT(5,7),USE(?NamePrompt)
ENTRY(@s20),AT(61,5,88,11),USE(CUS:NAME)
PROMPT('State:'),AT(5,22),USE(?StatePrompt)
LIST,AT(61,20,65,11),USE(CUS:State),FROM(StateQ),|
FORMAT('8L~STATECODE~@s2@'),DROP(5)
BUTTON('OK'),AT(60,39),USE(?OK),DEFAULT
BUTTON('Cancel'),AT(104,39),USE(?Cancel)
END
ThisWindow CLASS(WindowManager)
Init
PROCEDURE(),BYTE,PROC,VIRTUAL
Kill
PROCEDURE(),BYTE,PROC,VIRTUAL
END
StateDrop  CLASS(FileDropClass)
Q &StateQ
END
 CODE
 ThisWindow.Run()
ThisWindow.Init PROCEDURE()
ReturnValue       BYTE,AUTO
 CODE
 GlobalErrors.Init
 Relate:State.Init
 Relate:Customer.Init
 SELF.Request = InsertRecord
 ReturnValue = PARENT.Init()
 IF ReturnValue THEN RETURN ReturnValue.
 SELF.FirstField = ?CUS:NAME
 SELF.VCRRequest &= VCRRequest
 SELF.Errors &= GlobalErrors
 SELF.AddUpdateFile(Access:Customer)
 SELF.AddItem(?Cancel,RequestCancelled)
 SELF.OkControl = ?OK
 Relate:Customer.Open
 Relate:State.Open
 SELF.Primary &= Relate:Customer
 SELF.InsertAction = Insert:Batch
 IF SELF.PrimeUpdate() THEN RETURN Level:Notify.
 OPEN(CusWindow)
 SELF.Opened=True
```

```
 StateDrop.Init(?CUS:State,StateQ.ViewPosition,StateView,StateQ,Relate:State,ThisWindow)
 StateDrop.Q &= StateQ
 StateDrop.AddSortOrder()
 StateDrop.AddField(ST:STATECODE,StateDrop.Q.ST:STATECODE)
 StateDrop.AddUpdateField(ST:STATECODE,CUS:State)
 ThisWindow.AddItem(StateDrop)
 SELF.SetAlerts()
 RETURN ReturnValue

ThisWindow.Kill PROCEDURE()
ReturnValue        BYTE,AUTO
 CODE
 ReturnValue = PARENT.Kill()
 IF ReturnValue THEN RETURN ReturnValue.
 Relate:Customer.Close
 Relate:State.Close
 Relate:State.Kill
 Relate:Customer.Kill
 GlobalErrors.Kill
 RETURN ReturnValue

Access:State.Init PROCEDURE
 CODE
 PARENT.Init(State,GlobalErrors)
 SELF.FileNameValue = 'State'
 SELF.Buffer &= ST:Record
 SELF.LazyOpen = False
 SELF.AddKey(ST:StateCodeKey,'ST:StateCodeKey',0)

Access:Customer.Init PROCEDURE
 CODE
 PARENT.Init(Customer,GlobalErrors)
 SELF.FileNameValue = 'Customer'
 SELF.Buffer &= CUS:Record
 SELF.Create = True
 SELF.LazyOpen = False
 SELF.AddKey(CUS:BYNUMBER,'CUS:BYNUMBER',0)

Relate:State.Init PROCEDURE
 CODE
 Access:State.Init
 PARENT.Init(Access:State,1)
```

```
Relate:Customer.Init PROCEDURE
 CODE
 Access:Customer.Init
 PARENT.Init(Access:Customer,1)
```

Свойства FileDropClass

Класс FileDropClass наследует все свойства класса ViewManager, от которого он порожден. Более подробно см. *Свойства ViewManager.*

Кроме унаследованных свойств, FileDropClass содержит свойства, перечисленные ниже.

InitSyncPair (начальная позиция списка)

InitSyncPair BYTE

Свойство **InitSyncPair** управляет начальной позицией выпадающего списка. Единичное значение (1 или True) позиционирует список на элемент, наиболее близкий к текущему значению поля, для которого производится выбор. Нулевое значение (0 или False) позиционирует список на первый элемент списка при указанном порядке сортировки.

Реализация: Метод Init устанавливает значение свойства InitSyncPair в единицу (1). Метод ResetQueue ведет себя в соответствии со значением свойства InitSyncPair.

См. также: Init, ResetQueue

Методы FileDropClass

Класс FileDropClass наследует все методы класса ViewManager, от которого он порожден. Более подробно см. *Методы класса ViewManager.*

Помимо унаследованных методов (или вместо них), класс FileDropClass содержит методы, перечисленные ниже.

Функциональная организация - ожидаемое использование

В качестве помощи для понимания FileDropClass, можно разделить многообразные методы этого класса на две большие категории в соответствии с их ожидаемым использованием - первичный интерфейс и виртуальные методы. Такое структурирование отражает то, что мы понимаем под типичным использованием методов FileDropClass.

Методы первичного интерфейса

Методы первичного интерфейса, которые вы, скорее всего, явно вызываете в своих программах, могут в свою очередь быть разделены на три категории:

Разового применения:

Init	инициализирует объект класса FileDropClass
AddField	определяет показываемые поля
AddUpdateField	определяет поля присваивания
AddRange[I]	добавляет ограничивающий диапазон для текущего порядка сортировки
AppendOrder[I]	уточняет активный порядок сортировки
Kill	закрывает объект класса FileDropClass

Основного применения:

ResetQueue	заполняет или дополняет очередь выпадающего списка
TakeEvent	обрабатывает текущее событие ACCEPT-цикла

Эпизодического применения:

Open[I]	открывает VIEW для выпадающего списка
PrimeRecord[I]	подготавливает запись для добавления
SetFilter[I]	задает фильтр для текущего порядка сортировки
ApplyFilter[I]	устанавливает диапазон и фильтр для результирующей выборки
ApplyOrder [I]	сортирует результирующую выборку
GetFreeElementName[I]	возвращает имя поля свободного элемента
SetOrder[I]	заменяет текущую последовательность сортировки
Close[I]	закрывает VIEW для выпадающего списка

[I] Эти методы унаследованны от класса ViewManager Class.

Виртуальные методы

Обычно эти методы не вызываются непосредственно – их вызывают методы первичного интерфейса. Однако мы предполагаем, что у вас возникнет необходимость переопределить эти методы, и, поскольку они являются виртуальными, сделать это несложно. В противном случае, и их стандартное поведение представляется разумным.

SetQueueRecord	копирует данные из буфера файла в буфер очереди
Reset[I]	репозиционирует VEW
ValidateRecord	проверяет текущий элемент выборки

[I] Эти методы унаследованы от класса ViewManager Class.

AddField (задает показываемые поля)

AddField(*поле файла, поле очереди* **)**

AddField Идентифицирует соответствующие поля файла и очереди для столбца выпадающего списка.

поле файла Полная метка поля файла. Является источником данных для LIST-а выпадающего списка.

поле очереди Полная метка поля QUEUE. *Поле очереди* заполняется значением из *поля файла* и является непосредственным источником данных LIST-а выпадающего списка.

Метод **AddField** определяет соответствующие поля FILE и QUEUE для столбца выпадающего списка записей файлов. Вы должны вызвать AddField для каждого столбца, показываемого в выпадающем списке.

Вы также можете использовать метод AddField для просмотра переменных из памяти, указывая метку переменной в качестве параметра *поле файла*.

Реализация: Метод AddField использует FieldPairsClass для управления заданными парами полей.

Пример:

```
CODE

StFD.Init(?CLI:StCode,StateQ.Pos,StateView,StateQ,Relate:States,ThisWindow)
        StFD.Q &= StateQ
        StFD.AddSortOrder(StCodeKey)
        StFD.AddField(STFile:StCode,StFD.Q.StCode)
        StFD.AddField(STFile:StName,StFD.Q.StName)
        StFD.AddUpdateField(STFile:StCode,CLI:StCode)
```

AddUpdateField (задает поля присваивания)

AddUpdateField(*источник, адресат* **)**

AddUpdateField Задает поле-*источник* и соответствующее ему поле- *адресат*.

источник Полная метка поля, из которого происходит копирование, когда конечный пользователь выбирает элемент выпадающего списка.

адресат Полная метка поля, в которое происходит копирование, когда конечный пользователь выбирает элемент выпадающего списка

Метод **AddUpdateField** идентифицирует поле-*источник* и соответствующее ему поле-*адресат*, которое получает значение поля-*источника*, когда конечный пользователь выбирает элемент выпадающего списка.

Вы можете вызывать AddUpdateField многократно, чтобы выполнить присваивания нескольким полям, когда пользователь сделал выбор.

Реализация: Метод AddUpdateField использует FieldPairsClass для управления заданными парами полей.

Метод TakeEvent выполняет заданное копирование.

Пример:

```
CODE

StFD.Init(?CLI:StCode,StateQ.Pos,StateView,StateQ,Relate:States,ThisWindow)
        StFD.Q &= StateQ
        StFD.AddSortOrder(StCodeKey)
        StFD.AddField(STFile:StCode,StFD.Q.StCode)
        StFD.AddField(STFile:StName,StFD.Q.StName)
        StFD.AddUpdateField(STFile:StCode,CLI:StCode)
```

См. также: TakeEvent

Init (инициализирует объект класса FileDropClass)

Init(*список, позиция view, view, очередь, relationmanager , windowmanager* **)**

Init Инициализирует объект класса FileDropClass.

список Численная константа, переменная, EQUATE или выражение, содержащее номер управляющего элемента LIST выпадающего списка.

позиция view Метка строковой переменной в структуре *очереди,* содержащая POSITION VIEW.

view Метка VIEW, показываемая в выпадающем списке.

очередь Метка QUEUE для *списка.*

relationmanager

Метка объекта RelationManager первичного файла выпадающего списка. Более подробно см. *Relation Manager.*

windowmanager

Метка объекта WindowManager объекта FileDrop. Более подробно см. *Window Manager.*

Метод **Init** инициализирует объект класса FileDropClass.

Реализация: Помимо прочего, метод Init вызывает метод PARENT.Init (ViewManager.Init) для инициализации объекта ViewManager выпадающего списка. Более подробно см. *ViewManager*.

Пример:

```
CODE
StFD.Init(?CLI:StCode,StateQ.Pos,StateView,StateQ,Relate:States,ThisWindow)
```

```
StFD.Q &= StateQ
StFD.AddSortOrder(StCodeKey)
StFD.AddField(STFile:StCode,StFD.Q.StCode)
StFD.AddField(STFile:StName,StFD.Q.StName)
StFD.AddUpdateField(STFile:StCode,CLI:StCode)
```

См. также: ViewManager.Init

Kill (закрывает объект класса FileDropClass)

Kill, VIRTUAL

Метод **Kill** очищает всю память, выделенную в течение жизни объекта FileDropClass и выполняет любой другой требуемый код завершения.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Среди прочего, метод Kill вызывает метод PARENT.Kill (ViewManager.Kill) для закрытия объекта ViewManager выпадающего списка. Более подробно см. *View Manager*.

Пример:

```
CODE

StFD.Init(?CLI:StCode,StateQ.Pos,StateView,StateQ,Relate:States,ThisWindow)
StFD.Q &= StateQ
StFD.AddSortOrder(StCodeKey)
StFD.AddField(STFile:StCode,StFD.Q.StCode)
StFD.AddField(STFile:StName,StFD.Q.StName)
StFD.AddUpdateField(STFile:StCode,CLI:StCode)
!код процедуры
StFD.Kill
```

См. также: ViewManager.Kill

ResetQueue (заполняет очередь выпадающего списка)

ResetQueue([*обязательно*]), VIRTUAL, PROC

ResetQueue Заполняет или обновляет экранную очередь выпадающего списка.

обязательно Численная константа, переменная, EQUATE или выражение, которая показывает, обновлять ли очередь, даже если отсортированная

последовательность не изменялась. Единичное значение (1 или True) означает безусловное обновление очереди; нулевое значение (0 или False) обновляет очередь только если этого требуют обстоятельства. Если параметр опущен, значение *обязательно* принимается равным нулю.

Метод **ResetQueue** заполняет или обновляет очередь выпадающего списка, используя требуемый порядок сортировки, диапазон и фильтр, затем возвращает значение, указывающее, какая запись файла выбора (если такая существует) соответствует значениям полей- *адресатов* (заданных методом AddUpdateField). Нулевое (0) возвращаемое значение свидетельствует об отсутствии соответствующих элементов данных; любое другое значение указывает позицию соответствующего элемента данных.

Например, если выпадающий список служит для выбора кода штата, и текущий код штата заказчика уже имеет допустимое значение, метод ResetQueue условно (базируясь на свойстве InitSyncPair) позиционирует список на текущее значение кода штата заказчика.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно его можно вызывать как процедуру, игнорируя возвращаемый результат.

Тип результата: LONG

Пример:

```
        ACCEPT
IF EVENT() = EVENT:OpenWindow
 StateFileDrop.ResetQueue
END
        !код программы
        END
```

См. также: InitSyncPair

SetQueueRecord (копирует данные из буфера файла в буфер очереди)

SetQueueRecord, VIRTUAL

Метод **SetQueueRecord** копирует соответствующие данные из *полей файла* в *поля очереди*, заданные методом AddField. Обычно это поля буфера файла и поля буфера очереди выпадающего списка, поэтому буфер очереди соответствует буферам файлов.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод ResetQueue вызывает метод SetQueueRecord.

Пример:

```
MyFileDropClass.SetQueueRecord PROCEDURE
 CODE
 SELF.ViewPosition=POSITION(SELF.View)
 SELF.DisplayFields.AssignLeftToRight
        !здесь следует ваш код
```

См. также: ResetQueue

TakeEvent (обрабатывает текущее событие ACCEPT- цикла)

TakeEvent, VIRTUAL

Метод **TakeEvent** обрабатывает текущее событие ACCEPT- цикла для объекта класса FileDropClass.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод WindowManager.TakeEvent вызывает метод TakeEvent. Метод TakeEvent вызывает метод TakeNewSelection.

Пример:

```
MyWindowManager.TakeEvent PROCEDURE
RVal BYTE(Level:Benign)
I   USHORT,AUTO
 CODE
        !код процедуры
        LOOP I = 1 TO RECORDS(SELF.Browses)
                                GET(SELF.Browses,I)
                                SELF.Browses.Browse.TakeEvent
        END
        LOOP i=1 TO RECORDS(SELF.FileDrops)
                                GET(SELF.FileDrops,i)
                                ASSERT(~ERRORCODE())
                                SELF.FileDrops.FileDrop.TakeEvent
```

```
    END
    RETURN RVal
```

См. также: TakeNewSelection, WindowManager.TakeEvent

TakeNewSelection (обрабатывает события EVENT:NewSelection)

TakeNewSelection(*поле*), VIRTUAL

TakeNewSelection	обрабатывает событие EVENT:NewSelection .
поле	Численная константа, переменная, EQUATE или выражение, содержащее номер элемента управления, сгенерировавшего событие EVENT:NewSelection.

Метод **TakeNewSelection** обрабатывает событие EVENT:NewSelection для объекта FileDropClass.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод ResetQueue и метод TakeEvent вызывают метод TakeNewSelection. Если элемент управления LIST объектов класса FileDropClass генерирует событие EVENT:NewSelection, метод TakeNewSelection выполняет присваивания полей, заданные методом AddUpdateField, или очищает поля-*адресаты*, если этот выбор неверен.

Пример:

```
FileDropClass.TakeEvent PROCEDURE

 CODE
 CASE EVENT()
 OF EVENT:NewSelection
  SELF.TakeNewSelection(FIELD())
 END
```

См. также: AddUpdateField, ResetQueue, TakeEvent

ValidateRecord (виртуальный метод для проверки записей)

ValidateRecord, VIRTUAL

Метод **ValidateRecord** вызывается, когда объект класса FileDropClass заполняет показываемую очередь. ValidateRecord возвращает значение, сообщающее включена ли текущая запись в показываемый список. Таким образом, ValidateRecord создает фильтрующий механизм, являющийся дополнением методу ViewManager.SetFilter. Допустимые возвращаемые значения включают:

Record:OK	запись включена
Record:OutOfRange	запись исключена (вне диапазона)
Record:Filtered	запись исключена (отфильтрована)

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод ResetQueue вызывает метод ValidateRecord. Метод ValidateRecord вызывает метод PARENT.ValidateRecord (ViewManager.ValidateRecord).

Константы для возвращаемых значений объявлены в файле \LIBSRC\TPLEQU.CLW:

```
Record:OK                    EQUATE(0)    !Запись в диапазоне
                                          !и удовлетворяет условиям
                                          !фильтра
Record:OutOfRange            EQUATE(1)    !Запись вне диапазона
Record:Filtered              EQUATE(2)    !Запись не удовлетворяет
                                          !условиям фильтра
```

Тип результата: BYTE

Пример:

```
MyFileDropClass.ResetQueue PROCEDURE
i LONG
 CODE
 SETCURSOR(CURSOR:Wait)
 FREE(SELF.ListQueue)
 SELF.ApplyRange
 SELF.Reset
 LOOP UNTIL SELF.Next()
  IF SELF.ValidateRecord()=Record:OK          !Проверка записей
   SELF.SetQueueRecord
   ADD(SELF.ListQueue)
   ASSERT(~ERRORCODE())
   IF SELF.UpdateFields.Equal()
    i=RECORDS(SELF.ListQueue)
   END
  END
 END
        !код процедуры
```

См. также: ResetQueue, ViewManager.SetFilter, ViewManager.ValidateRecord

Свойства класса FileDropComboClass

FileDropComboClass наследует все свойства класса FileDropClass, от которого он порожден. Более подробно см. *Свойства класса FileDropClass*.

UseField (USE-переменная элемента управления COMBO)

UseField ANY, PROTECTED

Свойство **UseField** является ссылкой на USE-переменную элемента управления COMBO. FileDropComboClass использует это свойство для поиска текущего значения в очереди.

Это свойство имеет атрибут PROTECTED, следовательно обращаться к нему можно только из методов класса FileDropComboClass или порожденного от него класса.

Иниициализация: Метод Init инициализирует значение свойства UseField.
См. также: Init

Методы класса FileDropComboClass

Класс FileDropComboClass наследует все методы класса FileDropClass, от которого он порожден, так же, как и методы класса ViewManager, от которого порожден класс FileDropClass. Более подробно см. *Методы класса FileDropClas* и *Методы класса ViewManager*.

Помимо унаследованных методов (или вместо них), класс FileDropClass содержит методы, рассматриваемые ниже.

Функциональная организация - ожидаемое использование

В качестве помощи для понимания, FileDropComboClass можно разделить многообразные методы этого класса на две большие категории в соответствии с их ожидаемым использованием - первичный интерфейс и виртуальные методы. Такое структурирование отражает то, что мы понимает под типичным использованием методов FileDropComboClass.

Методы первичного интерфейса

Методы первичного интерфейса, которые вы, скорее всего, явно вызываете в своих программах, могут в свою очередь быть разделены на три категории:

Разового применения:

Init	инициализирует объект класса FileDropComboClass
AddField[I]	определяет показываемые поля
AddUpdateField[I]	определяет поля присваивания
AddRange[II]	определяет ограничивающий диапазон для активного порядка сортировки
AppendOrder[II]	уточняет порядок сортировки
Kill[I]	закрывает объект класса FileDropComboClass

Основного применения:

ResetQueue	обновляет очередь выпадающего списка
GetQueueMatch	ищет элемент списка
Ask[V]	добавляет запись в файл выбора
TakeEvent	обрабатывает текущее событие ACCEPT- цикла

Эпизодического применения:

Open[II]	открывает VIEW выпадающего списка
PrimeRecord[II]	подготавливает запись для добавления
SetFilter[II]	задает фильтр для активного порядка сортировки
ApplyFilter[II]	устанавливает диапазон и фильтр для результирующей выборки
ApplyOrder[II]	сортирует результирующую выборку
GetFreeElementName[II]	возвращает имя поля свободного элемента
SetOrder[II]	заменяет активный порядок сортировки
Close[II]	закрывает VIEW выпадающего списка

[I] Этот метод унаследован от класса FileDropClass.
[II] Этот метод унаследован от класса ViewManager.

Виртуальные методы

Обычно эти методы не вызываются непосредственно – их вызывают методы первичного интерфейса. Однако мы предполагаем, что у вас возникнет необходимость переопределить эти методы, и, поскольку они являются виртуальными, сделать это несложно. В противном случае, и их стандартное поведение представляется разумным.

Ask	добавляет запись в файл выбора
SetQueueRecord[I]	копирует данные из буфера файла в буфер очереди
Reset[II]	репозиционирует VIEW
ValidateRecord[I]	проверяет текущий элемент результирующей выборки

[I] Этот метод унаследован от класса FileDropClass.
[II] Этот метод унаследован от класса ViewManager.

Ask (добавление записи в файл выбора)

Ask, VIRTUAL, PROTECTED

Метод **Ask** добавляет новую запись в файл выбора выпадающего списка и возвращает значение, сообщающее об успешном завершении операции или неудаче. В случае успеха он возвращает Level:Benign, в противном случае он возвращает уровень серьезности последней ошибки, произошедшей во время попытки добавления записи. Более подробно об уровнях серьезности ошибок см. *Error Class*.

Данный метод является виртуальным, следовательно другие методы базового класса могут непосредственно обращаться к виртуальному методу Ask порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROTECTED, следовательно может быть вызван только из методов класса FileDropComboClass или порожденного от него класса.

Реализация: Метод TakeEvent вызывает метод Ask. Константы для возвращаемых значений объявлены в файле ABERROR.INC (более подробно см. *Error Class*):

```
Level:Benign          EQUATE(0)
Level:User            EQUATE(1)
Level:Program         EQUATE(2)
Level:Fatal           EQUATE(3)
Level:Cancel          EQUATE(4)
Level:Notify          EQUATE(5)
```

Возвращаемый тип данных: BYTE

Пример:

```
MyFileDropComboClass.TakeEvent PROCEDURE
UserStr    CSTRING(256),AUTO
 CODE                                  !код процедуры
IF SELF.Ask() = Level:Benign           !изменение файла выбора

SELF.UpdateFields.AssignLeftToRight
SELF.Close
SELF.ResetQueue
SELF.ListField{PROP:Selected} = SELF.GetQueueMatch(UserStr)
DISPLAY(SELF.ListField)
          END
!код процедуры
```

См. также: TakeEvent

GetQueueMatch (поиск элемента списка)

GetQueueMatch(*значение поиска* **), PROTECTED**

GetQueueMatch Ищет заданное *значение поиска* в первом поле экранной очереди
значение поиска Строковая константа, переменная, EQUATE или выражение, содержащее разыскиваемое значение

Метод **GetQueueMatch** ищет значение в первом поле экранной очереди и возвращает позицию соответствующего элемента списка. Возврат нулевого значения (0) означает отсутствие такого элемента.

Параметр *case* метода Init определяет требуемый тип поиска (чувствительный к регистру или нет).

Этот метод имеет атрибут PROTECTED, следовательно может быть вызван только из методов класса FileDropComboClass или порожденного от него класса.

Тип результата: LONG

Пример:

```
MyFileDropComboClass.TakeEvent PROCEDURE
UserStr    CSTRING(256),AUTO
 CODE
 CASE EVENT()
 OF EVENT:Accepted
UserStr=CLIP(SELF.UseField)
IF SELF.GetQueueMatch(UserStr) = 0        !если введенное значения
SELF.Reset                                !не содержится в файле/очереди выбора
IF SELF.Ask()=Level:Benign                !обновим файл

SELF.UpdateFields.AssignLeftToRight
SELF.Close
SELF.ResetQueue

SELF.ListField{PROP:Selected}=SELF.GetQueueMatch(UserStr)
                                          !позиция нового элемента данных
DISPLAY(SELF.ListField)
END
!код процедуры
```

См. также: Init

Init (инициализирует объект FileDropComboClass)

Init(*use-переменная, combo, позиция, view, очередь, relationmgr, windowmgr, errormgr* [,*добавлять*] [,*перечитывать*] [,*регистр*] **)**

Init	Инициализирует объект FileDropComboClass.
use-переменная	Метка USE-переменной элемента управления *combo*.
combo	Численная константа, переменная, EQUATE или выражение, содержащее номер COMBO для выпадающего списка.
позиция	Метка строкового поля *очереди*,содержащего позицию *view*.
view	Метка VIEW, показываемой в выпадающем списке.
очередь	Метка QUEUE, сслужащей источником данных для *combo*.
relationmgr	Метка объекта RelationManager для первичного файла выпадающего списка. Более подробно см. *Relation Manager*.
windowmgr	Метка объекта WindowManager выпадающего списка. Более подробно см. *Window Manager*.
errormgr	Метка объекта ErrorClass выпадающего списка. Более подробно см. *Error Manager*.
добавлять	Численная константа, переменная, EQUATE или выражение, указывающее, могут ли быть в файл выбора добавлены записи. Нулевое значение (0 or False) запрещает добавление; единичное (1or True) -разрешает. Если параметр опущен, значение по умолчанию принимается равным единице (1).
перечитывать	Численная константа, переменная, EQUATE или выражение, указывающее, следует ли перечитывать запись очереди при навигации по списку (поддержка “горячих” полей). Единичное значение (1or True) включает перечитывание, нулевое (0 or False) – отключает. Если параметр опущен, значение по умолчанию принимается равным единице (1).
регистр	Численная константа, переменная, EQUATE или выражение, указывающее должен ли поиск в выпадающем списке быть чувствительным к регистру. Единичное значение (1or True) требует чувствительного к регистру поиска, нулевое значение (0 or False) делает поиск нечувствительным к регистру. Если параметр опущен, значение по умолчанию принимается равным нулю (0).

Метод **Init** инициализирует объект FileDropComboClass.

Реализация: Помимо прочего, метод Init вызывает метод PARENT.Init (FileDropClass.Init) для инициализации объекта ViewManager выпадающего списка.

Пример:

```
ThisWindow.Init PROCEDURE
        CODE
```

```
                                    !код процедуры
                                    ! Инициализировать объект filedropcombo
FDBC4.Init( CLI:StateCode,|         ! USE-переменная
?CLI:StateCode,|                    ! элемент управления COMBO
Queue:FileDropCombo.ViewPosition,|  ! VIEW POSITION
FDCB4::View:FileDropCombo,|         ! VIEW
Queue:FileDropCombo,|               ! QUEUE
Relate:States,
          |                         ! объект RelationManager
ThisWindow,
|                                   ! объект WindowManager
GlobalErrors,                       ! объект ErrorClass
1,
|                                   ! разрешить добавление
1,
                                    ! обновлять горячие поля при навигации
0)                                  ! нечувствительный к регистру поиск
 FDBC4.Q &= Queue:FileDropCombo
 FDBC4.AddSortOrder()
 FDBC4.AddField(ST:StateCode,FDBC4.Q.ST:StateCode)
 FDBC4.AddField(ST:State,FDBC4.Q.ST:State)
 FDBC4.AddUpdateField(ST:StateCode,CLI:StateCode)
```

См. также: FileDropClass.Init

ResetQueue (обновляет очередь выпадающего списка)

ResetQueue([*обязательно*] **), VIRTUAL, PROC**

ResetQueue Обновляет очередь выпадающего списка и USE-переменную COMBO.

обязательно Численная константа, переменная, EQUATE или выражение, указывающее, обновлять ли очередь, если не изменился порядок сортировки. При единичном значение (1or True), очередь обновляется безусловно, при нулевом (0 or False) - только если обстоятельства требуют этого. Если параметр опущен, его значение принимается равным нулю (0).

Метод **ResetQueue** обновляет очередь выпадающего списка и USE-переменную COMBO, используя текущий порядок сортировки, диапазон и фильтр, затем возвращает значение, указывающее, какая запись в файле выбора (если она существует) соответствует значениям полей- *адресатов,* определенных методом AddUpdateField. Возвращение нулевого значения (0) свидетельствует об отсутствии соответствующих записей; любое другое значение указывает позицию соответствующего элемента.

Например, если выпадающий список служит для выбора кода штата, и текущий код штата заказчика уже имеет допустимое значение, метод ResetQueue условно (базируясь на свойстве InitSyncPair) позиционирует список на текущее значение кода штата заказчика.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Этот метод имеет атрибут PROC, следовательно его можно вызывать как процедуру, игнорируя возвращаемый результат.

Реализация: Метод TakeEvent вызывает метод ResetQueue. ResetQueue вызывает метод PARENT.ResetQueue, затем делает доступной или нет кнопку выпадающего списка в зависимости от наличия списка элементов выбора.

Тип результата: LONG

Пример:

```
MyFileDropComboClass.TakeEvent PROCEDURE
UserStr    CSTRING(256),AUTO
 CODE
 CASE EVENT()
 OF EVENT:Accepted
UserStr=CLIP(SELF.UseField)
IF SELF.GetQueueMatch(UserStr) = 0        ! если введенного значения
SELF.Reset                                ! нет в списке выбора
IF SELF.Ask()=Level:Benign                ! обновим файл выбора
SELF.UpdateFields.AssignLeftToRight
SELF.Close
SELF.ResetQueue(1)                        ! перенааполним очередь
SELF.ListField{PROP:Selected}=SELF.GetQueueMatch(UserStr)
                                          !позиция нового элемента
DISPLAY(SELF.ListField)
END
                                          !код процедуры
```

См. также: TakeEvent, FileDropClass.ResetQueue

TakeEvent (обрабатывает текущее событие ACCEPT-цикла)

TakeEvent, VIRTUAL

Метод **TakeEvent** обрабатывает текущее событие ACCEPT-цикла для объекта FileDropComboClass .

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод WindowManager.TakeEvent вызывает методTakeEvent. При навигации по списку, метод TakeEvent вызывает метод TakeNewSelection.

При значении EVENT:Accepted для вводной части COMBO, метод TakeEvent вызывает метод GetQueueMatch, для поиска ближайшего к введенному значению элемента списка. Если введенного значения нет в списке выбора, метод TakeEvent вызывает метод Ask для добавления нового значения в файл выбора. Если добавление прошло успешно, метод TakeEvent вызывает метод ResetQueue для обновления экранной очереди.

Пример:

```
MyWindowManager.TakeEvent PROCEDURE
RVal BYTE(Level:Benign)
I   USHORT,AUTO
 CODE      !код процедуры
           LOOP I = 1 TO RECORDS(SELF.Browses)
                                    GET(SELF.Browses,I)
                                    SELF.Browses.Browse.TakeEvent
           END
           LOOP i=1 TO RECORDS(SELF.FileDrops)
                                    GET(SELF.FileDrops,i)
                                    ASSERT(~ERRORCODE())
                                    SELF.FileDrops.FileDrop.TakeEvent
           END
           RETURN RVal
```

См. также: Ask, GetQueueMatch, ResetQueue, TakeNewSelection, WindowManager.TakeEvent

TakeNewSelection (обрабатывает событие EVENT:NewSelection)

TakeNewSelection(*поле*), VIRTUAL

TakeNewSelection	Обрабатывает событие EVENT:NewSelection.
поле	Численная константа, переменная, EQUATE или выражение, содержащее номер элемента управления, сгенерировавшего событие EVENT:NewSelection.

Метод **TakeNewSelection** обрабатывает событие EVENT:NewSelection для объекта класса FileDropComboClass.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод ResetQueue и метод TakeEvent вызывают метод TakeNewSelection. Если элемент управления COMBO объекта FileDropComboClass сгенерировал событие EVENT:NewSelection, метод TakeNewSelection выполняет присваивание полей, заданных методом AddUpdateField или зачищает поля-адресаты, если не сделан верный выбор.

Пример:

```
FileDropComboClass.TakeEvent PROCEDURE

 CODE
 CASE EVENT()
 OF EVENT:NewSelection
  SELF.TakeNewSelection(FIELD())
  SELF.WindowManager.Reset
 END
```

См. также: AddUpdateField, ResetQueue, TakeEvent

Классы Локатора

Обзор

Концепции класса LocatorClass

Класс LocatorClass является абстрактным классом – он не может использоваться сам по себе. Однако, от него порождаются другие полезные классы, и некоторые структуры (такие, как BrowseClass и FileDropClass) используют его для ссылки на любые производные от него типы.

Классы, порожденные от LocatorClass, позволяют вам задать элемент управления локатором и поле сортировки, по которому происходит поиск (свободный элемент ключа) для каждого порядка сортировки в browse или VIEW. Кроме того, класс помогает пролистывать записи в поиске запрошенных элементов данных.

Классы, порожденные от LocatorClass, реализуют несколько общих вариантов элементов управления локатором (никакой, STRING, ENTRY), активизации локатора (нажатием клавиши, по клавише ENTER, по клавише TAB) и методов поиска (по единичному символу, по наращиваемой последовательности символов, по особому поиску), которые встречаются в контексте browse.

Пошаговый локатор

Пошаговый локатор является односимвольным локатором, не требующим элемента управления локатором. Когда фокус находится на BrowseBox, и пользователь вводит символ, экранное поле списка продвигается на первую запись, в которой поле сортировки начинается с данного символа (или большего, если нет ключа, сопоставимого с символом локатора). Повторный ввод того же символа продвигает список на следующую запись, в которой поле сортировки начинается с данного символа.

Пошаговый локатор используется, когда поле сортировки имеет тип STRING, CSTRING, или PSTRING, и когда вы хотите, чтобы поиск произошел непосредственно после нажатия клавиши конечным пользователем. Пошаговые локаторы не подходят для цифровых ключей.

Вводной локатор

Вводной локатор – это многосимвольный локатор, который активизируется, когда элемент управления локатором *завершен* (по событию Event:Accepted, а не после каждого нажатия клавиши). Элемент управления локатором может быть ENTRY, COMBO или SPIN.

Когда конечный пользователь помещает один или более символов в элемент управления локатором, затем *завершает* элемент управления нажатием клавиши TAB, нажатием кнопки локатора или выбором другого элемента управления на экране, экранное поле списка продвигается на ближайшую соответствующую запись.

Вводной локатор используется, когда нужно искать записи по цифровым или алфавито-цифровым ключам и отложить поиск до полного ввода пользователем значения локатора. Такой отложенный поиск снижает сетевой трафик и обеспечивает более "ровную" работу в среде клиент-сервер.

Инкрементные (наращиваемые) локаторы

Инкрементный локатор является многосимвольным локатором, с необязательным (но настоятельно рекомендуемым) экранным элементом управления.

Инкрементный локатор используется в случае поиска цифровых или алфавито-цифровых ключей, когда поиск должен произойти сразу же после нажатия клавиши пользователем.

Элемент управления локатором может быть STRING, ENTRY, COMBO или SPIN, однако, любой элемент управления, кроме STRING, заставляет инкрементный локатор вести себя также, как и вводной локатор – поиск откладывается до завершения ввода.

Когда элемент управления имеет тип STRING и фокус находится на списке, символы автоматически помещаются в строку локатора при каждом нажатии клавиши, и экранное поле списка *немедленно* продвигается на ближайшую соответствующую запись. Нажатие клавиши backspace (забой) удаляет символы из строки локатора.

Мы настоятельно рекомендуем использовать элемент управления STRING в качестве элемента управления Инкрементным локатором по следующим причинам:

Поскольку поиск происходит *немедленно* с каждым нажатием клавиши, и

Поскольку пользователь может *видеть* значение ключа, по которому происходит поиск в BrowseBox.

Фильтрующие локаторы

Фильтрующий локатор является многосимвольным локатором, с необязательным (но настоятельно рекомендуемым) экранным элементом управления.

Фильтрующий локатор используется при поиске алфавито-цифровых ключей, когда желательно *минимизировать сетевой трафик*.

Этот локатор похож на Инкрементный локатор с диапазоном или фильтром. Он описывает *диапазон* значений, для которого происходит поиск, и возвращает *ограниченную* результирующую выборку – возвращаются только те записи, которые попадают в описанный диапазон. Каждый дополнительный ввденный символ поиска приводит к меньшей, более точной результирующей выборке. Например, поисковое значение 'A' возвращает все записи от'AA' до 'AZ'; поисковое значение 'AB' возвращает все записи от 'ABA' до'ABZ', и т.д.

Фильтрующий локатор определяет границы поиска на основе описанного пользователем значения поиска. Реализация границ зависит от базы данных --для баз данных SQL Фильтрующий локатор использует выражение LIKE; для баз данных ISAM он применяет верхний и нижний пределы.

Локатор возвращает *только те* записи, которые соответствуют значению поиска, обеспечивая эффективный, динамический диапазон или фильтр для browse.

Совет: Фильтрующий локатор очень хорошо работает с базами данных SQL и со старшими ключевыми полями; однако, производительность может пострадать, когда используются неключевые или младшие ключевые поля баз данных, не поддерживающих SQL.

Взаимоотношение с другими ABC-классами

Классы, порожденные от класса LocatorClass используются классами BrowseClass и FileDropClass.Таким образом, если в вашей программе объявлены объекты BrowseClass или FileDropClass, в ней также должен быть объявлен класс LocatorClass. В значительной степени это делается автоматически, когда вы

включаете (INCLUDE) заголовки классов BrowseClass или FileDropClass (ABBROWSE.INC или ABDROPS.INC) в раздел данных своей программы. См. *Концептуальный пример.*

Реализация ABC -шаблонов

ABC-шаблоны автоматически включают все классы, необходимые для объявления классов локатора в ваших Browses и Droplists.

Исходные модули класса LocatorClass

Исходные модули LocatorClass по умолчанию устанавливаются в каталог \CLARION4\LIBSRC. Ниже перечислены файлы LocatorClass и соответствующие компоненты:

ABBROWSE.INC	Объявления класса LocatorClass
	Объявления класса StepLocatorClass
	Объявления класса EntryLocatorClass
	Объявления класса IncrementalLocatorClass
	Объявления класса FilterLocatorClass
ABBROWSE.CLW	Определение методов LocatorClass
	Определение методов StepLocatorClass
	Определение методов EntryLocatorClass
	Определение методов IncrementalLocatorClass
	Определение методов FilterLocatorClass

Концептуальный пример

Следующий пример иллюстрирует типичную последовательность операторов, объявляющих, активизирующих, инициализирующих, использующих и завершающих объект BrowseClass и связанные объекты, включая объект локатор. Пример инициализирует и постранично загружает список LIST, затем обрабатывает ряд характерных событий, включая скроллинг, обновление и поиск записей.

Обратите внимание, что объекты WindowManager и BrowseClass самостоятельно обрабатывают обычные события локатора.

```
  PROGRAM
  INCLUDE('ABWINDOW.INC')                    !объявить класс WindowManager
  INCLUDE('ABBROWSE.INC')                    ! объявить класс BrowseClass
  MAP
  END
State
FILE,DRIVER('TOPSPEED'),PRE(ST),THREAD
StateCodeKey KEY(ST:STATECODE),NOCASE,OPT
Record      RECORD,PRE()
```

```
    STATECODE                       STRING(2)
    STATENAME                       STRING(20)
                                        END
                                    END

    StView              VIEW(State)
                                    !объявить обрабатываемую VIEW
                                    END

    StateQ              QUEUE

!объявить очередь для показа
    ST:STATECODE        LIKE(ST:STATECODE)
    ST:STATENAME        LIKE(ST:STATENAME)
    ViewPosition        STRING(512)
                                    END

    GlobalErrors    ErrorClass
                                !объявить объект GlobalErrors
    Access:State    CLASS(FileManager)
                        !объявить объект Access:State
    Init                            PROCEDURE
                                    END
    Relate:State    CLASS(RelationManager)
            !объявить объект Relate:State
    Init                            PROCEDURE
                                    END
    VCRRequest          LONG(0),THREAD

    StWindow    WINDOW('Browse States'),AT(,,123,152),IMM,SYSTEM,GRAY

LIST,AT(8,5,108,124),USE(?StList),IMM,HVSCROLL,FROM(StateQ),|

FORMAT('27L(2)|M~CODE~@s2@80L(2)|M~STATENAME~@s20@')

BUTTON('&Insert'),AT(8,133),USE(?Insert)

BUTTON('&Change'),AT(43,133),USE(?Change),DEFAULT
```

```
BUTTON('&Delete'),AT(83,133),USE(?Delete)
END
ThisWindow CLASS(WindowManager)                !объявить объект ThisWindow
Init
PROCEDURE(),BYTE,PROC,VIRTUAL
Kill
PROCEDURE(),BYTE,PROC,VIRTUAL
END
BrowseSt  CLASS(BrowseClass)          !объявить объект BrowseSt
Q &StateQ
END
StLocator  StepLocatorClass           !объявить объект StLocator
StStep     StepStringClass            !объявить объект StStep
           CODE
           ThisWindow.Run()           !запустить процедуру обработки окна
ThisWindow.Init   PROCEDURE()         !инициализация
ReturnValue BYTE,AUTO
           CODE
           ReturnValue = PARENT.Init()          !вызов метода инициализации
                                                !базового класса
           IF ReturnValue THEN RETURN ReturnValue.
           GlobalErrors.Init          !инициализация объекта GlobalErrors
           Relate:State.Init          !инициализация объекта Relate:State
           SELF.FirstField = ?StList  !установить первое поле для ThisWindow
           SELF.VCRRequest &= VCRRequest
                                      !VCRRequest не используется
           SELF.Errors &= GlobalErrors
                                      !установить обработчик ошибок
                                      !для ThisWindow
           Relate:State.Open          !открыть State и связанные файлы
           !Init BrowseSt object by naming its LIST,VIEW,Q,RelationManager &
WindowManager
           BrowseSt.Init(?StList,StateQ.ViewPosition,StView,StateQ,Relate:State,SELF)
           OPEN(StWindow)
           SELF.Opened=True
           BrowseSt.Q &= StateQ       !установить ссылку на QUEUE !просмотра
           StStep.Init(+ScrollSort:AllowAlpha,ScrollBy:Runtime)
                                      !инициализация объекта StStep
           BrowseSt.AddSortOrder(StStep,ST:StateCodeKey)
                                      !установить порядок сортировки
           BrowseSt.AddLocator(StLocator)
                                      !включить локатор
           StLocator.Init(,ST:STATECODE,1,BrowseSt)
```

```
                                        !инициализировать локатор
BrowseSt.AddField(ST:STATECODE,BrowseSt.Q.ST:STATECODE)
                                        !установить столбец просмотра
BrowseSt.AddField(ST:STATENAME,BrowseSt.Q.ST:STATENAME)
                                        !установить столбец просмотра
BrowseSt.InsertControl=?Insert
                                        !установить элемент управления для
                                        !добавления записей
BrowseSt.ChangeControl=?Change
                                        !установить элемент управления для
                                        !изменения записей
BrowseSt.DeleteControl=?Delete
                                        !установить элемент управления для
                                        !удаления записей
SELF.SetAlerts()
                                        !взвести горячие клавиши для ThisWindow
RETURN ReturnValue
ThisWindow.Kill  PROCEDURE()            !завершить все
ReturnValue  BYTE,AUTO
             CODE
             ReturnValue = PARENT.Kill()    !вызов метода базового класса
             IF ReturnValue THEN RETURN ReturnValue.
             Relate:State.Close             !закрыть Stateи связанные файлы
             Relate:State.Kill              !закрыть объект Relate:State
             GlobalErrors.Kill              !закрыть объект GlobalErrors
             RETURN ReturnValue
Access:State.Init  PROCEDURE
             CODE
             PARENT.Init(State,GlobalErrors)
             SELF.FileNameValue = 'State'
             SELF.Buffer &= ST:Record
             SELF.AddKey(ST:StateCodeKey,'ST:StateCodeKey',0)
Relate:State.Init  PROCEDURE
             CODE
             Access:State.Init
             PARENT.Init(Access:State,1)
```

Свойства LocatorClass

Класс LocatorClass имеет ряд свойств, описанных ниже. Эти свойства наследуются классами, порожденными из класса LocatorClass.

Control (номер элемента управления локатором)

Control SIGNED

Свойство **Control** содержит номер элемента управления локатора, если таковой существует. Если элемента управления локатора не существует, свойство содержит нулевое значение (0). Класс LocatorClass использует свойство Control для обновления элемента управления или изменения его свойств.

Метод Init устанавливает значение свойства Control.
См. также: Init

FreeElement (первый свободный элемент ключа локатора)

FreeElement ANY

Свойство **FreeElement** содержит ссылку на компонент сортировочной последовательности набора данных, в котором осуществляется поиск. Шаблонам ABC в дальнейшем требуется существование свободного компонента ключа. Свободным компонентом считается компонент, не ограниченный по диапазону единственным значением. Обычно это также USE –переменная элемента управления локатором. LocatorClass использует свойство FreeElement для присваивания свободному элемента соответствующего значения поиска.

Метод Init устанавливает значение свойства FreeElement.
См. также: Init

NoCase (флаг поиска с учетом регистра)

NoCase BYTE

Свойство **NoCase** определяет, должен объект класса LocatorClass выполнять поиск с учетом или без учета регистра символов.

Метод Init устанавливает значение свойства NoCase.

Реализация: Если свойство NoCase содержит ненулевое значение, поиск осуществляется без учета регистра. Так поиск для “Tx,” “tx,” или “TX” дадут одинаковый результат. Если свойство NoCase содержит нулевое значение (0), поиск происходит с учетом регистра.
См. также: Init

ViewManager (объект ViewManager локатора)

ViewManager &BrowseClass

Свойство **ViewManager** является ссылкой на объект класса BrowseClass для данного объекта LocatorClass. Более подробно см. *ViewManager* и *BrowseClass*. Класс LocatorClass использует это свойство для манипуляции набором данных, в котором осуществляется поиск, и экранным полем LIST.

Метод Init устанавливает значение свойства ViewManager.

См. также: Init

Методы класса LocatorClass

Init (инициализирует объект LocatorClass)

Init([*элемент управления*] , *свободный элемент, без учета регистра* [,*browseclass*] **)**

Init	Инициализирует объект LocatorClass.
элемент управления	Численная константа, переменная, EQUATE или выражение, которое устанавливает номер элемента управления локатора для объекта LocatorClass. Если параметр опущен, номер элемента управления принимается по умолчанию равным нулю (0), что указывает отсутствие элемента управления локатора.
свободный элемент	Полная метка компонента сортировочной последовательности набора данных поиска. Шаблонам ABC в дальнейшем требуется существование свободного компонента ключа. Свободным компонентом считается компонент, не ограниченный по диапазону единственным значением. Обычно это также USE –переменная элемента управления локатором. LocatorClass использует свойство FreeElement для присваивания свободному элемента соответствующего значения поиска.
без учета регистра	Численная константа, переменная, EQUATE или выражение, которое определяет должен ли объект LocatorClass выполнять поиск с учетом или без учета регистра символов.
browseclass	Метка объекта BrowseClass для локатора. Если параметр опущен, объект LocatorClass не имеет прямой доступ к QUEUE или лежащей в ее основе VIEW.

Метод **Init** инициализирует объект LocatorClass.

Реализация: Метод Init устанавливает значения свойств Control, FreeElement, NoCase и ViewManager.

Нулевое значение (0 или False) параметра *без учета регистра* задает поиск с учетом регистра; единичное значение (1 или True) - без учета регистра символов.

По умолчанию, только StepLocatorClass и FilterLocatorClass используют параметр *browseclass*. Другие классы локатора не используют его.

Пример:

```
BRW1::Sort1:Locator.Init(,CUST:StateCode,1)      !без экранного элемента
                                                 !управления
BRW1::Sort2:Locator.Init(?CUST:CustMo,CUST:CustNo,1)
                                                 !с экранным элементом управления
```

См. также: Control, FreeElement, NoCase, ViewManager

Reset (переустанавливает локатор для следующего поиска)

Reset, VIRTUAL

Метод **Reset** является виртуальным "должностным" методом для переустановки локатора для следующего поиска.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод Reset является виртуальным "должностным" методом для переустановки локатора для следующего поиска. Метод BrowseClass.TakeAcceptedLocator вызывает метод Reset.

Пример:

```
BrowseClass.TakeAcceptedLocator PROCEDURE
                                     !обработка завершения ввода локатора
CODE
IF ~SELF.Sort.Locator &= NULL AND ACCEPTED() = SELF.Sort.Locator.Control
IF SELF.Sort.Locator.TakeAccepted()
                                     !вызов метода TakeAccepted локатора
SELF.Reset(1)                        !если требуется  поиск, переустановить VIEW
SELECT(SELF.ListControl)
                                     !установить фокус на экранный список browse
SELF.ResetQueue( Reset:Done )        !перенабрать очередь просмотра
SELF.Sort.Locator.Reset              !переустановить локатор
SELF.UpdateWindow                    !перерисовать окно
```

```
        END
        END
        END
```

Set (рестартует локатор)

Set, VIRTUAL

Метод **Set** подготавливает локатор для следующего поиска.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод Set очищает свойство FreeElement.

Пример:

```
MyBrowseClass.TakeScroll PROCEDURE( SIGNED Event )
CODE
CASE Event
OF Event:ScrollUp OROF Event:ScrollDown
SELF.ScrollOne( Event )
OF Event:PageUp OROF Event:PageDown
SELF.ScrollPage( Event )
OF Event:ScrollTop OROF Event:ScrollBottom
SELF.ScrollEnd( Event )
END                                          !после события скроллинга
IF ~SELF.Sort.Locator &= NULL THEN
                                             !если наличествует локатор
SELF.Sort.Locator.Set                        !очистить его
        END
```

SetAlerts (устанавливает горячие клавиши для элемента управления LIST)

SetAlerts(*эл-т управления* **), VIRTUAL**

SetAlerts Устанавливает соответствующие горячие клавиши для указанного элемента управления.

эл-т управления Численная константа, переменная, EQUATE или выражение, которое содержит номер элемента управления, отображающего данные для поиска.

Метод **SetAlerts** устанавливает соответствующие горячие клавиши для указанного элемента управления, обычно для LIST или COMBO.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод SetAlerts является "должностным" методом для классов, порожденных из класса LocatorClass—IncrementalLocatorClass, и т.д.
См. также: IncrementalLocatorClass.SetAlerts

SetEnabled (делает доступным или недоступным элемент управления локатора)

SetEnabled(*доступность* **)**

SetEnabled	Делает доступным или недоступным элемент управления локатора.
доступность	Численная константа, переменная, EQUATE или выражение, которое делает доступным или недоступным элемент управления локатора. Нулевое значение (0 или False) делает элемент управления недоступным; единичное значение (1 или True) делает элемент управления доступным.

Метод **SetEnabled** делает доступным или недоступным элемент управления локатора для данного объекта LocatorClass. См. *ENABLE* и *DISABLE* в *Language Reference*.

Пример:

```
MyBrowseClass.Enable PROCEDURE
CODE
IF ~SELF.Sort.Locator &= NULL          !если локатор присутствует
SELF.Sort.Locator.SetEnabled(RECORDS(SELF.ListQueue))
                                       !сделать недоступным, если !нет зписей
END
```

TakeAccepted (обрабатывает введенное значение локатора)

TakeAccepted, VIRTUAL

Метод **TakeAccepted** обрабатывает введенное значение локатора и возвращает значение, указывающее должно ли измениться отображение списка browse. Данный метод подходит только для объектов LocatorClass с элементами управление локатора, допускающими пользовательский ввод (например, элементы управления ENTRY, COMBO или SPIN).

Значение локатора считается введенным, когда конечный пользователь уходит с поля при помощи клавиши TAB или переносит фокус на другой элемент управления в этом же окне иным способом.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод TakeAccepted является "должностным" методом для классов, порожденных из класса LocatorClass— EntryLocatorClass, FilterLocatorClass, и т.д.

Возвращаемый тип данных: BYTE
См. также: EntryLocatorClass.TakeAccepted, FilterLocatorClass.TakeAccepted

TakeKey (обрабатывает нажатия горячих клавиш)

TakeKey, VIRTUAL

Метод **TakeKey** обрабатывает нажатия горячих клавиш для элементов управления LIST или COMBO и возвращает значение, указывающее должно ли измениться отображение списка browse.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод TakeKey является "должностным" методом для классов, порожденных из класса LocatorClass— StepLocatorClass, EntryLocatorClass, IncrementalLocatorClass и т.д.

Возвращаемый тип данных: BYTE
См. также: StepLocatorClass.TakeKey, EntryLocatorClass.TakeKey, IncrementalLocatorClass.TakeKey

UpdateWindow (перерисовывает элемент управления локатора с его текущим значением)

UpdateWindow, VIRTUAL

Метод **UpdateWindow** является "должностным" методом для перерисовки элемента управления локатора с его текущим значением.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных

классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод UpdateWindow является должностным методом для классов, порожденных из класса LocatorClass—IncrementalLocatorClass, FilterLocatorClass и т.д.

См. также: IncrementalLocatorClass.UpdateWindow, FilterLocatorClass.UpdateWindow

Свойства класса StepLocatorClass

StepLocatorClass наследует все свойства LocatorClass, от которого он порожден. Более подробно см. *LocatorClass Properties* и *LocatorClass Concepts*.

Методы StepLocatorClass

StepLocatorClass наследует все методы LocatorClass, от которого он порожден. Более подробно см. *LocatorClass Methods* и *LocatorClass Concepts*.

Помимо унаследованных методов (или вместо них), класс StepLocatorClass содержит методы, пречисленные ниже:

Set (рестартует локатор)

Set, VIRTUAL

Метод **Set** подготавливает локатор для нового поиска.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод ничего не выполняет, потому что каждый новый поиск пошагового локатора переустанавливает FreeElement локатора.

TakeKey (обрабатывает нажатия горячих клавиш)

TakeKey, VIRTUAL

Метод **TakeKey** обрабатывает обрабатывает нажатия горячих клавиш для элементов управления LIST или COMBO и возвращает значение, указывающее должно ли измениться отображение списка browse.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод TakeKey присваивает свойству FreeElement соответствующее значение поиска, затем возвращает единицу (1), если требуется новый поиск или возвращает ноль (0), если нового поиска не требуется. Поиск требуется только если горячая клавиша означает допустимый символ поиска.

Возвращаемый тип данных: BYTE

Пример:

```
IF SELF.Sort.Locator.TakeKey()          ! обработке нажатия клавиши
  SELF.Reset(1)                         !    если требуется поиск, переустановим VIEW
  SELF.ResetQueue( Reset:Done )         ! и очередь просмотра
  END
```

См. также: FreeElement

Свойства EntryLocatorClass

EntryLocatorClass наследует все свойства LocatorClass, от которого он порожден. Более подробно см. *LocatorClass Properties* и *LocatorClass Concepts*.

Методы EntryLocatorClass

EntryLocatorClass наследует все методы LocatorClass, от которого он порожден. Более подробно см. *LocatorClass Methods* и *LocatorClass Concepts*.

Помимо унаследованных методов (или вместо них), класс EntryLocatorClass содержит методы, пречисленные ниже:

TakeAccepted (обрабатывает введенное значение локатора)

TakeAccepted, VIRTUAL

Метод **TakeAccepted** обрабатывает введенное значение локатора и возвращает значение, указывающее должно ли измениться отображение списка browse.

Значение локатора считается введенным, когда конечный пользователь уходит с поля при помощи клавиши TAB или переносит фокус на другой элемент управления в этом же окне иным способом.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод TakeAccepted присваивает свойству FreeElement соответствующее значение поиска, затем возвращает единицу (1), если требуется новый поиск, или ноль (0), если новый поиск не требуется.

Возвращаемый тип данных: BYTE

Пример:

```
MyBrowseClass.TakeAcceptedLocator PROCEDURE
CODE
IF ~SELF.Sort.Locator &= NULL          !если приссутствует локатор
IF SELF.Sort.Locator.TakeAccepted()    !если значение локатора требует поиска
SELF.Reset(1)                          !репозиционировать VIEW
SELECT(SELF.ListControl)               !перенести фркус на LIST
SELF.ResetQueue( Reset:Done )          !переустановить очередь промотра
SELF.Sort.Locator.Reset                !переутановить USE-переменную локатора
END
END
```

См. также: FreeElement

TakeKey (обрабатывает нажатия горячих клавиш)

TakeKey, VIRTUAL

Метод **TakeKey** обрабатывает определенные горячие клавиши для элементов управления LIST или COMBO, которые отображают данные для поиска.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод TakeKey передает нажатия горячих клавиш от элемента управления локатора элементу управления LIST или COMBO, которые отображают данные для поиска, затем возвращает ноль (0).

Возвращаемый тип данных: BYTE

Пример:

```
MyEntryLocator.TakeKey FUNCTION
KeyVal STRING(1),AUTO
        CODE                            !здесь ваш код
        RETURN (PARENT.TakeKey)
```

Свойства IncrementalLocatorClass

IncrementalLocatorClass наследует все свойства EntryLocatorClass, от которого он порожден. Более подробно см. *EntryLocatorClass Properties* и *LocatorClass Concepts*.

Помимо унаследованных свойств, класс EntryLocatorClass содержит свойства, перечисленные ниже:

Shadow (значение поиска)

Shadow CSTRING(40)

Свойство **Shadow** содержит значение поиска для инкрементного локатора.

Метод TakeKey добавляет или убирает символы из значения поиска на основе клавиш, нажатых конечным пользователем.

См. также: TakeKey

Методы IncrementalLocatorClass

IncrementalLocatorClass наследует все методы EntryLocatorClass, от которого он порожден. Более подробно см. *EntryLocatorClass Methods* и *LocatorClass Concepts*.

Помимо унаследованных методов (или вместо них), класс IncrementalLocatorClass содержит методы, перечисленные ниже:

Init (инициализирует объект IncrementalLocatorClass)

Init([*эл-т управления*] ,*свободный элемент, без учета регистра* [,*browseclass*] **)**

Init	Инициализирует объект IncrementalLocatorClass.
эл-т управления	Численная константа, переменная, EQUATE или выражение, которое устанавливает для локатора экранный элемент управления. Если параметр опущен, номер элемента управления принимается по умолчанию равным нулю (0), что указывает отсутствие элемента управления локатора.
свободный элемент	Полная метка компонента сортировочной последовательности набора данных поиска. Шаблонам ABC в дальнейшем требуется

	существование свободного компонента ключа. Свободным компонентом считается компонент, не ограниченный по диапазону единственным значением. Обычно это также USE –переменная элемента управления локатором. LocatorClass использует свойство FreeElement для присваивания свободному элемента соответствующего значения поиска.
без учета регистра	Численная константа, переменная, EQUATE или выражение, которое определяет, должен ли локатор выполнять поиск с учетом или без учета регистра.
browseclass	Метка объекта BrowseClass для локатора. Если параметр опущен, объект LocatorClass не имеет прямого доступа к QUEUE или лежащей в ее основе VIEW.

Метод **Init** инициализирует объект IncrementalLocatorClass.

Реализация: Метод Init устанавливает значения свойств Control, FreeElement, NoCase и ViewManager. Свойство Shadow является USE- переменной *эл-та управления*.

Нулевое значение (0 или False) параметра *без учета регистра* выполняет поиск с учетом регистра; единичное значение (1 или True) выполняет поиск без учета регистра.

По умолчанию, только StepLocatorClass и FilterLocatorClass используют параметр *browseclass*. Другие классы локатора не используют этот параметр.

Пример:

```
BRW1::Sort1:Locator.Init(,CUST:StateCode,1)
                                    !без элемента управления
BRW1::Sort2:Locator.Init(?CUST:CustMo,CUST:CustNo,1)
                                    !с элемеентом управления
```

See Also: Control, FreeElement, NoCase, ViewManager

Set (рестартует локатор)

Set, VIRTUAL

Метод **Set** подготавливает локатор для нового поиска.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод Set очищает свойства FreeElement и Shadow.

Пример:

```
MyBrowseClass.TakeScroll PROCEDURE(SIGNED Event)
                                        !обработка события скроллинга
        CODE
        CASE Event                      !определим, какое именно событие
        OF Event:ScrollUp OROF Event:ScrollDown
        SELF.ScrollOne( Event )         !прокрутка на один элемент
        OF Event:PageUp OROF Event:PageDown
        SELF.ScrollPage( Event )        !прокрутка на одну страницу
        OF Event:ScrollTop OROF Event:ScrollBottom
        SELF.ScrollEnd( Event )         !на первую или последнюю страницу
        END
        SELF.PostNewSelection           !передадим событие  EVENT:NewSelection
                                        !экранному списку (LIST)
        IF ~SELF.Sort.Locator &= NULL
                                        !если имеется локатор
        SELF.Sort.Locator.Set
                                        !очистим его
        END
        IF SELF.Sort.Thumb &= NULL
                                        !если имеется индикатор скроллинга
        SELF.UpdateThumbFixed
                                        !репозиционируем его
        END
```

См. также: FreeElement, Shadow

SetAlerts (устанавливает горячие клавиши для элемента управления LIST)

SetAlerts(*эл-т управления* **), VIRTUAL**

SetAlerts	Устанавливает соответствующие горячие клавиши для указанного элемента управления.
эл-т управления	Численная константа, переменная, EQUATE или выражение, которое содержит номер элемента управления LIST или COMBO, отображающих данные для поиска.

Метод **SetAlerts** устанавливает соответствующие горячие клавиши для указанного элемента управления LIST или COMBO.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод SetAlerts устанавливает в качесчтве горячих клавиш Backspace и пробел.

Пример:

```
MyBrowseClass.SetAlerts PROCEDURE
                                        !горячие клавиши для объекта browse
I BYTE,AUTO
CODE
LOOP I = 1 TO RECORDS( SELF.Sort )      !для каждого порядка сортировки
GET( SELF.Sort, I )
IF ~ ( SELF.Sort.Locator &= NULL )      !если есть локатор
SELF.Sort.Locator.SetAlerts( SELF.ListControl )
                                        !вызовем метод Locator.SetAlerts
END
END
```

TakeKey (обрабатывает нажатия горячих клавиш)

TakeKey, VIRTUAL

Метод **TakeKey** обрабатывает нажатия горячих клавиш для элемента управления LIST или COMBO, которые отображают данные для поиска.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод TakeKey добавляет или убирает символы из значения поиска (свойство Shadow) на основе клавиш, нажатых конечным пользователем, затем возвращает единицу (1), если требуется новый поиск, или ноль (0), если новый поиск не требуется. Поиск требуется только если горячая клавиша означает допустимый символ поиска.

Возвращаемый тип данных: BYTE

Пример:

```
CheckLocator ROUTINE
IF SELF.Sort.Locator.TakeKey()
                                        !обработка горячих клавиш локатора
SELF.Reset(1)                           !если требуется поиск, переустановим VIEW
SELF.ResetQueue(Reset:Done)             соответствующую очередь
ELSE                                    !если поиск не нужен
SELF.ListControl{PROP:Selected}=SELF.CurrentChoice
```

```
                                    !выделим выбранный элемент
                                    !экранного списка
        END
```

См. также: Shadow

UpdateWindow (перерисовывает элемент управления локатора с его текущим значением)

UpdateWindow, VIRTUAL

Метод **UpdateWindow** перерисовывает элемент управления локатора с его текущим значением.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод UpdateWindow присваивает свойству FreeElement текущее значение поиска (свойство Shadow), затем перерисовывает экранный элемент управления.

Пример:

```
MyBrowseClass.UpdateWindow PROCEDURE
                                    !обновление связанных с browse элементов
                                    !управления
CODE
IF ~(SELF.Sort.Locator &= NULL)
                                    !если есть локатор
SELF.Sort.Locator.UpdateWindow
                                    !перерисовать его
        END
```

См. также: FreeElement, Shadow

Свойства FilterLocatorClass

Класс FilterLocatorClass наследует все свойства класса IncrementalLocatorClass, от которого он порожден. Более подробно см. *IncrementalLocatorClass Properties* и *LocatorClass Concepts*.

Помимо унаследованных свойств, класс FilterLocatorClass содержит свойства, перечисленные ниже:

FloatRight (флаг “содержит” или “начинается с”)

FloatRight BYTE

Свойство **FloatRight** определяет, должен FilterLocator применять значение поиска ко всему полю (поле *содержит* значение поиска) или только к крайним левым символам поля (поле *начинается со* значения поиска). Единичное значение (1 или True) применяет проверку “содержит”; нулевое значение (0 или False) применяет проверку “начинается с”.

Класс FilterLocatorClass не инициализирует свойство FloatRight, поэтому FloatRight по умолчанию принимается равным нулю.

Реализация: Метод UpdateWindow реализует действие, описанное свойством FloatRight.

Пример: FilterLocator при поиске для “ba” возвращает следующее:

```
FloatRight=False          FloatRight=True
Bain                      Bain
Barber                    Barber
Bayert                    Bayert
                          Dunbar
                          Suba
```

См. также: UpdateWindow

Методы FilterLocatorClass

Класс FilterLocatorClass наследует все методы класса IncrementalLocatorClass, от которого он порожден. Более подробно см. *IncrementalLocatorClass Methods* и *LocatorClass Concepts*.

Помимо унаследованных методов (или вместо них), класс FilterLocatorClass содержит методы, перечисленные ниже:

TakeAccepted (обрабатывает введенное значение локатора)

TakeAccepted, VIRTUAL

Метод **TakeAccepted** обрабатывает введенное значение локатора и возвращает значение, указывающее должно ли измениться отображение списка browse. Данный метод подходит только для объектов LocatorClass с элементами управление локатора, допускающими пользовательский ввод (например, элементы управления ENTRY,

COMBO или SPIN).

Значение локатора считается введенным, когда конечный пользователь уходит с поля при помощи клавиши TAB или переносит фокус на другой элемент управления в этом же окне иным способом.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод TakeAccepted присваивает свойству FreeElement значение поиска, затем возвращает единицу (1), если требуется новый поиск, или ноль (0), если новый поиск не требуется.

Возвращаемый тип данных: BYTE

Пример:

```
BrowseClass.TakeAcceptedLocator PROCEDURE
                                        !обработка ввода значения локатора
CODE
IF ~SELF.Sort.Locator &= NULL AND ACCEPTED() = SELF.Sort.Locator.Control
IF SELF.Sort.Locator.TakeAccepted()
                                        !вызов метода TakeAccepted локатора
SELF.Reset(1)                           !если требуется поиск, переустановим VIEW
SELECT(SELF.ListControl)
                                        !переместим фркус на экранный список
                                        !browse
SELF.ResetQueue( Reset:Done )           !перенаберем очередь просмотра
IF ~SELF.Sort.Locator &= NULL           !если имеется локатор
SELF.Sort.Locator.Reset                 !установим соответствие значения поиска с
                                        !реальной записью
END
END
END
```

См. также: FreeElement

UpdateWindow (перерисовывает элемент управления локатора с его текущим значением)

UpdateWindow, VIRTUAL

Метод **UpdateWindow** перерисовывает элемент управления локатора с его текущим значением.

Рассматриваемый метод является виртуальным, следовательно, другие методы базового класса могут непосредственно обращаться к этому методу порожденных классов. Это позволяет вам легко реализовать собственные варианты данного метода.

Реализация: Метод UpdateWindow перефильтрует лежащую в основе VIEW, присваивает свойству FreeElement текущее значение поиска (свойство Shadow), затем перерисовывает элемент управления локатора.

Пример:

```
MyBrowseClass.UpdateWindow PROCEDURE
                                        !обновить относящиеся к browse
                                        !элементы управления
CODE
IF ~(SELF.Sort.Locator &= NULL)         !если есть локатор
SELF.Sort.Locator.UpdateWindow          !перерисовать его
          END
```

См. также: FreeElement, Shadow

Класс Process

Обзор

Класс ProcessClass представляет собой класс ViewManager с добавленным окном индикации степени завершения процесса. Этот класс позволяет осуществить "пакетную" обработку виртуального файла (VIEW), используя необходимую упорядоченную последовательность обработки, ограничение диапазона значений ключевого поля и фильтрование записей, для того, чтобы обработать только конкретный набор записей в требуемой последовательности. Дополнительно к этому класс ProcessClass обеспечивает соответствующую визуальную "обратную связь" с конечным пользователем, демонстрируя степень завершенности процесса обработки.

Взаимосвязь с другими ABC-классами

ProcessClass является производным классом от класса ViewManager, кроме того, он привлекает многие другие ABC-классы для выполнения присущих им задач. Поэтому, если в вашей программе объявляется использование класса ProcessClass, то должно объявляться и использование этих других классов. Большинство этих объявлений делается автоматически при включении оператором INCLUDE заголовка класса ProcessClass (ABREPORT.INC) в раздел данных вашей программы. См. раздел *Концептуальный пример*.

В классе ReportManager класс ProcessClass используется для обеспечения последовательной обработки печатаемых данных и визуальной обратной связи с пользователем в процессе подготовки печатного документа.

Реализация ABC-шаблона

Шаблоны ABC автоматически включают все необходимые для поддержки процесса пакетной обработки классы (процедуры Process и Report), заданные в вашем приложении.

В шаблонах для каждого процесса пакетной обработки ((процедуры Process и Report) в приложении создается "производный" класс от класса ProcessClass. К производному классу обращение происходит следующим образом: *Имя процедуры*:Process. Объект этого производного класса поддерживает все функциональные возможности, реализуемые в шаблоне процедуры типа Process или Report.

Производный объект ProcessClass является локальным по отношению к процедуре, специфичен для одного процесса и опирается на глобальные объекты, реализующие работу с файлами RelationManager и FileManager.

Исходные файлы для ProcessClass

Исходные тексты класса ProcessClass по умолчанию устанавливаются в каталог \CLARION4\LIBSRC. Характерные для ProcessClass файлы и соответствующие им компоненты:

ABREPORT.INC	Объявление класса ProcessClass
ABREPORT.CLW	Определения методов ProcessClass

Концептуальный пример

В следующем примере демонстрируется обычная последовательность операторов объявления объекта класса ProcessClass, его активизации, инициализации, использования и терминирования самого объекта и связанных с ним объектов. В этом примере обрабатываются выбранные в файле записи, производится изменение данных и выводится окно с индексацией степени завершенности процесса.

```
PROGRAM
INCLUDE('ABWINDOW.INC')                          ! Объявим клвсс
                                                 !WindowManager
MAP
END
Customer FILE,DRIVER('TOPSPEED'),PRE(CUS),THREAD !объявим файл Customer
BYNUMBER KEY(CUS:CUSTNO),NOCASE,OPT,PRIMARY
Record RECORD,PRE()
CUSTNO LONG
Name STRING(30)
State STRING(2)
END
END
CusView VIEW(Customer)                           !объявим виртуальный файл (VIEW )
END
Access:Customer CLASS(FileManager)               !объявим объект Access:Customer
Init PROCEDURE
END
Relate:Customer CLASS(RelationManager)           !объявим объект Relate:Customer
Init PROCEDURE
END
```

```
    ThisWindow CLASS(ReportManager)       !объявим объект  ThisWindow
    Init PROCEDURE(),BYTE,PROC,VIRTUAL
    Kill PROCEDURE(),BYTE,PROC,VIRTUAL
    END
    ThisProcess CLASS(ProcessClass)   !объявим объект  ThisProcess
    TakeRecord PROCEDURE(),BYTE,PROC,VIRTUAL
    END
    ProgressMgr StepLongClass  !объявим объект  ProgressMgr
    GlobalErrors ErrorClass       !объявим объект  GlobalErrors
    VCRRequest LONG(0),THREAD
    Thermometer BYTE    !объявим переменную PROGRESS variable
    ProgressWindow
WINDOW('Progress...'),AT(,,142,59),CENTER,TIMER(1),GRAY,DOUBLE
    PROGRESS,USE(Thermometer),AT(15,15,111,12),RANGE(0,100)
    STRING(''),AT(0,3,141,10),USE(?UserString),CENTER
    STRING(''),AT(0,30,141,10),USE(?PctText),CENTER
    BUTTON('Cancel'),AT(45,42),USE(?Cancel)
    END
    CODE
    ThisWindow.Run()     !выполним процедуру Process
    ThisWindow.Init PROCEDURE()   !процедура инициализации
    ReturnValue BYTE,AUTO
    CODE
    GlobalErrors.Init       !инициализация объекта GlobalErrors
    Relate:Customer.Init   !инициализация  Relate:Customer
    ReturnValue = PARENT.Init()       !вызов инициализации базового класса
    IF ReturnValue THEN RETURN ReturnValue.
    SELF.FirstField = ?Thermometer      !установить FirstField для объекта
ThisWindow
    SELF.VCRRequest &= VCRRequest !VCRRequest не используется
    SELF.Errors &= GlobalErrors!установить обработчик ошибок для ThisWindow
    Relate:Customer.Open !Открыть файл Customer и связанные с ним
    OPEN(ProgressWindow)      !открыть окно
    SELF.Opened=True     !установить признак, что окно ThisWindow открыто
    ProgressMgr.Init(ScrollSort:AllowNumeric) !инициализация объекта
ProgressMgr
    !инициализируем объект ThisProcess, указав имена его VIEW,
RelationManager,ProgressMgr
    ! и переменных
ThisProcess.Init(CusView,Relate:Customer,?PctText,Thermometer,ProgressMgr,CUS:CUSTNO)
```

```
ThisProcess.AddSortOrder(CUS:BYNUMBER)      !установить последовательность
                                            !обработки
SELF.Init(ThisProcess)                      !инициализация характерная для
                                            !процесса
SELF.AddItem(?Cancel,RequestCancelled )     !зарегистрировать Cancel для окна
ThisWindow
SELF.SetAlerts()                            !алерт-клавиши для окна ThisWindow
RETURN ReturnValue
ThisWindow.Kill PROCEDURE()                 !действия по завершению работы.
ReturnValue BYTE,AUTO
CODE
ReturnValue = PARENT.Kill()                 !выполнение завершающих
                                            !действий для базового класса
IF ReturnValue THEN RETURN ReturnValue.
Relate:Customer.Close                       !закрыть файл Customer
Relate:Customer.Kill                        !уничтожить объект Relate:Customer
GlobalErrors.Kill                           !уничтожить объект GlobalErrors
RETURN ReturnValue
ThisProcess.TakeRecord PROCEDURE()          !действия для каждой
                                            !обрабатываемой записи
ReturnValue BYTE,AUTO
CODE
IF NOT CUS:State                            !если поле State пустое
CUS:State = 'FL'                            ! занести в него 'FL'
END
ReturnValue = PARENT.TakeRecord()           !для каждой записи вызвать объект
                                            !базового класса
PUT(CusView)                                !записать измененную запись
IF ERRORCODE()                              !если произошла ошибка,
ThisWindow.Response = RequestCompleted      ! прервать процесс
ReturnValue = Level:Fatal                   !для обеспечения целостности
                                            !файла
            Use IF Relate:Customer.Update()
END                                         !Customer и связанных с ним
                                            !использовать
          RETURN ReturnValue                !проверку IF
Relate:Customer.Update()
Access:Customer.Init PROCEDURE
CODE
PARENT.Init(Customer,GlobalErrors)
SELF.FileNameValue = 'Customer'
SELF.Buffer &= CUS:Record
SELF.LazyOpen = False
```

```
SELF.AddKey(CUS:BYNUMBER,'CUS:BYNUMBER',0)
Relate:Customer.Init PROCEDURE
CODE
Access:Customer.Init
PARENT.Init(Access:Customer,1)
```

Свойства класса ProcessClass

Класс ProcessClass наследует все свойства своего родительского класса ViewManager, от которого он произведен. Более подробную информацию смотри в разделе "*Свойства класса ViewManager*"

Дополнительно к унаследованным свойствам класс ProcessClass имеет следующие свойства:

Percentile (степень выполнения процесса)

Percentile &BYTE, PROTECTED

Свойство **Percentile** представляет собой ссылку на переменную, содержимое которой показывает, в какой степени выполнен процесс. Объект класса ProcessClass периодически обновляет значение этого свойства, поэтому его можно использовать в качестве USE-переменной для элемента управления типа PROGRESS.

В методе Init это свойство принимает начальное значение. См. раздел *Концептуальный пример*.

Эта переменная имеет атрибут PROTECTED, поэтому к ней могут обращаться только методы класса ProcessClass или производных от него классов.

Смотри также: Init

PText (номер элемента управления)

PText SIGNED

Свойство **PText** содержит номер текстового элемента управления, такого как STRING или PROMPT. Объект класса ProcessClass использует этот элемент управления для обратной связи с конечным пользователем. Свойство **PText** инициализируется в методе Init. См. раздел *Концептуальный пример*.

Эта переменная имеет атрибут PROTECTED, поэтому к ней могут обращаться только методы класса ProcessClass или производных от него классов.

Смотри также: Init

RecordsProcessed (число обработанных элементов)

RecordsProcessed LONG

Свойство **RecordsProcessed** содержит число обработанных к этому моменту элементов. Объект класса ProcessClass использует это свойство для вычисления степени завершенности процесса.

RecordsToProcess (число элементов, подлежащих обработке)

RecordsToProcess LONG

Свойство **RecordsToProcess** содержит общее количество элементов, подлежащих обработке. В объекте класса ProcessClass это свойство используется для определения степени завершенности процесса.

Методы класса ProcessClass

Объект класса ProcessClass наследует все методы класса ViewManager, от которого он произведен. Более подробную информацию смотрите в разделе *Свойства класса ViewManager*. Дополнительно к (или вместо) унаследованным методам объект класса ProcessClass имеет следующие методы:

Функциональная организация—предполагаемое использование

В целях облегчения понимания методов класса ProcessClass полезно разделить их на две категории в соответствии с их предполагаемым использованием: интерфейсные и виртуальные методы. Это разделение отражает тот факт, как мы предполагаем использовать методы класса ProcessClass.

Primary Interface Methods

Внутреннее (однократное) использование:

Init	инициализация объекта класса ProcessClass
AddRange	добавление к активной последовательности сортировки ограничения по диапазону значений
AddSortOrder	добавление последовательности сортировки
AppendOrder	очистка последовательноси сортировки
SetProgressLimits	калибровка индикатора степени завершенности процесса StepClass
Kill	уничтожение объекта класса ProcessClass

Основное использование:

Open	открыть виртуальный файл
Next	получить следующий элемент из набора подошедших по диапазону и фильтру
Previous	получить предыдущий элемент из набора подошедших по диапазону

	и фильтру
PrimeRecord	приготовить добавление записи
ValidateRecord	проверить текущий набор записей на заданные ограничения
SetFilter	указать фильтр для активной последовательности сортировки
SetSort	указать активную последовательность сортировки
ApplyFilter	получить результирующий набор записей после отфильтровывания
ApplyOrder	упорядочить результирующий набор записей
ApplyRange	наложить ограничения на диапазон значений в упорядочивающей последовательности результирующего набора
Close	закрыть виртуальный файл

Эпизодическое использование:

GetFreeElementName	получить имя поля свободного элемента
Reset	встать на начало результирующего набора записей
SetOrder	сменить активную последовательность упорядочения

Виртуальные методы

Обычно такие методы не вызываются напрямую—к ним обращаются методы первичного интерфейса (Primary Interface methods). Однако мы предвидим, что вы захотите переопределить эти методы и, поскольку они являются виртуальными, их очень легко переопределить. Если их не переопределить, эти методы обеспечивают разумные действия предпринимаемые по умолчанию.

Next	получить следующий элемент из результирующего набора.
Previous	получить предыдущий элемент из результирующего набора.
Reset	установиться на первый элемент из результирующего набора.
SetSort	установить активную последовательность сортировки
ValidateRecord	проверить текущую запись из результирующего набора
Kill	уничтожить объект класса ProcessClass

Init (инициализировать объект класса ProcessClass)

Init(view, relationmanager [, progress txt] [, progress pct] | [, total records] | **)**
|,stepclass, free element |

Init	инициализировать объект класса ProcessClass
view	метка структуры VIEW (имя виртуального файла)
relationmanager	имя объекта класса RelationManager для первичного файла структуры VIEW
progress txt	числовая константа, переменная, мнемоническое имя или выражение, содержащее номер текстового экранного элемента управления. Объект класса ProcessClass использует этот элемент управления для реализации обратной связи с конечным пользователем. Если этот

параметр опущен, то объект ProcessClass не обеспечивает текстовой обратной связи с пользователем.

progress pct — Имя переменной типа BYTE, содержимое которой соответствует степени завершенности процесса. Объект класса ProcessClass периодически изменяет значение этой переменной, поэтому она может служить USE-переменной экранного элемента управления PROGRESS. Если этот параметр опущен, то объект не обеспечивает количественной обратной связи.

total records — Числовая константа, переменная, мнемоническое имя соответствия или выражение, которое дает оценочное число записей, которое нужно обработать. Объект класса ProcessClass использует это число при вычислении степени завершенности процесса. В случаях, когда легко вычислить число записей, подлежащих обработке, вам следует использовать этот параметр, например, когда не используется динамическое фильтрование записей. Если этот параметр опущен, то его значение устанавливается в 0.

stepclass — Имя объекта класса StepClass, который служит для контроля степени выполнения процесса. Он используется для определения того, какая доля процесса уже выполнена. Вы должны использовать этот параметр, когда трудно вычислить число записей, подлежащих обработке, как в случае использования динамического фильтрования.

free element — Метка поля "свободных" элементов, относящихся к виртуальному файлу. Объект класса StepClass использует это поле при определении того, в какой степени выполнен процесс. Более подробную информацию см. в разделе Методы StepClass – GetPercentile.

Метод **Init** инициализирует объект класа ProcessClass. Если вы обеспечили для процесса параметр *total records*, то объект ProcessClass вычисляет степень завершенности процесса, как функцию от параметра *total records* и числа обработанных записей. В противном случае ProcessClass полагается на объект *stepclass* в вычислении степени завершенности процесса. Более подробную информацию смотрите в разделе Методы StepClass – GetPercentile.

Реализация: Метод **Init** присваивает значение *progress txt* свойству PText, указатель на *progress pct* присваивается свойству Percentile, а значение total records свойству RecordsToProcess. Метод Init обращается к методу Init класса ViewManager.

Пример:

```
                                    !инициализация объекта класса ProcessClass
Process.Init( Process:View, |        !остановить виртуальный файл VIEW
Relate:Client, |                     !установить менеджер связей
                                    !RelationManager для первичного файла.
```

```
?PctText, |                    !установить экранный элемент управления
                               !для текстовых сообщений
PctDone, |                     !установить use-переменную для элемента
                               !управления PROGRESS
ProgressMgr, |                 !установить объект StepClass для управления
                               !процессом
CLI:Name)
```

Смотри также: Percentile, PText, RecordsToProcess, ViewManager.Init

Kill (уничтожить объект класса ProcessClass)

Kill, VIRTUAL

Метод **Kill** уничтожает объект класса ProcessClass, освобождая всю память, выделенную в процессе его существования и выполняет другие действия по завершении.

Метод Kill является виртуальным, поэтому другие методы базового класса могут обращаться напрямую к виртуальному методу производного класса. Виртуальный метод позволяет легко реализовать свою собственную версию этого метода.

Реализация: Метод Kill обращается к методу ViewManager.Kill.

Пример:

```
                               !инициализация объекта класса ProcessClass
Process.Init( Process:View, |  !установить VIEW
Relate:Client, |               ! установить менеджер связей
                               !RelationManager для первичного файла.
?PctText, |                    ! установить экранный элемент управления
                               !для текстовых сообщений
PctDone, |                     ! установить use-переменную для элемента
                               !управления PROGRESS
ProgressMgr, |                 ! установить объект StepClass для управления
                               !процесса
CLI:Name)                      !программный код процедуры
Process.Kill                    !уничтожение объекта класса ProcessClass
```

Смотри также: ViewManager.Kill

Next (get next element)

Next([process records] **), VIRTUAL**

Next получить следующий элемент из результирующего набора записей.

process records Логическая константа, переменная, мнемоническая метка соответствия или выражение, с помощью которой объекту класса ProcessClass сообщается, нужно ли обновить индикатор степени выполнения процесса. Значение нуль (0) не изменяет индикатор степени выполнения процесса, любое другое значение вызывает изменение индикатора. Если этот параметр опущен, то по умолчанию подразумевается единица (1).

Метод **Next** считывает следующую запись из результирующего набора и возвращает код, означающий, успешно это произведено или с ошибкой.

Next является виртуальным методом, поэтому другие методы базового класса могут напрямую обращаться методу Next в порожденном классе. Это позволяет легко реализовать свою собственную пользовательскую версию этого метода.

Реализация: Метод Next обращается к методу ViewManager.Next. Метод ProcessClass.Next обновляет и свойство RecordsProcessed и свойство Percentile.

Тип возвращаемого значения: BYTE

Пример:

```
ACCEPT
CASE EVENT()
OF Event:OpenWindow
  Process.Reset                          !встать на первую запись
IF Process.Next()                        !прочитать первую запись
   POST(Event:CloseWindow)               !если записей нет убить объект
   CYCLE
END
OF Event:Timer                           !по таймерному событию обрабатывать
                                         !записи
  StartOfCycle=Process.RecordsProcessed
LOOP WHILE Process.RecordsProcessed-StartOfCycle<RecordsPerCycle
  CASE Process.Next()                    !взять слудующую запись
  OF Level:Notify                        !если конец файла
    MESSAGE('Process Completed')         ! оповестить пользователя
    POST(EVENT:CloseWindow)              ! и убить объект
```

```
  BREAK
 OF Level:Fatal                    !если фатальная работа
  POST(EVENT:CloseWindow)          ! убить объект
  BREAK
....
```

Смотри также: Percentile, RecordsProcessed, ViewManager.Next

Reset (встать на первый элемент)

Reset, VIRTUAL

Метод **Reset** позиционируется на первый элемент из результирующего набора записей и сбрасывает идикатор степени выполнения процесса.

Reset является виртуальным методом, поэтому другие методы базового класса могут напрямую обращаться к методу Reset в порожденном классе. Это позволяет легко реализовать свою собственную пользовательскую версию этого метода.

Реализация: Метод Reset устанавливает свойство RecordsProcessed в ноль (0), по условию обращается к методу SetProgressLimits, затем обращается к методу ViewManager.Reset.

Пример:

```
CASE EVENT()
OF Event:OpenWindow
Process.Reset                      !встать на первую запись
IF Process.Next()                  !получить первую запись
 POST(Event:CloseWindow)           !если записей нет - закончить
 CYCLE
END
```

Смотри также: SetProgressLimits, ViewManager.Reset

SetProgressLimits (калибровка индикатора степени выполнения процесса)

SetProgressLimits

Метод **SetProgressLimits** предоставляет нижнюю и верхнюю границы результирующего набора записей—учитывая активный порядок сортировки, ограничивающие диапазон условия и фильтры—для объекта StepClass, который управляет индикацией степени завершенности процесса.

Метод Init задает объект класса StepClass.

Реализация: Метод SetProgressLimits подразумевает, что задан объект класса StepClass. Метод Reset в зависимости от условий обращается к методу SetProgessLimits.

Пример:

```
MyProcessClass.Reset PROCEDURE    !приготовиться к обработке записей
CODE
SELF.RecordsProcessed = 0         !установить RecordsProcessed в 0
SELF.SetProgressLimits            !установить границы в StepClass
                                  ! на реальные значения
PARENT.Reset                      !обратиться к ViewManager.Reset
                                  !и встать на первую запись
```

Смотри также: Init, Reset, StepClass.SetLimits

TakeRecord (a virtual to process each report record)

TakeRecord, VIRTUAL, PROC

Метод **TakeRecord** является виртуальным вместилищем каждой записи из результирующего набора. Он возвращает значение, означающее, должна ли продолжаться обработка записей или нет. Метод TakeRecord возвращает Level:Benign для индикации того факта, что процесс должен продолжаться нормальным образом. Возвращаемое значение Level:Notify означает, что обработка завершена и должна прекратиться.

Метод TakeRecord является виртуальным методом, т.е. другие методы базового класса могут напрямую обращаться к нему в производном классе. Это позволяет легко реализовать свою собственную пользовательскую версию этого метода.

Этот метод имеет атрибут PROC, поэтому к нему можно обращаться как к процедуре и игнорировать возвращаемое им значение.

Реализация: Метод ReportManager.TakeWindowEvent обращается к методу TakeRecord для считывания каждой обрабатываемой записи. Для печатных отчетов этот метод TakeRecord обычно реализует любые присущие разделам DETAIL фильтры и печатает разделы DETAIL, не имеющие фильтров. При последовательных обработках метод TakeRecord обычно реализует всевозможные необходимые действия с записью.

Тип возвращаемого значения: BYTE

Пример:

```
ThisWindow.TakeRecord PROCEDURE()
CODE
IF ORD:Date = TODAY()
PRINT(RPT:detail)
END
RETURN Level:Benign
```

Смотри также: ReportManager.TakeWindowEvent

Обзор

Концепции StepClass

Класс StepClass является абстрактным классом—он не используется сам по себе. Однако от него производятся другие полезные классы и другие структуры (такие, как BrowseClass и ProcessClass) используют его для обращения к производным от него классам.

Производные от StepClass классы позволяют определить нижнюю и верхнюю границы диапазона и последовательность шагов внутри диапазона. Кроме того, эти классы помогают управлять шагами, определенными для бегунка на линейке скроллинга, индикатора степени выполнения процесса и любого элемента управления, который отображает относительное положение на оси в рамках ограниченного диапазона.

Производные от StepClass классы реализуют некоторое общее изменение границ (алфавитно-цифровых или числовых) и шаги (побуквенное приближение, плавное приближение), которые выполняются в контексте процесса чтения или пакетной обработки.

Класс StepClass требует, чтобы данные считывались по ключу. В противном случае можно исхитриться отслеживать процесс просто по счетчику записей и тогда класс StepClass не нужен.

Связь с другими классами Application Builder

Производные от StepClass классы используются классами BrowseClass и ProcessClass. Таким образом, если в программе активизируется объект класса BrowseClass или ProcessClass, также должен активизироваться объект класса StepClass. Большая часть действий по активизации выполняется автоматически при включении оператором INCLUDE заголовка класса BrowseClass или ProcessClass (файлы ABBROWSE.INC или ABPRINT.INC) в раздел данных программы. Смотри раздел *Концептуальный пример*)

Реализация ABC шаблона

ABC шаблоны автоматически включают все эти классы и генерируют код, необходимый для использования объекта класса StepClass в процессе просмотра, получения отчета или пакетной обработки.

Исходные файлы для класса StepClass

По умолчанию исходные файлы класса StepClass устанавливаются в каталог \CLARION4\LIBSRC. Характерные для класса StepClass файлы и их соответствующие компоненты:

ABBROWSE.INC объявления класса StepClass :

объявления класса StepLongClass
объявления класса StepRealClass

объявления класса StepStringClass
объявления класса StepCustomClass

ABBROWSE.CLW определения методов класса StepClass :

определения методов класса StepLongClass
определения методов класса StepRealClass
определения методов класса StepStringClass
определения методов класса StepCustomClass

Концептуальный пример

В нижеследующем примере демонстрируется типовая последовательность операторов объявления, активизации, инициализации, использования и уничтожения объекта класса BrowseClass и связанных с ним объектов. В этом примере инициализируется постранично загружаемое окно списка, затем обрабатывается ряд связанных с этим событий, включая поиск, листание и изменение данных. При соответствующей инициализации объекты класса BrowseClass и WindowManager выполняют большую часть работы (обработка событий принятая по умолчанию) сами.

```
PROGRAM
INCLUDE('ABWINDOW.INC')              !объявить класс WindowManager
INCLUDE('ABBROWSE.INC')              ! объявить классы BrowseClass и StepClasses
MAP
END
State FILE,DRIVER('TOPSPEED'),PRE(ST),THREAD
StateCodeKey KEY(ST:STATECODE),NOCASE,OPT
Record RECORD,PRE()
STATECODE STRING(2)
STATENAME STRING(20)
END
END
StView VIEW(State)                    !объявить виртуальный файл VIEW
END
StateQ QUEUE                          !объявить очередь для окна списка
ST:STATECODE LIKE(ST:STATECODE)
ST:STATENAME LIKE(ST:STATENAME)
ViewPosition STRING(512)
END
GlobalErrors ErrorClass
Access:State CLASS(FileManager)
Init PROCEDURE
END
Relate:State CLASS(RelationManager)
Init PROCEDURE
```

```
END
VCRRequest LONG(0),THREAD
StWindow WINDOW('Browse States'),AT(,,123,152),IMM,SYSTEM,GRAY
LIST,AT(8,5,108,124),USE(?StList),IMM,HVSCROLL,FROM(StateQ),|
FORMAT('27L(2)|M~CODE~@s2@80L(2)|M~STATENAME~@s20@')
BUTTON('&Insert'),AT(8,133),USE(?Insert)
BUTTON('&Change'),AT(43,133),USE(?Change),DEFAULT
BUTTON('&Delete'),AT(83,133),USE(?Delete)
END
ThisWindow CLASS(WindowManager)
Init PROCEDURE(),BYTE,PROC,VIRTUAL
Kill PROCEDURE(),BYTE,PROC,VIRTUAL
END
BrowseSt CLASS(BrowseClass)                    !объявить объект BrowseSt
Q &StateQ
END
StStep StepStringClass                         ! объявить объект StStep
CODE
ThisWindow.Run()                               !запустить процедуру
ThisWindow.Init PROCEDURE()                    !инициализация
ReturnValue BYTE,AUTO
CODE
ReturnValue = PARENT.Init()
IF ReturnValue THEN RETURN ReturnValue.
GlobalErrors.Init
Relate:State.Init
SELF.FirstField = ?StList
SELF.VCRRequest &= VCRRequest
SELF.Errors &= GlobalErrors
Relate:State.Open
BrowseSt.Init(?StList,StateQ.ViewPosition,StView,StateQ,Relate:State,SELF)
OPEN(StWindow)
SELF.Opened=True
BrowseSt.Q &= StateQ
StStep.Init(+ScrollSort:AllowAlpha,ScrollBy:Runtime)    !инициализировать объект
StStep
BrowseSt.AddSortOrder(StStep,ST:StateCodeKey)          ! и подкльчить его к BrowseSt
BrowseSt.AddField(ST:STATECODE,BrowseSt.Q.ST:STATECODE)
BrowseSt.AddField(ST:STATENAME,BrowseSt.Q.ST:STATENAME)
SELF.SetAlerts()
RETURN ReturnValue
ThisWindow.Kill PROCEDURE()                    !уничтожение
ReturnValue BYTE,AUTO
```

```
CODE
ReturnValue = PARENT.Kill()
IF ReturnValue THEN RETURN ReturnValue.
Relate:State.Close
Relate:State.Kill
GlobalErrors.Kill
RETURN ReturnValue
Access:State.Init PROCEDURE
CODE
PARENT.Init(State,GlobalErrors)
SELF.FileNameValue = 'State'
SELF.Buffer &= ST:Record
SELF.AddKey(ST:StateCodeKey,'ST:StateCodeKey',0)
Relate:State.Init PROCEDURE
CODE
Access:State.Init
PARENT.Init(Access:State,1)
```

Свойства класса StepClass

Класс StepClass имеет единственное свойство— Controls. Это свойство наследуется производными он него классами. Это свойство описывается ниже.

Controls (последовательность сортировки StepClass)

Controls BYTE

Свойство **Controls** содержит значение, которое идентифицирует для объекта класса StepClass следующее:

- символы, включаемые в последовательность сортировки
- направление упорядочения (возрастающая или убывающая)

Значение этого свойства устанавливается методом Init.

Объект класса StepClass может быть связан с последовательностью сортировки в объекте BrowseClass. Для объекта класса BrowseClass последовательность сортировки устанавливается методом BrowseClass.AddSortOrder.

Реализация: Свойство Controls представляет собой один байт, в котором каждый бит представляет различную важную информацию для объекта класса StepClass. Установка значения этого свойства производится сложением значений мнемонических имен соответствия объявленных операторами EQUATE в файле ABBROWSE.INC.

Смотри также: Init, BrowseClass.AddSortOrder

Методы StepClass

GetPercentile (получить величину процентиля)

GetPercentile(*значение* **), VIRTUAL**

GetPercentile	Возвращает для заданного значения процентиль— выраженное в процентах его положение относительно границ диапазона объекта класса StepClass
значение	константа, переменная, мнемоническая метка соответствия или выражение, которое задает величину, для которой вычисляется процентиль.

Метод **GetPercentile** возвращает для заданного значения процентиль относительно границ диапазона установленного для объекта класса StepClass.

Это виртуальный метод, поэтому другие методы базового класса могут напрямую обращаться методу GetPercentile в производном классе. Это позволяет легко реализовать свою собственную версию этого метода.

Реализация: Метод GetPercentile представляет собой метод-заполнитель для классов, производных от класса StepClass— StepLongClass, StepRealClass, StepStringClass, StepCustomClass и др.

Тип возвращаемого значения: BYTE

Смотри также: StepLongClass.GetPercentile, StepRealClass.GetPercentile, StepStringClass.GetPercentile, StepCustomClass.GetPercentile

GetValue (получить по процентилю саму величину)

GetValue(процентиль **), VIRTUAL**

GetValue	Возвращает для заданного процентиля относительно границ диапазона, установленного для объекта класса StepClass, величину соответствующую такому процентилю.
процентиль	Целочисленная константа, переменная, метка соответствия или выражение, которое задает процентиль, для которого вычисляется величина.

Метод **GetValue** возвращает для заданного процентиля относительно границ диапазона, установленного для объекта класса StepClass, величину соответствующую такому процентилю.

Это виртуальный метод, поэтому другие методы базового класса могут напрямую обращаться методу GetPercentile в производном классе. Это позволяет легко реализовать

свою собственную версию этого метода.

Реализация: Метод GetPercentile представляет собой метод-заполнитель для классов, производных от класса StepClass— StepLongClass, StepRealClass, StepStringClass, StepCustomClass и др.

Тип возвращаемого значения: STRING

Смотри также: StepLongClass.GetPercentile, StepRealClass.GetPercentile, StepStringClass.GetPercentile, StepCustomClass.GetPercentile

Init (инициализация объекта класса StepClass)

Init(*элем. управления* **)**

Init Инициализирует объект класса StepClass.

элем. управления Целочисленные константы, переменные, мнемоническая метки соответствия или выражения, которые идентифицируют объект класса StepClass.

- символы включаемые в последовательность сортировки
- направление упорядочения (возрастающая или убывающая)

Метод **Init** инициализирует объект класса StepClass.

Реализация: метод Init устанавливает значение свойства Controls. Устанавливайте значение этого свойства, складывая значения соответствующих меток соответствия, объявленных в файле ABBROWSE.INC, как показано в следующем примере:

```
ITEMIZE,PRE(ScrollSort)
AllowAlpha EQUATE(1)                !включать в сортировке символы
ABCDEFGHIJKLMNOPQRSTUVWXYZ
AllowAlt EQUATE(2)                  ! включать в сортировке символы
‘!”J$%%^&*()”-=_+][#;~@:/.,?\|
AllowNumeric EQUATE(4)              ! включать в сортировке символы 0123456789
CaseSensitive EQUATE(8)             ! включать в сортировке символы
abcdefghijklmnopqrstuvwxyz
Descending EQUATE(16)               !сортировка в убывающей посл-ти
END
```

Пример:

```
MyStepClass.Init(ScrollSort:AllowAlpha+ScrollSort:AllowNumeric)
```

Смотри также: Controls

Kill (уничтожить объект класса StepClass)

Kill, **VIRTUAL**

Метод **Kill** является виртуальным методом, предназначенным для уничтожения объекта класса StepClass. Поскольку это виртуальный метод, методы других базовых классов могут напрямую обращаться к методу Kill в производных классах. Это позволяет легко реализовать свою собственную версию этого метода.

Реализация: Метод Kill представляет собой метод-заполнитель для классов, производных от класса StepClass—StepStringClass, StepCustomClass и т.д.

Смотри также: StepStringClass.Kill, StepCustomClass.Kill

SetLimit (установить границы диапазона для объекта StepClass)

SetLimit(*нижняя, верхняя* **), VIRTUAL**

SetLimit	Устанавливает границы диапазона для объекта класса StepClass
нижняя	Константа, переменная, мнемоническая метка соответствия или выражение, задающее для объекта класса StepClass нижнюю границу диапазона. Значение может быть числовым или алфавитно-цифровым.
верхняя	Константа, переменная, мнемоническая метка соответствия или выражение, задающее для объекта класса StepClass верхнюю границу диапазона. Значение может быть числовым или алфавитно-цифровым.

Метод **SetLimit** устанавливает нижнюю и верхнюю границы диапазона для объекта класса StepClass

Это виртуальный метод, поэтому другие методы базового класса могут напрямую обращаться методу SetLimit в производном классе. Это позволяет легко реализовать свою собственную версию этого метода.

Реализация: Метод SetLimit является методом-заполнителем для классов, производных от StepClass—StepLongClass, StepRealClass, StepStringClass и др.

Смотри также: StepLongClass.SetLimit, StepRealClass.SetLimit, StepStringClass.SetLimit

SetLimitNeeded (возвратить признак статических/динамических границ диапазона)

SetLimitNeeded, VIRTUAL

Метод **SetLimitNeeded** возвращает значение, означающее, являются ли границы диапазона объекта класса StepClass статическими (устанавливаемыми во время компиляции программы) или же они динамические (устанавливаемые во время выполнения). Возвращаемое значение единица (1) означает динамические границы, которые может потребоваться переустановить, если рассматриваемый результирующий набор записей изменяется (записи добавляются, удаляются или фильтруются). Возвращаемое значение ноль (0) означает, что границы диапазона фиксированы во время компиляции и не подстраиваются при изменении результирующего набора записей.

Метод SetLimitNeeded—виртуальный метод, поэтому другие методы базового класса могут напрямую обращаться методу SetLimitNeeded в производном классе. Это позволяет легко реализовать свою собственную версию этого метода.

Реализация: Метод SetLimitNeeded является методом-заполнителем для производных от класса StepClass классов, таких как StepStringClass.

Тип возвращаемого значения: BYTE

Смотри также: StepStringClass.SetLimitNeeded, BrowseClass.ResetThumbLimits

Свойства класса StepLongClass

Класс StepLongClass наследует все свойства класса StepClass, от которого он произведен.

Более подробную информацию смотри в разделах *Свойства класса StepClass* и *Концепции класса StepClass*. Дополнительно к унаследованным свойствам класс StepLongClass имеет следующие свойства:

Low (нижняя граница)

Low LONG

Свойство **Low** содержит значение нижней границы объекта класса StepLongClass.

Значение этого свойства устанавливает метод SetLimit.

Смотри также: SetLimit

High (верхняя граница)

High LONG

Свойство **High** содержит значение верхней границы объекта класса StepLongClass.

Значение этого свойства устанавливает метод SetLimit.

Смотри также: SetLimit

Методы класса StepLongClass

Класс StepLongClass наследует все методы класса StepClass, от которого он произведен.

Более подробную информацию смотри в разделах *Методы класса StepClass* и *Концепции класса StepClass.*

Дополнительно к унаследованным методам класс StepLongClass имеет следующие методы:

GetPercentile (получить величину процентиля)

GetPercentile(*значение*), VIRTUAL

GetPercentile	Возвращает для заданного значения процентиль— выраженное в процентах его положение относительно границ диапазона объекта класса StepLongClass
значение	константа, переменная, мнемоническая метка соответствия или выражение, которое задает величину, для которой вычисляется процентиль.

Метод **GetPercentile** возвращает для заданного значения процентиль относительно границ диапазона, установленного для объекта класса StepLongClass. Например, если установлены границы 0 и 1000, то выражение GetPercentile(750) даст в результате 75.

Это виртуальный метод, поэтому другие методы базового класса могут напрямую обращаться методу GetPercentile в производном классе. Это позволяет легко реализовать свою собственную версию этого метода.

Реализация: Метод SetLimit устанавливает нижнюю и верхнюю границы объекта класса StepLongClass

Тип возвращаемого значения: BYTE

Пример:

```
IF FIELD() = ?Locator                    !фокус на локаторное поле
IF EVENT() = EVENT:Accepted              !если событие - accepted
MyBrowse.TakeAcceptedLocator             ! его обрабатывает класс BrowseClass
?MyList{PROP:VScrollPos}=MyStep.GetPercentile(Locator)      !установить “бегунок”
                                                            !в соответствующую позицию
END
```

END

Смотри также: SetLimit

GetValue (получить по процентилю саму величину)

GetValue(*процентиль* **), VIRTUAL**

GetValue	Возвращает для заданного процентиля относительно границ диапазона, установленного для объекта класса StepLongClass, величину, соответствующую такому процентилю.
процентиль	Целочисленная константа, переменная, метка соответствия или выражение, которое задает процентиль, для которого вычисляется величина.

Метод **GetValue** возвращает для заданного процентиля относительно границ диапазона, установленного для объекта класса StepLongClass, величину, соответствующую такому процентилю. Например, если установлены границы 0 и 1000, то выражение GetValue(25) даст в результате '250'.

Это виртуальный метод, поэтому другие методы базового класса могут напрямую обращаться методу GetPercentile в производном классе. Это позволяет легко реализовать свою собственную версию этого метода.

Реализация: Метод SetLimit устанавливает нижнюю и верхнюю границы объекта класса StepLongClass

Тип возвращаемого значения: STRING

Пример:

```
IF FIELD() = ?MyList                                    !фокус на список
IF EVENT() = EVENT:ScrollDrag                           !если бегунок переместился
Locator=MyStep.GetValue(?MyList{PROP:VScrollPos})       !обновить локатор
END
END
```

Смотри также: SetLimit

SetLimit (установить границы диапазона для объекта StepClass)

SetLimit(*нижняя, верхняя* **), VIRTUAL**

SetLimit	Устанавливает границы диапазона для объекта класса StepLongClass
нижняя	Числовая константа, переменная, мнемоническая метка соответствия или выражение, задающее нижнюю границу диапазона. Значение может быть числовым или алфавитно-цифровым.
верхняя	Числовая константа, переменная, мнемоническая метка соответствия или выражение, задающее верхнюю границу диапазона. Значение может быть числовым или алфавитно-цифровым.

Метод **SetLimit** устанавливает границы диапазона для объекта класса StepLongClass

Это виртуальный метод, поэтому другие методы базового класса могут напрямую обращаться методу SetLimit в производном классе. Это позволяет легко реализовать свою собственную версию этого метода.

Пример:

```
MyStep.SetLimit(0,100) !установить границы 0-100%
```

Свойства класса StepRealClass

Класс StepRealClass наследует все свойства класса StepClass, от которого он произведен. Более подробную информацию смотри в разделах *Свойства класса StepClass* и *Концепции класса StepClass*. Дополнительно к унаследованным свойствам класс StepRealClass имеет следующие свойства:

Low (нижняя граница)

Low REAL

Свойство **Low** содержит значение нижней границы объекта класса StepRealClass.

Значение этого свойства устанавливает метод SetLimit.

Смотри также: SetLimit

High (верхняя граница)

High LONG

Свойство **High** содержит значение верхней границы объекта класса StepRealClass.

Значение этого свойства устанавливает метод SetLimit.

Смотри также: SetLimit

Методы класса StepRealClass

Класс StepRealClass наследует все методы класса StepClass, от которого он произведен.
Более подробную информацию смотри в разделах *Методы класса StepClass* и *Концепции класса StepClass*.

Дополнительно к унаследованным методам класс StepRealClass имеет следующие методы:

GetPercentile (получить величину процентиля)

GetPercentile(*значение*), VIRTUAL

GetPercentile Возвращает для заданного значения процентиль— выраженное в процентах его положение относительно границ диапазона объекта класса StepRealClass

значение константа, переменная, мнемоническая метка соответствия или выражение, которое задает величину для которой вычисляется процентиль.

Метод **GetPercentile** возвращает для заданного значения процентиль относительно границ диапазона, установленного для объекта класса StepRealClass. Например, если установлены границы 0 и 1000, то выражение GetPercentile(750) даст в результате 75.

Это виртуальный метод, поэтому другие методы базового класса могут напрямую обращаться методу GetPercentile в производном классе. Это позволяет легко реализовать свою собственную версию этого метода.

Реализация: Метод SetLimit устанавливает нижнюю и верхнюю границы объекта класса StepLongClass

Return Data Type: BYTE

Пример:

```
IF FIELD() = ?Locator                                    !фокус на локаторное поле
IF EVENT() = EVENT:Accepted                              !если событие - accepted
MyBrowse.TakeAcceptedLocator                              !его обработает BrowseClass
?MyList{PROP:VScrollPos}=MyStep.GetPercentile(Locator) !установить бегунок в
                                                         !нужное положение
END
END
```

Смотри также: SetLimit

GetValue (получить по процентилю саму величину)

GetValue(*процентиль*), VIRTUAL

GetValue Возвращает для заданного процентиля относительно границ диапазона, установленного для объекта класса StepRealClass, величину соответствующую такому процентилю.

процентиль Целочисленная константа, переменная, метка соответствия или выражение, которое задает процентиль, для которого вычисляется величина.

Метод **GetValue** возвращает для заданного процентиля относительно границ диапазона, установленного для объекта класса StepRealClass, величину соответствующую такому процентилю. Например, если установлены границы 0 и 1000, то выражение GetValue(25) даст в результате '250'.

Это виртуальный метод, поэтому другие методы базового класса могут напрямую обращаться методу GetPercentile в производном классе. Это позволяет легко реализовать свою собственную версию этого метода.

Реализация: Метод SetLimit устанавливает нижнюю и верхнюю границы объекта класса StepRealClass

Тип возвращаемого значения: STRING

Пример:

```
IF FIELD() = ?MyList                                    !фокус на список
IF EVENT() = EVENT:ScrollDrag                           !если бегунок переместился
Locator=MyStep.GetValue(?MyList{PROP:VScrollPos})       !изменить локатор
END
END
```

Смотри также: SetLimit

SetLimit (установить границы диапазона для объекта StepClass)

SetLimit(*нижняя, верхняя***), VIRTUAL**

SetLimit	Устанавливает границы диапазона для объекта класса StepRealClass
нижняя	Числовая константа, переменная, мнемоническая метка соответствия или выражение, задающее нижнюю границу диапазона. Значение может быть числовым или алфавитно-цифровым.
верхняя	Числовая константа, переменная, мнемоническая метка соответствия или выражение, задающее верхнюю границу диапазона. Значение может быть числовым или алфавитно-цифровым.

Метод **SetLimit** устанавливает границы диапазона для объекта класса StepRealClass.

Это виртуальный метод, поэтому другие методы базового класса могут напрямую обращаться методу SetLimit в производном классе. Это позволяет легко реализовать свою собственную версию этого метода.

Пример:

```
MyStep.SetLimit(0,100) !установить границы 0-100%
```

Свойства класса StepStringClass

Класс StepStringClass наследует все свойства класса StepClass, от которого он произведен. Более подробную информацию смотри в разделах *Свойства класса StepClass* и *Концепции класса StepClass*. Дополнительно к унаследованным свойствам класс StepStringClass имеет следующие свойства:

LookupMode (ожидаемое распределение данных)

LookupMode BYTE

Свойство **LookupMode** *ожидаемое* распределение данных, реализуемое объектом класса StepStringClass. Ожидаемое плюс *реальное* распределение данных в конечном счете определяет, как "далеко" индикатор (бегунок линейки скроллинга или индикатор степени выполнения процесса) в действительности перемещается в процессе обработки записей.

Значение этого свойства устанавливает метод Init.

Реализация: Действительное распределение данных для американских фамилий, английского алфавита и распределение данных во время выполнения рассчитывается из реальных данных. Класс StepCustomClass поддерживает и другие распределения данных. Соответствующие им мнемонические метки объявлены в файле ABBROWSE.INC:

```
ITEMIZE,PRE(ScrollBy)
Name EQUATE                 !распределение американских фамилий
Alpha EQUATE                !распределение слов по буквам английского
                            !алфавита
Runtime EQUATE              !реальное распределение имеющее место во
                            !время выполнения программы
END
```

Распределение американских фамилий и распределение слов по буквам английского алфавита определены в ABBROWSE.CLW как:

```
Scroll:Alpha STRING(' AFANATB BFBNBTC CFCNCT'|
&'D DFDNDTE EFENETF FFFNFT'|
&'G GFGNGTH HFHNHTI IFINIT'|
&'J JFJNJTK KFKNKTL LFLNLT'|
&'M MFMNMTN NFNNNTO OFONOT'|
&'P PFPNPTQ QNR RFRNRTS SF'|
&'SNSTT TFTNTTU UFUNUTV VF'|
&'VNVTW WFWNWTX XFXNXTY YF'|
&'YNYTZ ZN')
Scroll:Name STRING(' ALBAMEARNBAKBATBENBIABOBBRA'|
&'BROBUACACCARCENCHRCOECONCORCRU'|
&'DASDELDIADONDURELDEVEFELFISFLO'|
&'FREFUTGARGIBGOLGOSGREGUTHAMHEM'|
```

```
&'HOBHOTINGJASJONKAGKEAKIRKORKYO' |
&'LATLEOLIGLOUMACMAQMARMAUMCKMER' |
&'MILMONMORNATNOLOKEPAGPAUPETPIN' |
&'PORPULRAUREYROBROSRUBSALSCASCH' |
&'SCRSHASIGSKISNASOUSTESTISUNTAY' |
&'TIRTUCVANWACWASWEIWIEWIMWOLYOR')
```

Смотри также: Init

Root (статическая часть шага)

Root &CSTRING, PROTECTED

Свойство **Root** представляет собой ссылку на структуру, содержащую статические или неизменные символы в шаге. Например, если установлены границы диапазона 'abbey' и 'abracadabra', то свойство Root содержит 'ab'. Связанное с этим свойством свойство TestLen равно длине неизменной части, то есть в данном примере 2.

Это свойство имеет атрибут PROTECTED, поэтому к нему может обращаться только метод класса StepStringClass или соответствующий ему метод в производном от StepStringClass классе.

Реализация: свойства Root и TestLen используются методами GetPercentile и GetValue для эффективного просмотра определенных шагов.

Смотри также: GetPercentile, GetValue, TestLen

SortChars (допустимые для сортировки символы)

SortChars &CSTRING

Свойство **SortChars** указывает на структуру, содержащую допустимые в сортировке символы или алфавит для объекта класса StepStringClass. Объект этого класса использует свойство SortChars для вычисления своих шагов. Например, если свойство SortChars содержит только символы 'ABYZ', то эта информация используется классом StepStringClass для расчета своих перемещений.

Значение свойства SortChars устанавливается методом Init.

Реализация: свойство SortChars влияет только на объекты класса StepStringClass с заданным в LookupMode распределением данных, определяемым во время выполнения программы.

Смотри также: Init, LookupMode

TestLen (длина статической части шага)

TestLen BYTE, PROTECTED

Свойство **TestLen** содержит длину значения свойства Root. Например, если установлены границы 'abbey' и 'abracadabra', Root содержит 'ab'. Соответственно свойство TestLen, равное длине значения свойства Root, содержит 2.

Это свойство имеет атрибут PROTECTED, поэтому к нему может обращаться только метод класса StepStringClass или соответствующий ему метод в производном от StepStringClass классе.

Метод StepStringClass.Init устанавливает значение свойства TestLen.

Реализация: свойства Root и TestLen используются методами GetPercentile и GetValue для эффективного просмотра определенных шагов.

Значение свойства TestLen зависит от значения свойства LookupMode. Распределение американских фамилий использует TestLen, равное 3, в распределении по английскому алфавиту используется TestLen, равное 2, а в распределении, определяемом во время выполнения, используется TestLen 4.

Смотри также: Init, LookupMode, Root

Методы класса StepStringClass

Класс StepStringClass наследует все методы класс StepClass, от которого он произведен. Более подробную информацию смотри в разделах *Методы класса StepClass* и *Концепции класса StepClass*.

Дополнительно к унаследованным методам класс StepRealClass имеет следующие методы:

GetPercentile (получить величину процентиля)

GetPercentile(*значение*), VIRTUAL

GetPercentile	Возвращает для заданного значения процентиль— выраженное в процентах его положение относительно границ диапазона объекта класса StepStringClass
значение	строковая константа, переменная, мнемоническая метка соответствия или выражение, которое задает величину, для которой вычисляется процентиль.

Метод **GetPercentile** возвращает для заданного значения процентиль относительно

границ диапазона, установленного для объекта класса StepStringClass. Например, если установлены границы 'A' и 'Z', то выражение GetPercentile('M') даст в результате 50.

Это виртуальный метод, поэтому другие методы базового класса могут напрямую обращаться методу GetPercentile в производном классе. Это позволяет легко реализовать свою собственную версию этого метода.

Реализация: Метод SetLimit устанавливает нижнюю и верхнюю границы объекта класса StepStringClass

Return Data Type: BYTE

Пример:

```
IF FIELD() = ?Locator                                    !фокус на локаторное поле
IF EVENT() = EVENT:Accepted                              !если событие - accepted
MyBrowse.TakeAcceptedLocator                             !его обработает BrowseClass
?MyList{PROP:VScrollPos}=MyStep.GetPercentile(Locator) !установить бегунок
END
END
```

Смотри также: SetLimit

GetValue (получить по процентилю саму величину)

GetValue(*процентиль* **), VIRTUAL**

GetValue	Возвращает для заданного процентиля относительно границ диапазона, установленного для объекта класса StepStringClass, величину соответствующую такому процентилю.
процентиль	Целочисленная константа, переменная, метка соответствия или выражение, которое задает процентиль, для которого вычисляется величина.

Метод **GetValue** возвращает для заданного процентиля относительно границ диапазона, установленного для объекта класса StepStringClass, величину, соответствующую такому процентилю. Например, если установлены границы 'A' и 'Z', то выражение GetValue(50) даст в результате 'M'.

Это виртуальный метод, поэтому другие методы базового класса могут напрямую обращаться методу GetPercentile в производном классе. Это позволяет легко реализовать свою собственную версию этого метода.

Реализация: Метод SetLimit устанавливает нижнюю и верхнюю границы объекта класса StepStringClass

Тип возвращаемого значения: STRING

Пример:

```
IF FIELD() = ?MyList                                    !фокус на список
IF EVENT() = EVENT:ScrollDrag                           !если бегунок переместился
Locator=MyStep.GetValue(?MyList{PROP:VScrollPos})       !изменить значение локатора
END
END
```

Смотри также: SetLimit

Init (инициализировать объект класса StepStringClass)

Init(*элем.управления, режим* **)**

Init инициализирует объект класса StepStringClass.

элем.управления целочисленные константы, переменные, мнемонические метки соответствия или выражения, идентифицирующие для объекта класса StepStringClass:

- символы, включаемые в последовательность сортировки
- направление упорядочения (возрастающая или убывающая)

режим целочисленная константа, переменная, мнемоническая метка соответствия или выражение, которое определяет точки распределения данных (или шаги), которые реализует объект класса StepStringClass. Это свойство плюс реальное распределение данных в конечном счете определяет, как "далеко" индикатор (бегунок линейки скроллинга или индикатор степени выполнения процесса) в действительности перемещается в процессе обработки записей. Допустимые режимы распределения данных: по американским фамилиям, по английскому алфавиту и распределение, вычисляемое во время выполнения из реальных данных.

Объект класса StepStringClass инициализируется методом **Init**.

Реализация: Метод Init устанавливает значение свойств Controls и LookupMode. Устанавливайте значение свойства Controls, складывая соответствующие мнемонические метки, объявленные в файле ABBROWSE.INC следующим образом:

```
ITEMIZE,PRE(ScrollSort)
AllowAlpha EQUATE(1)                !включить символы
                                    !ABCDEFGHIJKLMNOPQRSTUVWXYZ
AllowAlt EQUATE(2)                  ! включить символы '!"J$%%^&*()"-
                                    !=_+][#;~@:/.,?\|
AllowNumeric EQUATE(4)              ! включить символы 0123456789
CaseSensitive EQUATE(8)             ! включить символы
                                    !abcdefghijklmnopqrstuvwxyz
Descending EQUATE(16)               !сортировка по убыванию
END
```

Пример:

```
MyStepStringClass.Init(ScrollSort:AllowAlpha+ScrollSort:AllowNumeric)
                                                    !program code
MyStepStringClass.Kill
```

Смотри также: StepClass.Controls, LookupMode

Kill (виртуальный метод для уничтожения объекта класса StepStringClass)

Kill, VIRTUAL

Метод **Kill** освобождает любую выделенную ему в течение существования объекта память и выполняет другие завершающие работу действия.

Метод Kill имеет атрибут VIRTUAL, поэтому другие методы базового класса могут напрямую обращаться методу GetPercentile в производном классе. Это позволяет легко реализовать свою собственную версию этого метода.

Реализация: Метод Kill освобождает память, выделенную для свойств Ref, Root и SortChars.

Пример:

```
MyStepStringClass.Init(ScrollSort:AllowAlpha+ScrollSort:AllowNumeric)
                                                    !программный код
MyStepStringClass.Kill
```

SetLimit (установить границы для объекты класса StepClass)

SetLimit(*нижняя, верхняя* **), VIRTUAL**

SetLimit	Устанавливает верхнюю и нижнюю границы диапазона для объекта класса StepClass
нижняя	Строковая константа, переменная, мнемоническая метка соответствия или выражение, задающее для объекта класса StepClass нижнюю границу диапазона. Значение может быть числовым или алфавитно-цифровым.
верхняя	Строковая константа, переменная, мнемоническая метка соответствия или выражение, задающее для объекта класса StepClass верхнюю границу диапазона. Значение может быть числовым или алфавитно-цифровым.

Метод **SetLimit** устанавливает нижнюю и верхнюю границы диапазона для объекта класса StepClass

Это виртуальный метод, поэтому другие методы базового класса могут напрямую

обращаться к методу SetLimit в производном классе. Это позволяет легко реализовать свою собственную версию этого метода.

Пример:

```
MyStep.SetLimit('A','Z')          !установить алфавитные (большими буквами)
                                  !границы для линейки скроллинга
```

SetLimitNeeded (возвратить признак статических/динамических границ диапазона)

SetLimitNeeded, VIRTUAL

Метод **SetLimitNeeded** возвращает значение, означающее, являются ли границы диапазона объекта класса StepClass статическими (устанавливаемыми во время компиляции программы) или же они динамические (устанавливаемые во время выполнения). Возвращаемое значение единица (1) означает динамические границы, которые может потребоваться переустановить, если рассматриваемый результирующий набор записей изменяется (записи добавляются, удаляются или фильтруются). Возвращаемое значение ноль (0) означает, что границы диапазона фиксированы во время компиляции и не подстраиваются при изменении результирующего набора записей.

Метод SetLimitNeeded—виртуальный метод, поэтому другие методы базового класса могут напрямую обращаться к методу SetLimitNeeded в производном классе. Это позволяет легко реализовать свою собственную версию этого метода.

Реализация: Метод SetLimitNeeded возвращает единицу (1 или Истина), если свойство LookupMode равно ScrollBy:RunTime; в противном случае он возвращает ноль (0 или Ложь).

Тип возвращаемого значения: BYTE

Пример:

```
BrowseClass.ResetThumbLimits PROCEDURE
HighValue ANY
CODE
IF SELF.Sort.Thumb &= NULL OR ~SELF.Sort.Thumb.SetLimitNeeded()
RETURN
END
SELF.Reset
IF SELF.Previous()
RETURN
END
HighValue = SELF.Sort.FreeElement
SELF.Reset
IF SELF.Next()
RETURN
```

```
END
SELF.Sort.Thumb.SetLimit(SELF.Sort.FreeElement,HighValue)
```

Смотри также: StepClass.SetLimitNeeded, BrowseClass.ResetThumbLimits

Свойства класса StepCustomClass

Класс StepCustomClass наследует все свойства класса StepClass, от которого он произведен. Более подробную информацию смотри в разделах *Свойства класса StepClass* и *Концепции класса StepClass*. Дополнительно к унаследованным свойствам класс StepCustomClass имеет следующие свойства:

Entries (ожидаемое распределение данных)

Entries &CStringList, PROTECTED

Свойство **Entries** представляет собой ссылку на структуру, содержащую отметки границ, которые определяют ожидаемое распределение данных для объекта StepCustomClass. Это свойство определяет точки (или шаги) ожидаемого распределения данных, а также реализуемые объектом StepCustomClass верхнюю и нижнюю границы диапазона. Это свойство плюс реальное распределение данных определяют в конечном счете определяет как "далеко" индикатор (бегунок линейки скроллинга или индикатор степени выполнения процесса) в действительности перемещается в процессе обработки записей.

Это свойство имеет атрибут PROTECTED, поэтому к нему может обращаться только метод класса StepCustomClass или соответствующий ему метод в производном от StepCustomClass классе.

Метод AddMarker устанавливает значение свойства Entries.

Реализация: Свойство Entries ссылается на очередь QUEUE, объявленную в файле BROWSE.INC следующим образом:

```
CStringList QUEUE,TYPE
Item &CSTRING
END
```

Смотри также: AddMarker

Методы StepCustomClass

Класс StepCustomClass наследует все методы класса StepClass, от которого он произведен. Более подробную информацию смотри в разделах *Методы класса StepClass* и *Концепции класса StepClass*.

Дополнительно к унаследованным методам класс StepRealClass имеет следующие методы:

AddMarker (добавить отметку шага)

AddMarker(*отметка шага* **)**

AddMarker Добавляет отметку шага в ожидаемое распределение данных для объекта класса StepCustomClass.

отметка шага Строковая константа, переменная, мнемоническая метка соответствия или выражение, задающее границу следующего шага для каждого шага в ожидаемом распределении данных для объекта класса StepCustomClass.

Метод **AddMarker** добавляет отметку шага в ожидаемое распределение данных в объекте класса StepCustomClass.

Реализация: Метод AddMarker устанавливает значение свойства Entries.

Пример:

```
GradeStepClass.AddMarker('0')        !пропустить: 0-65
GradeStepClass.AddMarker('65')       !ниже среднего: 65-75
GradeStepClass.AddMarker('75')       !среднее: 75-85
GradeStepClass.AddMarker('85')       !больше среднего: 85-95
GradeStepClass.AddMarker('95')       !за пределами: 95-
GradeStepClass.AddMarker('1000')     !верхняя граница для остального
```

Смотри также: Markers

GetPercentile (получить величину процентиля)

GetPercentile(*значение* **), VIRTUAL**

GetPercentile Возвращает для заданного значения процентиль— выраженное в процентах его положение относительно границ диапазона объекта класса StepCustomClass

значение строковая константа, переменная, мнемоническая метка соответствия или выражение, которое задает величину, для которой вычисляется процентиль.

Метод **GetPercentile** возвращает для заданного значения процентиль относительно границ диапазона, установленного для объекта класса StepCustomClass. Например, если границы 0 и 1000, то GetPercentile(750) возвращает 75.

Это виртуальный метод, поэтому другие методы базового класса могут напрямую обращаться к методу GetPercentile в производном классе. Это позволяет легко реализовать свою собственную версию этого метода.

Реализация: Метод AddMarker устанавливает верхнюю и нижнюю границы для объекта класса StepCustomClass.

Тип возвращаемого значения: BYTE

Пример:

```
IF FIELD() = ?Locator                          !фокус на локаторе ?
IF EVENT() = EVENT:Accepted                    !если событие Accepted
MyBrowse.TakeAcceptedLocator                   !его обрабатывает BrowseClass
?MyList{PROP:VScrollPos}=MyStep.GetPercentile(Locator) !установить бегунок в
                                               !соответствующее положение
END
END
```

Смотри также: AddMarker

GetValue (получить по процентилю саму величину)

GetValue(*процентиль* **), VIRTUAL**

GetValue	Возвращает для заданного процентиля относительно границ диапазона, установленного для объекта класса StepCustomClass, величину соответствующую такому процентилю.
процентиль	Целочисленная константа, переменная, метка соответствия или выражение, которое задает процентиль, для которого вычисляется величина.

Метод **GetValue** возвращает для заданного процентиля относительно границ диапазона, установленного для объекта класса StepRealClass, величину, соответствующую такому процентилю. Например, если установлены границы 0 и 1000, то выражение GetValue(25) даст в результате ‘250’.

Это виртуальный метод, поэтому другие методы базового класса могут напрямую обращаться к методу GetPercentile в производном классе. Это позволяет легко реализовать свою собственную версию этого метода.

Реализация: Метод AddMarker устанавливает для объекта класса StepCustomClass верхнюю и нижнюю границы:

Тип возвращаемого значения: STRING

Пример:

```
IF FIELD() = ?MyList                                  !фокус на списке?
IF EVENT() = EVENT:ScrollDrag                         !если бегунок переместился
Locator=MyStep.GetValue(?MyList{PROP:VScrollPos})     !изменить соответственно
                                                      !локатор
```

```
END
END
```

Смотри также: AddMarker

Init (инициализировать объект класса StepCustomClass)

Init(*элем.управления*)

Init инициализирует объект класса StepCustomClass.
элем.управления целочисленные константы, переменные, мнемонические метки соответствия или выражения, идентифицирующие для объекта класса StepCustomClass:

- символы, включаемые в последовательность сортировки
- направление упорядочения (возрастающая или убывающая)

Объект класса StepCustomClass инициализируется методом **Init**.

Реализация: Метод Init устанавливает значение свойства Controls. Устанавливайте значение свойства Controls, складывая соответствующие мнемонические метки, объявленные в файле BROWSE.INC следующим образом:

```
ITEMIZE,PRE(ScrollSort)
AllowAlpha EQUATE(1)                !включить ABCDEFGHIJKLMNOPQRSTUVWXYZ
AllowAlt EQUATE(2)                  ! включить '!"J$%%^&*()"-=_+][#;~@:/.,?\|
AllowNumeric EQUATE(4)              ! включить 0123456789
CaseSensitive EQUATE(8)             ! включить abcdefghijklmnopqrstuvwxyz
Descending EQUATE(16) !             сортировка по убыванию
END
```

Пример:

```
MyStepCustomClass.Init(ScrollSort:AllowAlpha+ScrollSort:AllowNumeric)
                                                    !программный код
MyStepCustomClass.Kill
```

Смотри также: StepClass.Controls

Kill (уничтожить объект класса StepCustomClass)

Kill, VIRTUAL

Метод **Kill** освобождает любую выделенную ему в течение существования объекта память и выполняет другие действия , завершающие работу.

Метод Kill имеет атрибут VIRTUAL, поэтому другие методы базового класса могут напрямую обращаться к методу GetPercentile в производном классе. Это позволяет легко реализовать свою собственную версию этого метода.

Реализация: Метод Kill освобождает память, выделенную для свойства Custom.

Пример:

```
MyStepCustomClass.Init(ScrollSort:AllowAlpha+ScrollSort:AllowNumeric)
                                                    !программный код
MyStepCustomClass.Kill
```

ASCIIViewerClass

Обзор

Существует несколько взаимосвязанных классов, общее предназначение которых—обеспечить возможность просмотра в режиме "только чтение", пролистывания, контекстного поиска и печати файлов, включая файлы с переменной длиной записей. Хотя эти классы в первую очередь разработаны для работы с текстовыми, ASCII файлами и в них имеется ввиду использование для доступа к файлам драйвера Clarion ASCII Driver, они также работают с двоичными файлами и другими драйверами баз данных. Кроме того, эти классы можно использовать для создания других программных компонент и функциональных элементов.

Общее название этих классов, обеспечивающих просмотр файлов, ASCII Viewer классы. Вот входящие в состав ASCII Viewer классы и их предназначение:

ASCIIViewerClass	управляющий класс
ASCIIFileClass	открытие, чтение, фильтрование и индексация файлов.
ASCIIPrintClass	печать одной или нескольких строк
ASCIISearchClass	поиск и скроллинг текста

Эти файлы всесторонне описываются в оставшейся части этой главы.

Класс ASCIIViewerClass

ASCIIViewerClass использует создания одной реализации полнофункционального объекта просмотра файлов классы ASCIIFileClass, ASCIIPrintClass и ASCIISearchClass. Этот объект использует для вывода содержимого файлов на экран, прокрутки его и печати элемент управления типа LIST. Обычно, вы активизируете только объект класса ASCIIViewerClass в своей программе, который в свою очередь активизирует все необходимые для работы файлы.

Класс ASCIIFileClass

Класс ASCIIFileClass идентифицирует, открывает (только в режиме чтения), индексирует и постранично загружает содержимое файла в очередь в памяти (структуру QUEUE). Идексация ускоряет повторный доступ к записям и облегчает постраничную загрузку, которая в свою очередь позволяет просматривать очень большие по объему файлы.

Класс ASCIIPrintClass

Этот класс позволяет конечному пользователю задать диапазон строк, который должен быть напечатан, а затем печатает их. Он также обеспечивает доступ к стандартному диалогу Windows Print Setup

Класс ASCIISearchClass

Класс ASCIISearchClass позволяет осуществить контекстный поиск, обеспечивает различие прописных и строчных букв и направление поиска, а затем переход к следующему экземпляру контекста в файле.

Связь с другими классами Application Builder

Класс ASCIIViewerClass произведен от класса ASCIIFileClass, плюс к тому он связан с классами ASCIIPrintClass, ASCIISearchClass, ErrorClass и PopupClass при выполнении некоторых задач, связанных с построением пользовательского интерфейса. Поэтому, если ваша программа активизирует класс ASCIIViewerClass, то в ней также должны быть активизированы и эти классы тоже. Большая часть процесса активизации выполняется автоматически, когда вы включаете в раздел данных своей программы файл заколовков класса ASCIIViewerClass (ABASCII.INC). Смотрите раздел. Концептуальный пример.

Реализация шаблонов ABC

В шаблонах ABC объявляется локальный класс ASCIIViewerClass и реальзация объекта для каждого управляющего шаблона ASCIIViewControl. Шаблоны ABC автоматически включают все классы, необходимые для поддержки фукциональных возможностей указанных в управляющем шаблоне ASCIIViewControl.

В шаблонах создаются производный от ASCIIViewerClass класс для каждой процедуры просмотра файлов в приложении. К этим производным классам обращение производится следующим образом:

процедура:Viewer#, где процедура—это имя процедуры, а # представляет собой номер реализации управляющего шаблона ASCIIViewControl. Эти шаблоны обеспечивают производные классы, поэтому для того чтобы легко изменить поведение процедуры просмотра файлов от реализации к реализации, можно использовать закладку Classes управляющего шаблона ASCIIViewControl. Объекты получают обозначения Viewer#, где # —номер реализации управляющего шаблона. Производный класс ASCIIViewerClass является локальным по отношению к процедуре и связан с глобальным объектом класса ErrorClass.

Исходные файлы ASCII Viewer

Исходный текст класса ASCII Viewer Classes по умолчанию устанавливается в каталог \CLARION4\LIBSRC. Вот присущие классу ASCIIViewerClass исходные файлы и их назначение:

ABASCII.INC	объявление класса ASCIIFileClass объявление класса ASCIIPrintClass объявление класса ASCIISearchClass объявление класса ASCIIViewerClass
ABASCII.CLW	объявление методов класса ASCIIFileClass объявление методов класса ASCIIPrintClass объявление методов класса ASCIISearchClass объявление методов класса ASCIIViewerClass

Концептуальный пример

В следующем примере демонстрируется типовая последовательность операторов, для объявления, активизации, инициализации, использования и уничтожения объектов класса ASCIIViewerClass и связанных с ними объектов.

В этом примере конечному пользователю предосавляется возможность выбрать файл, затем просматривать, пролистывать его, осуществлять контекстный поиск и печать содержимого.

```
PROGRAM
MAP
END
INCLUDE('ABASCII.INC')                  !объявить класс ASCIIViewer Class
ViewWindow WINDOW('View a text file'),AT(3,7,296,136),SYSTEM,GRAY
LIST,AT(5,5,285,110),USE(?AsciiBox),IMM
BUTTON('&Print'),AT(5,120),USE(?Print)
BUTTON('&Search'),AT(45,120),USE(?Search)
BUTTON('&Close'),AT(255,120),USE(?Close)
END
GlobalErrors ErrorClass                 !объявить объект класса GlobalErrors
Viewer AsciiViewerClass,THREAD          !объявить объект Viewer
ViewerActive BYTE(False),THREAD !Viewer initialized flag
Filename STRING(255),THREAD !FileName variable
StartSearch LONG !hold selected line number
AsciiFile FILE,DRIVER('ASCII'),NAME(Filename),PRE(A1),THREAD
RECORD RECORD,PRE()
Line STRING(255)
```

```
END
END
CODE
GlobalErrors.Init                    !инициализировать объект класса GlobalErrors
OPEN(ViewWindow)                     !открыть окно
!Initialize Viewer with:
ViewerActive=Viewer.Init( AsciiFile, | ! file label,
A1:line, |                           ! поля файла для вывода на экран
Filename, |                          ! переменная для имени файла
?AsciiBox, |                         ! номер элем. управления типа LIST
GlobalErrors, |                      ! объект класса ErrorClass
EnableSearch+EnablePrint)            ! реализация флагов
IF ~ViewerActive THEN RETURN.        !если инициализация неудачна, не
                                     ! вызывать другие методы объекта Viewer
ACCEPT
IF EVENT() = EVENT:CloseWindow       !при завершении работы с объектом
IF ViewerActive THEN Viewer.Kill.    !уничтожить объект Viewer
END
CASE FIELD()
OF ?AsciiBox                         !по событиями элемента управления LIST
IF ViewerActive
IF Viewer.TakeEvent(EVENT())=Level:Notify THEN CYCLE. !их обрабатывает метод
TakeEvent
END
OF ?Print   !по кнопке “печать”
IF EVENT() = EVENT:Accepted
IF ViewerActive THEN Viewer.Printer.Ask.    !обратиться к методу Printer.Ask
END
OF ?Search                           !по кнопке “поиск”
IF EVENT() = EVENT:Accepted
IF ViewerActive                      !обратиться к методу Searcher.Ask
StartSearch=CHOOSE(CHOICE(?AsciiBox)>0, |    ! передавая выбранную
Viewer.TopLine+CHOICE(?AsciiBox)-1,1)   ! в данный момент строку в качестве
Viewer.Searcher.Ask(StartSearch)     ! начальной точки поиска
END
END
OF ?Close
IF EVENT() = EVENT:Accepted
POST(Event:CloseWindow)
END
END
END
GlobalErrors.Kill
```

Свойства класса AsciiFileClass

ASCIIFile (файл ASCII)

ASCIIFile &FILE

Свойство File представляет собой указатель на обрабатываемый файл. Это свойство просто идентифицирует обрабатываемый файл для различных методов класса ASCIIFileClass.

Реализация:.Это свойство инициализирует метод Init.
Смотри также: Init

ErrorMgr (объект класса ErrorClass)

ErrorMgr &ErrorClass, PROTECTED

Свойство ErrorMgr представляет собой указатель на объект класса ErrorClass для данного объекта класса ASCIIFileClass. Объект класса ASCIIFileClass использует его для обработки разнообразных ошибочных ситуаций и условий, которые встречаются при обработке файла.

Реализация: Метод Init инициализирует свойство ErrorMgr.
Смотри такжк: Init

Методы класса AsciiFileClass

Функциональная организация—предполагаемое использование

С точки зрения облегчения понимания работы класса ASCIIFileClass полезно разбить его методы на две большие категории, по их предполагаемому использованию—методы первичного интерфейса и виртуальные методы. Такое разделение отражает наше понимание предолагаемого использования методов.

Методы первичного интерфейса

Методы первичного интерфейса, которые обычно вызываются из вашей программы совершенно обыкновенным образом, можно еще подразделить на три категории.

Методы внутреннего (однократного) использования:

Init инициализация объекта класса ASCIIViewerClass

SetTranslator	установить перекодировшик выводимого в окне текста
Kill	уничтожить объект класса ASCIIViewerClass

Методы основного применения:

GetLastLineNo	получить номер последней строки
GetLine	получить строку текста
GetPercentile	пробразовать текущее положение в процентиль
SetPercentile	пробразовать процентиль в текущее положение

Методы эпизодического использования:

GetFilename	получить имя файла
Reset	осуществить “сброс” режимов объекта ASCIIViewerClass

Виртуальные методы

Обычно, к этим методам напрямую не обращаются—к ним обращаются методы первичного интерфейса. Однако, мы предвидим, что у вас появится желание переопределить эти методы, и потому они являются виртуальными, очень легко переопределяемые. В случае, когда вы их не переопределяете, эти методы обеспечиывают разумное поведение принимаемое по умолчанию.

GetDOSFilename	предлагает конечному пользователю выбрать файл для работы
FormatLine	виртуальный метод для форматирования текста
SetLine	установиться на заданную строку
ValidateLine	виртуальный метод для реализации фильтрации записей

FormatLine (виртуальный метод для форматирования текста)

FormatLine(строка [, номер строки] **), PROTECTED, VIRTUAL**

FormatLine	Виртуальый метод-заполнитель для форматирования текста.
строка	Имя переменной типа STRING содежащей текст для форматирования
номер строки	Целочисленная константа, переменная или мнемоническая метка соответствия, или выражение, содержащее смещение или позицию строки текста, которую надлежит отформатировать. Если этот параметр опущен, то метод работает с текущей строкой

Метод FormatLine представляет собой виртуальный метод-заполнитель, выполняющий форматирование текста перед выводом его на экран.

FormatLine является виртуальным методом, поэтому другие методы базового класса могут вызывать напрямую метод FormatLine в производном классе. Это позволяет вам легко реализовать свою собственную версию этого метода

Этот метод имеет атрибут PROTECTED, поэтому на него могут ссылаться только методы класса ASCIIFileClass, или производных от него методов.

Реализация: Метод FormatLine—метод-заполнитель для производных классов. Он обеспечивает легкий способ форматирования текста перед выводом его на экран. Метод GetLine method обращается к методу FormatLine.

Пример:

```
INCLUDE('ABASCII.INC')                          !объявить класс ASCIIViewerClass
MyViewer CLASS(AsciiViewerClass),TYPE           !производный класс MyViewer
FormatLine PROCEDURE(*STRING),VIRTUAL           !прототип виртуального метода
FormatLine
END
Viewer MyViewer,THREAD                          !объявить объект Viewer
AsciiFile FILE,DRIVER('ASCII'),NAME('MyText'),PRE(A1),THREAD
RECORD RECORD,PRE()
Line STRING(255)
END
END
CODE
!текст программы
MyViewer.FormatLine PROCEDURE(*STRING line)
CODE
line = line[1:5]' '&line[5:55]                  !форматировать текст
```

Смотри также: GetLine

GetDOSFilename (дать конечному пользователю выбрать файл)

GetDOSFilename(Имя_файла **), VIRTUAL**

GetDOSFilename	Предлагает конечному пользователю выбрать файл для просмотра.
Имя_файла	Метка атрибута NAME переменной, соответствующей свойству ASCIIFile, которой присваивается имя выбранного файла.

Метод **GetDOSFilename** предлагает конечному пользователю выбрать файл для просмотра и возвращает значение, указывающее выбрал пользователь файл или нет. Возвращаемое значение равное единице (1) означает, что файл был выбран и переменная *Имя_файла* содержит путь к нему и его имя, а возвращаемое значение ноль (0), указывает, что пользователь не выбрал файл и переменная *Имя_файла* пуста.

GetDOSFilename является виртуальным методом, поэтому другие методы базового класса могут напрямую обращаться к этому методу в производном классе. Это позволяет пользователю легко реализовать свою собственную версию этого метода.

Реализация: Метод GetDOSFileName использует для запроса и получения от конечного пользователя имени файла объект SelectFileClass.

Тип возвращаемого значения: BYTE

Пример:

```
MyAsciiFileClass.Reset FUNCTION(*STRING FName)
RVal BYTE(True)
SavePath CSTRING(FILE:MaxFilePath+1),AUTO
CODE
CLOSE(SELF.AsciiFile)
SavePath=PATH()
LOOP
IF ~FName AND ~SELF.GetDOSFilename(FName)
RVal=False
BREAK
END
OPEN(SELF.AsciiFile,ReadOnly+DenyNone)
IF ERRORCODE()
MESSAGE('Can't open ' & FName)
RVal=False
ELSE
BREAK
END
END
IF RVal
SELF.FileSize=BYTES(SELF.AsciiFile)
END
SETPATH(SavePath)
RETURN RVal
```

Смотри также: ASCIIFile, SelectFileClass

GetFilename (получить имя файла)

GetFilename

Метод **GetFilename** возвращает имя ASCII файла.

Реализация: метод GetFileName использует функцию NAME. Детальную

информацию об этой функции смотри в *Руководстве по языку Clarion.*

Тип возвращаемого значения: STRING

Пример:

```
INCLUDE('ABASCII.INC')                  !объявить класс ASCIIViewerClass
Viewer AsciiViewerClass,THREAD          !объявить объект Viewer
Filename STRING(255),THREAD             !объявить переменную для имени файла
AsciiFile FILE,DRIVER('ASCII'),NAME(Filename),PRE(A1),THREAD
RECORD RECORD,PRE()
Line STRING(255)
END
END
CODE
!программный код
MESSAGE('Filename:'&Viewer.GetFilename())      !получить имя ASCII файла
```

GetLastLineNo (получить номер последней строки)

GetLastLineNo

Метод **GetLastLineNo** возвращает номер последней строки в файле и индексирует весь файл.

У этого метода имеется атрибут PROC, поэтому можно обращаться к нему как к процедуре и игнорировать возвращаемое значение.

Тип возвращаемого значения: LONG

Пример:

```
MyViewer.TakeScroll PROCEDURE(UNSIGNED EventNo)
LineNo LONG
CODE
IF FIELD()=SELF.ListBox
IF EVENT() = EVENT:ScrollBottom                  !при скроллинге в конец файла
LineNo = SELF.GetLastLineNo()                    !индексировать до конца файла
SELF.DisplayPage(LineNo-SELF.ListBoxItems+1)     !вывести последнюю страницу
SELECT(SELF.ListBox,SELF.ListBoxItems)           !выделить последнюю сроку
END
END
```

GetLine (получить строку текста)

GetLine(номер строки **), PROC**

GetLine возвращает строку текста

номер Целочисленная константа, переменная, мнемоническая метка соответствия или выражение, которое содержит смещение или положение требуемой строки текста.

Метод **GetLine** возвращает определенную параметром *номер* строку текста.

Этот метод имеет атрибут PROC, поэтому к нему можно обращаться как к процедуре и игнорировать возвращаемое значение.

Реализация: Метод GetLine берет строку с заданным *номером* из ASCII файла, при необходимости расширяя очередь индексов. Если очередь индексов уже содержит требуемый номер строки, то происходит чтение файла по уже имеющемуся смещению, в противном случае расширяет индексный список. Если строка с заданным номером отсутствует в файле, то переменная, куда должна быть возвращена строк очищается и устанавливается значение функции ERRORCODE().

Тип возвращаемого значения: STRING

Пример:

```
MyViewer.DisplayPage PROCEDURE(LONG LineNo)
LineOffset USHORT,AUTO
CODE
IF LineNo > 0                                    !строка задана?
SELF.ListBoxItems=SELF.ListBox{PROP:Items}       !отметим размер окна списка
FREE(SELF.DisplayQueue)                          !очистим выводимую очередь
SELF.GetLine(LineNo+SELF.ListBoxItems-1)         !индекс записи в конце страницы
LOOP LineOffset=0 TO SELF.ListBoxItems-1         !для каждой строки окна списка
SELF.DisplayQueue.Line=SELF.GetLine(LineNo+LineOffset) !считываем запись
IF ERRORCODE()                                   !по концу файла
BREAK                                            ! прекращаем
END
ADD(SELF.DisplayQueue)                           !добавим к выводимой очереди
END
SELF.TopLine=LineNo                              !отметим первую запись
DISPLAY(SELF.ListBox)                            !перерисуем окно списка
END
```

Смотри также: GetLine

GetPercentile (преобразовать номер записи в процентиль)

GetPercentile(номер **)**

GetPercentile Возвращает процентиль от заданного номера строки

номер строки Целочисленная константа, переменная, метка соответствия или

выражение, которое содержит смещение или положение, которое следует преобразовать в процентиль.

Метод **GetPercentile** возвращает указанную ему позицию записи в виде приблизительного процента, который используется для позиционирования бегунка на вертикальной линейке скроллинга.

Тип возвращаемого значения: USHORT

Пример:

```
SetThumb ROUTINE
        !номер текущей строки в процентах от общего количества записей в
        !файле
PctPos=MyASCIIFile.GetPercentile(MyASCIIFile.TopLine+CHOICE(?ASCIIBox)-1)
?ASCIIBox{PROP:VScrollPos}=PctPos
        !установить в соответствующую позицию бегунок
```

Init (инициализировать объект класса ASCIIFileClass)

Init(файл, поле [,имя_файла] , обработчик_ошибок **)**

Init	инициализирует объект класса ASCIIFileClass.
файл	метка файла подлежащего просмотру.
поле	Полный путь и имя файла в файловой системе.
Имя_файла	Имя переменной, указанной атрибутом NAME структуры FILE. Если этот параметр опущен, то атрибут NAME содержит константу. Если указана пустая строка (''), то метод Init выдает запрос пользователю на выбор файла.
error handler	Метка объекта класса ErrorClass для управления ошибочными ситуациями, возникающими для этого объекта класса ASCIIFileClass.

Метод **Init** инициализирует объект класса ASCIIFileClass и возвращает значение означающее успешно ли осуществляется доступ к файлу и готов ли файл для обработки.

Реализация: метод Init возвращает единицу (1), если доступ к файлу возможен и файл готов для обработки и возвращает ноль (0) и обращается к методу Kill, в том случае, если доступ к файлу невозможен и файл нельзя обрабатывать.

Если метод Init возвращает ноль (0), то объект класса ASCIIFileClass не инициализируется и вы не можете обращаться к методам этого объекта.

Тип возвращаемого значения: BYTE

Пример:

```
Filename STRING(255),THREAD          !объявим переменную для имени файла
FileActive BYTE                      !объявим переключатель успещно/неуспешно
AsciiFile FILE,DRIVER('ASCII'),NAME(Filename),PRE(A1)
RECORD RECORD,PRE()
Line STRING(255)
END
END
CODE                                 !инициализировать объект класса
                                     !ASCIIFileClass:
FileActive=ASCIIFile.Init(AsciiFile, |   !имя файла
A1:Line, |                           ! поля файла для высвечивания
Filename, |                          ! переменная атрибута NAME
GlobalErrors)                        ! Объект класса ErrorClass
IF ~FileActive THEN RETURN.          !если инициализация неудачна, не работаем
ACCEPT                               ! если успешна, обрабатываем файл
IF EVENT() = EVENT:CloseWindow
IF FileActive THEN ASCIIFile.Kill.
END
!программный код
END
```

Смотри также: Kill

Kill (уничтожить объект класса ASCIIFileClass)

Kill

Метод **Kill** освобождает всю память, выделенную в течение времени существования объекта и выполняет другие необходимые действия по завершению работы.

Пример:

```
Filename STRING(255),THREAD          !объявить переменную для имени файла.
FileActive BYTE                      !объявить переключатель успешно/неуспешно
AsciiFile FILE,DRIVER('ASCII'),NAME(Filename),PRE(A1)
RECORD RECORD,PRE()
Line STRING(255)
END
END
CODE                                 !инициализировать объект класса
                                     !ASCIIFileClass
FileActive=ASCIIFile.Init(AsciiFile, |   ! имя файла
```

```
A1:Line, |                          ! поля для высвечивания
Filename, |                         ! переменная атрибута NAME
GlobalErrors)                       ! объект класса ErrorClass
IF ~FileActive THEN RETURN.         !если инициализация неудачна
ACCEPT                              !если успешна - работаем
IF EVENT() = EVENT:CloseWindow
IF FileActive THEN ASCIIFile.Kill.
END
!программный код
END
```

Reset (осуществить "сброс" объекта ASCIIFileClass)

Reset(имя_файла **)**

Reset	сбрасывает объект класса ASCIIFileClass
имя_файла	метка переменной атрибута NAME свойства объекта ASCIIFile

Метод **Reset** осуществляет сброс объекта класса ASCIIFileClass и возвращает значение, указывающее выбрал пользователь файл или нет. Возвращаемое значение единица (1) означает, что файл выбран и переменная *Имя_файла* содержит путь и имя файла; возвращаемое значение ноль (0), означает, что пользователь файл не выбрал, и эта переменная пуста.

Реализация: Метод Reset обращается к методу GetDOSFileName для того, чтобы запросить у конечного пользователя имя просматриваемого файла. Затем он открывает файл и "сбрасывает" статистические переменные и флажки, связанные с выбранным файлом.

Тип возвращаемого значения: BYTE

Пример:

```
AsciiViewerClass.Reset FUNCTION(*STRING Filename)
CODE
FREE(SELF.DisplayQueue)
DISPLAY(SELF.ListBox)
IF ~PARENT.Reset(Filename) THEN RETURN False.
SELF.TopLine=1
SELF.DisplayPage
SELECT(SELF.ListBox,1)
RETURN True
```

Смотри также: ASCIIFile, GetDOSFilename

SetLine (виртуальный метод для позиционирования)

SetLine(номер_строки **), PROTECTED, VIRTUAL**

SetLine	виртуальный метод-заполнитель для позиционирования внутри файла.
номер_строки	смещение или положение строки внутри файла

Метод **SetLine** является виртуальным методом-заполнителем, с помощью которого производится позиционирование внутри файла.

Он является виртуальным, поэтому другие методы базового класса могут напрямую обращаться к методу SetLine в производном классе. Это позволяет легко реализовать свою собственную версию этого метода. Этот метод имеет атрибут PROTECTED, поэтому к нему могут обращаться только методы класса ASCIIFileClass или производных от него классов. SetLine обращаются методы SetPercentile, ASCIIViewerClass.AskGotoLine и ASCIISearchClass.Ask .

Пример:

```
MyViewerClass.SetLine PROCEDURE(LONG LineNo)
                                        !синхронизировать окно списка с номером
                                        !строки
CODE
SELF.DisplayPage(LineNo)                !листание до LineNo
                                        !выделить строку LineNo
SELECT(SELF.ListBox,CHOOSE(SELF.TopLine=LineNo,1,LineNo-SELF.TopLine+1))
```

Смотри также: SetPercentile, ASCIIViewerClass.AskGoToLine, ASCIISearchClass.Ask

SetPercentile (установить относительное положение в файле)

SetPercentile(процентиль **)**

SetPercentile	положение в файле устанавливается на запись, ближайшую к определенному по формуле *размер_файла* * *процентиль* / 100 положению.
процентиль	значение между 0 и 100, указывающее относительное положение внутри файла. Эта величина может устанавливаться на основании положения бегунка на вертикальной линейке скроллинга.

Метод **SetPercentile** устанавливает положение внутри файла на запись, положение которой ближе всего к вычисленному по формуле: *Размер_файла* * *процентиль* / 100. Этот метод можно использовать для установки положения в файле,

соответствующего положению бегунка на вертикальной линейке прокрутки.

Реализация: метод SetPercentile устанавливает заданное положение в файле (обычно на основании положения бегунка на вертикальной линейке скроллинга). При необходимости метод SetPercentile расширяет список индексов и обращается к виртуальному методу SetLine для установки положения в файле. Он вычисляет положение делением параметра *процентиль* на 100 и последующим умножением результата на размер файла.

Пример:

```
MyViewerClass.TakeDrag PROCEDURE(UNSIGNED EventNo)
CODE
IF FIELD()=SELF.ListBox
IF EventNo = EVENT:ScrollDrag
SELF.SetPercentile(SELF.ListBox{PROP:VScrollPos})      !переместиться по полож-ю
                                                       !бегунка
END
END
```

Смотри также: SetLine

ValidateLine (виртуальный метод для фильтрования записей)

ValidateLine(строка **), PROTECTED, VIRTUAL**

ValidateLine виртуальный метод-заполнитель для реализации фильтра.
строка смещение или положение строки текста для проверки ее пригодности.

Метод **ValidateLine** является виртуальным методом-заполнителем для реализации фильтра записей. Он возвращает единицу (1), если запись подходит, и ноль (0)— если ее следует исключить из просмотра.

Это виртуальный метод, поэтому другие методы базового класса могут напрямую обращаться к методу ValidateLine в производном классе. Это позволяет легко реализовать свою собственную версию метода ValidateLine.

Он имеет атрибут PROTECTED, поэтому к нему могут обращаться только методы класса ASCIIFileClass или производных от ASCIIFileClass классов.

Реализация: Метод ValidateLine является методом-заполнителем для производных классов. Класс ASCIIFileClass обращается к этому методу при первоначальном считывании записи.

Тип возвращаемого значения: BYTE

Пример:

```
MyFileClass.ValidateLine FUNCTION(STRING LineToTest)
CODE
IF LineToTest[1] = '!'                !проверить на ! в первой колонке
RETURN False                          !исключить строки с !
ELSE
RETURN True                           !все другие включить в просмотр
END
```

Свойства класса AsciiViewerClass

Класс AsciiViewerClass наследует все свойства класса AsciiFileClass, от которого он произведен. Более подробную информацию об этих свойствах вы найдете в разделе *Свойства класса AsciiFileClass.*

Дополнительно к унаследованным свойствам класс AsciiViewerClass имеет свойства перечисленные далее.

TopLine (первая строка текущего экрана)

```
TopLine UNSIGNED
```

Свойство **TopLine** содержит смещение или положение первой строки отображаемого в данный момент объектом ASCIIViewerClass экрана внутри просматриваемого файла. Объект класса ASCIIViewerClass использует свойство TopLine для управления прокруткой и позиционирования бегунка на линейке скроллинга.

Popup (объект класса PopupClass)

```
Popup &PopupClass
```

Свойство **Popup** представляет собой указатель на объект класса PopupClass для данного объекта класса ASCIIViewerClass. Объект класса ASCIIViewerClass использует свойство Popup для определения и управления работой всплывающего меню.

Реализация: Это свойство инициализируется в методе Init.
Смотри также: Init

Printer (Объект класса ASCIIPrintClass)

Printer &ASCIIPrintClass

Свойство **Printer** указывает на объект класса ASCIIPrintClass для данного объекта класса ASCIIViewerClass. Свойство Printer используется при запросе диапазона печатаемых строк и указаний от конечного пользователя, и последующей печати по этим указаниям.

Реализация: свойство Printer инициализирую методы AddItem и Init.
Смотри также: AddItem, Init

Searcher (объект класса ASCIISearchClass)

Searcher &ASCIISearchClass

Свойство **Searcher** представляет собой указатель на объект класса ASCIISearchClass для данного объекта класса ASCIIViewerClass. Объект класса ASCIIViewerClass использует это свойство для запроса поискового значения у конечного пользователя и, затем, поиска его в просматриваемом файле.

Реализация: Значение свойства Searcher устанавливают методы AddItem и Init.
Смотри также: AddItem, Init

Методы класса AsciiViewerClass

Класс AsciiViewerClass наследует все методы класса AsciiFileClass, от которого он произведен. Более подробную информацию об этих методах вы найдете в разделе *Методы класса AsciiFileClass*

Дополнительно к унаследованным (или вместо некоторых из них) класс AsciiViewerClass содержит перечисленные ниже методы.

Функциональная организация—предполагаемое использование

В целях облегчения понимания работы класса ASCIIViewerClass и в соответствии с их предполагаемым использованием полезно разделить все его методы на две большие категории—методы первичного интерфейса и виртуальные методы. Такое разделение отражает, то как разработчики предполагали типовое использование методов класса ASCIIViewerClass.

Методы первичного интерфейса

Методы первичного интерфейса, обращение к которым будет весьма обычным делом для разработчика программ, можно в свою очередь разделить на три группы:

Методы внутреннего (однократного) использования:

Init	инициализация объекта класса ASCIIViewerClass
Kill	уничтожить объект класса ASCIIViewerClass

Методы основного применения:

AskGotoLine	переход к указанной пользователем строке
DisplayPage	отобразить новую страницу
PageDown	перейти к следующей странице
PageUp	перейти к предыдущей странице
TakeEvent	обработать событие в цикле ACCEPT

Методы эпизодического использования:

AddItem	добавить объект для печати или поиска
GetFilename(I)	получить имя файла
GetLastLineNo(I)	получить номер последней строки
GetLine (I)	получить строку текста
GetPercentile(I)	преобразовать смещение в файле в процентиль
Reset	установить начальные значения свойствобъекта ASCIIViewerClass
SetPercentile(I)	преобразовать процентиль в смещение.
SetLine(V)	встать на заданную строку
SetLineRelative	переместиться на N строк

I - Эти методы наследуются от класса ASCIIFileClass.
V - Эти методы к тому же являются виртуальными.

Виртуальные методы

Обычно вам не приходится непосредственно обращаться к этим методам—к ним обращаются методы первичного интерфейса. Однако, мы предвидим, что у вас появится желание переопределить эти методы, и потому они являются виртуальными, что делает их очень легко переопределяемыми. В случае, когда вы их не переопределяете, эти методы по обеспечивают разумное поведение, принимаемое по умолчанию.

FormatLine(I)	форматировать текст
SetLine	встать на заданную строку
ValidateLine(I)	реализовать фильтр

I - Эти методы наследуются от класса ASCIIFileClass.

AddItem (программировать объект класса AsciiViewer)

AddItem(| принтер | **)**
| поиск |

AddItem добавляет заданные функциональные возможности к объекту AsciiViewer.
принтер имя объекта класса AsciiPrintClass.
поиск имя объекта класса AsciiSearchClass.

Метод **AddItem** добавляет указанные функциональные возможности к объекту AsciiViewer.

Эти методы предоставляют альтернативу методу Init при добавлении или изменении возможностей объекта AsciiViewer по печати содержимого файла и контекстному поиску в нем.

Реализация: Метод AddItem устанавливает значения свойств Printer или Searcher, инициализирует объекты печати и поиска, затем делает доступным соответствующее всплывающее меню.

Пример:

```
MyPrinter CLASS(AsciiPrintClass)        !объявить объект печати
NewMethod PROCEDURE
END
MySearcher CLASS(AsciiSearchClass) !объявить объект поиска
NewMethod PROCEDURE
END
CODE
Viewer.Init(AsciiFile,A1:line,Filename,?AsciiBox,GlobalErrors)
Viewer.AddItem(MyPrinter)               !добавить возможность печати
Viewer.AddItem(MySearcher)              ! добавить возможность поиска
```

Смотри также: Init, Printer, Searcher

AskGotoLine (перейти к заданной пользователем строке)

AskGotoLine

Метод **AskGotoLine** просит конечного пользователя ввести номер строки, которую необходимо отобразить, затем встает на строку, ближайшую к заданной.

Реализация : объект класса ASCIIViewerClass вызывает метод AskGotoLine из всплывающего по щелчку правой кнопки мыши меню. Для позиционирования внутри файла на заданную запись метод AskGotoLine обращается к методу SetLine.

Пример:

```
MyViewerClass.TakeEvent PROCEDURE(UNSIGNED EventNo)
CODE
CASE EventNo
OF EVENT:AlertKey
IF KEYCODE()=MouseRight
CASE SELF.Popup.Ask()
OF 'Print'
SELF.Printer.Ask
OF 'Goto'
SELF.AskGotoLine
...
```

Смотри также: SetLine

DisplayPage (отобразить новую страницу)

DisplayPage([номер_строки] **)**

DisplayPage Отобразить новую страницу содержимого файла.

номер_строки Целочисленная константа, переменная, мнемоническая метка соответствия или выражение, которое содержит смещение или позицию строки текста внутри файла. Если этот параметр опущен, то по умолчанию используется значение свойства TopLine.

Метод **DisplayPage** отображает новую страницу строк из файла. Происходит отображение фрагмента текста начинающегося с заданной параметром *номер_строки* или, если этот параметр опущен, значением свойства TopLine.

Пример:

```
MyViewerClass.Reset PROCEDURE(*STRING Filename)
CODE
FREE(SELF.DisplayQueue)
DISPLAY(SELF.ListBox)
PARENT.Reset(Filename)
SELF.TopLine=1
SELF.DisplayPage
SELECT(SELF.ListBox,1)
```

Смотри также: TopLine

Init (инициализировать объект класса ASCIIViewerClass)

Init(файл, поле, [имя файла] , окно списка, обработчик ошибок [, возможность] **)**

Init	инициализировать объект класса ASCIIViewerClass.
файл	имя структуры FILE для чтения файла.
поле	точно квалифицированное имя высвечиваемого поля в файле.
имя_файла	метка переменной из атрибута NAME структуры FILE. Если этот параметр опущен, то атрибута NAME структуры FILE содержит константу. Если этот параметр—пустая строка, то метод Init просит конечного пользователя выбрать файл для просмотра.
окно_списка	целочисленная константа, переменная, мнемоническая метка соответсвия или выражение, содержащее номер элемента управления в окне, соответствующий окну списка, в котором происходит просмотр файла.
обработчик ошибок	имя объекта ErrorClass для управления обработкой ошибочных ситуаций, встречаемых объектом класса ASCIIViewerClass.
возможность	целочисленная константа, переменная выражение или мнемоническая метка соответствия, которая информирует объект класса ASCIIViewerClass о том, какие функциональные возможности необходимо реализовать, например, печать (EnablePrint), поиск (EnableSearch) или и то, и другое. Если этот параметр опущен, то не реализуются никакие дополнительные функциональные возможности.

Метод **Init** инициализирует объект класса ASCIIViewerClass и возвращает значение, означающее, успешен ли доступ к просматриваемому файлу и готов ли он для просмотра.

Реализация: Метод Init возвращает единицу (1), если доступ к файлу возможен и он готов для обработки, и возвращает ноль (0) и вызывает метод Kill в противном случае. Если метод Init возвращает ноль (0), объект класса ASCIIViewerClass не инициализируется, и вам не следует вызывать его методы.

Можно установить дополнительные функциональные *возможности* с помощью следующих мнемонических имен, объявленных в файле ASCII.INC. Для реализации этих возможностей по отдельности подставьте соответствующее мнемоническое имя. А если нужны обе эти возможности, то подставьте в качестве параметра метода Init сумму этих значений.

```
EnableSearch BYTE(001b)
EnablePrint BYTE(010b)
```

Тип возвращаемого значения: BYTE

Пример:

```
PROGRAM
MAP
END
INCLUDE('ABASCII.INC')                    !объявить объект класса ASCIIViewer
ViewWindow WINDOW('View an ASCII File'),AT(3,7,296,136),SYSTEM,GRAY
LIST,AT(5,5,285,110),USE(?AsciiBox),IMM
BUTTON('&Print'),AT(5,120),USE(?Print)
BUTTON('&Search'),AT(45,120),USE(?Search)
BUTTON('&Close'),AT(255,120),USE(?Close)
END
GlobalErrors ErrorClass                   !объявить объект класса GlobalErrors
Viewer AsciiViewerClass,THREAD            ! объявить объект Viewer
ViewerActive BYTE(False),THREAD           !флаг инициализации объекта Viewer
Filename STRING(255),THREAD               !Переменная для имени файла
AsciiFile FILE,DRIVER('ASCII'),NAME(Filename),PRE(A1),THREAD
RECORD RECORD,PRE()
Line STRING(255)
END
END
CODE
GlobalErrors.Init                         !инициализировать объект GlobalErrors
OPEN(ViewWindow)                          !открыть окно
                                          !инициализировать просмотр с параметрами:
ViewerActive=Viewer.Init( AsciiFile, |    ! метка файла,
A1:line, |    ! поле для вывода
Filename, |                               ! переменная из атрибута NAME
?AsciiBox, |                              ! номер элемента LIST
GlobalErrors, |                           ! объект ErrorClass
EnableSearch+EnablePrint)                 ! флаг для реализации возможностей
IF ~ViewerActive THEN RETURN.             !если инициализация неудачна, то
                                          ! не обращаться к методам объекта Viewer
ACCEPT                                    !Если успешно инициализирован то работаем
IF EVENT() = EVENT:CloseWindow
IF ViewerActive THEN Viewer.Kill.         !если инициализация была-уничтожим объект
END
!программный код
END
```

Смотри также: Kill

Kill (уничтожить объект класса ASCIIViewerClass)

Kill

Метод **Kill** освобождает всю память, выделенную во время существования объекта и выполняет другие требуемые для завершения работы объекта действия.

Пример:

```
PROGRAM
MAP
END
INCLUDE('ABASCII.INC')                  !объявить класс ASCIIViewer
ViewWindow WINDOW('View an ASCII File'),AT(3,7,296,136),SYSTEM,GRAY
LIST,AT(5,5,285,110),USE(?AsciiBox),IMM
BUTTON('&Print'),AT(5,120),USE(?Print)
BUTTON('&Search'),AT(45,120),USE(?Search)
BUTTON('&Close'),AT(255,120),USE(?Close)
END
GlobalErrors ErrorClass                 !объявить объект класса GlobalErrors
Viewer AsciiViewerClass,THREAD          ! объявить объект Viewer
ViewerActive BYTE(False),THREAD         !флаг инициализации объекта Viewer
Filename STRING(255),THREAD             !переменная для имени файла
AsciiFile FILE,DRIVER('ASCII'),NAME(Filename),PRE(A1),THREAD
RECORD RECORD,PRE()
Line STRING(255)
END
END
CODE
GlobalErrors.Init                       !инициализировать объект GlobalErrors
OPEN(ViewWindow)                        !открыть окно
!инициализировать объект Viewer с параметрами:
ViewerActive=Viewer.Init( AsciiFile, |  ! метка структуры FILE,
A1:line, |   ! поле файла для вывода
Filename, |                             ! переменная из атрибута NAME
?AsciiBox, |                            ! номер элемента LIST
GlobalErrors, |                         ! объект ErrorClass
EnableSearch+EnablePrint)               ! флаг для доп. возможностей
IF ~ViewerActive THEN RETURN.           !если инициализация неудачна, то не
                                        ! обращаться к другим методам Viewer
ACCEPT                                  !если инициализация удачна-работаем
IF EVENT() = EVENT:CloseWindow
IF ViewerActive THEN Viewer.Kill.
END
!программный код
END
```

PageDown (перейти к следующей странице)

PageDown, PROTECTED

Метод **PageDown** перелистывает одну "страницу" вперед. Страница представляет собой ряд строк, выводимых в окне списка объекта класса ASCIIViewerClass.

Этот метод имеет атрибут PROTECTED, поэтому к нему могут обращаться только методы данного класса ASCIIViewerClass или производного от него класса.

Пример:

```
MyViewerClass.TakeEvent PROCEDURE(UNSIGNED EventNo)
CODE
IF FIELD()=SELF.ListBox
CASE EventNo
OF EVENT:Scrollup
SELF.SetLineRelative(-1)
OF EVENT:ScrollDown
SELF.SetLineRelative(1)
OF EVENT:PageUp
SELF.PageUp
OF EVENT:PageDown
SELF.PageDown
END
END
```

PageUp (перейти к предыдущей странице)

PageUp, PROTECTED

Метод **PageUp** перелистывает одну "страницу" назад. Страница представляет собой ряд строк, выводимых в окне списка объекта класса ASCIIViewerClass.

Этот метод имеет атрибут PROTECTED, поэтому к нему могут обращаться только методы данного класса ASCIIViewerClass или производного от него класса.

Пример:

```
MyViewerClass.TakeEvent PROCEDURE(UNSIGNED EventNo)
CODE
IF FIELD()=SELF.ListBox
CASE EventNo
OF EVENT:Scrollup
```

```
SELF.SetLineRelative(-1)
OF EVENT:ScrollDown
SELF.SetLineRelative(1)
OF EVENT:PageUp
SELF.PageUp
OF EVENT:PageDown
SELF.PageDown
END
END
```

Reset (перенастроить объект ASCIIViewerClass)

Reset(имя_файла **)**

Reset перенастраивает объект ASCIIViewerClass.
имя_файла Имя переменной из атрибута NAME свойства объекта ASCIIFile.

Метод **Reset** перенастраивает объект ASCIIViewerClass на просмотр другого файла и возвращает значение, показывающее выбрал ли конечный пользователь файл или нет. Возвращаемое значение равное единице (1) означает, что пользователь файл выбрал, а параметр *имя_файла* содержит его путь, значение равное нулю (0) указывает, что файл выбран не был, а указанная переменная пуста.

Реализация: метод Reset освобождает использовавшуюся для вывода информации очередь и обращается к методу ASCIIFileClass.Reset для получения от пользователя нового имени файла. Затем, если новый файл выбран, он заново заполняет очередь и заново выводит на экран окно списка.

Тип возвращаемого значения: BYTE

Пример:

```
AsciiFileClass.Init FUNCTION|
(FILE AsciiFile,*STRING FileLine,*STRING FName,ErrorClass ErrorHandler)
CODE
SELF.AsciiFile&=AsciiFile
SELF.Line&=FileLine
SELF.ErrorMgr&=ErrorHandler
SELF.IndexQueue&=NEW(IndexQueue)
IF ~SELF.Reset(FName)
SELF.Kill
RETURN False
END
RETURN True
```

Смотри также: ASCIIFile, ASCIIFileClass.Reset

SetLine (позиционироваться на заданную строку)

SetLine(номер_строки **), PROTECTED, VIRTUAL**

SetLine	позиционирует объект ASCIIViewerClass на заданную строку.
номер_строки	целочисленная константа, переменная, мнемоническая метка соответствия или выражение, содержащее смещение или положение строки текста, на которую нужно «встать».

Метод **SetLine** позиционирует объект ASCIIViewerClass на заданную строку внутри просматриваемого файла.

Метод SetLine является виртуальным методом, поэтому другие методы базового класса могут напрямую обращаться к методу SetLine в производном классе. Это позволяет легко реализовать свою собственную версию данного метода.

Этот метод имее атрибут PROTECTED, поэтому к нему могут обращаться только методы класса ASCIIViewerClass или производных от него классов.

Реализация: Методы AskGotoLine method, ASCIIFileClass.SetPercentile, и ASCIISearchClass.Ask используют метод SetLine для установки на требуемую строку текста.

Пример:

```
MyViewerClass.AskGotoLine PROCEDURE
LineNo LONG,STATIC
OKGo BOOL(False)
GotoDialog WINDOW('Goto'),AT(,,96,38),GRAY,DOUBLE
SPIN(@n_5),AT(36,4,56,13),USE(LineNo),RANGE(1,99999)
PROMPT('&Line No:'),AT(4,9,32,10),USE(?Prompt1)
BUTTON('&Go'),AT(8,22,40,14),USE(?GoButton)
BUTTON('&Cancel'),AT(52,22,40,14),USE(?CancelButton)
END
CODE
OPEN(GotoDialog)
ACCEPT
CASE EVENT()
OF EVENT:Accepted
CASE ACCEPTED()
OF ?GoButton
OKGo=True
```

```
POST(EVENT:CloseWindow)
OF ?CancelButton
POST(EVENT:CloseWindow)
END
END
END
CLOSE(GotoDialog)
IF OKGo THEN SELF.SetLine(LineNo).
```

Смотри также: AskGoToLine, ASCIIFileClass.SetPercentile, ASCIISearchClass.Ask

SetLineRelative (переместиться на n строк)

SetLineRelative(n), PROTECTED

SetLineRelative	позиционирует объект ASCIIViewerClass относительно текущего положения.
n	целочисленная константа, переменная, выражение или мнемоническая метка соответствия, содержащая число строк, на которое нужно переместиться относительна текущего положения. Положительное число задает движение вперед, отрицательное вызывает перемещение в направлении к началу файла.

Метод **SetLineRelative** переустанавливает объект класса ASCIIViewerClass на n строк относительно текущего положения.

Этот метод имеет атрибут PROTECTED, поэтому к нему могут обращаться только методы класса ASCIIViewerClass или производных от него классов.

Пример:

```
MyViewerClass.TakeScrollOne PROCEDURE(UNSIGNED EventNo)
CODE
IF FIELD()=SELF.ListBox
CASE EventNo
OF EVENT:Scrollup
SELF.SetLineRelative(-1)
OF EVENT:ScrollDown
SELF.SetLineRelative(1)
END
END
```

SetTranslator (установить объект-транслятор)

SetTranslator(транслятор **)**

SetTranslator	Устанавливает объект классаTranslatorClass для данного объекта AsciiViewerClass
транслятор	Имя объекта класса TranslatorClass для данного объекта AsciiViewerClass.

Метод **SetTranslator** устанавливает для данного объекта объект класса TranslatorClass. Задав имя объекта-транслятора, вы автоматически перекодируете любой текст в окне или выпадающем меню, отображаемый при просмотре файла.

Реализация: Метод SetTranslator устанавливает объект класса TranslatorClass для объектов класса PopupClass, AsciiPrintClass и AsciiSearchClass.

Пример:

```
Viewer AsciiViewerClass                    !объявить объект Viewer
Translator TranslatorClass                 !объявить объект Translator
CODE
Translator.Init                             !инициализировать объект Translator
ViewerActive=Viewer.Init( AsciiFile, |      ! имя файла,
A1:line, |                                 ! поле для вывода
Filename, |                                ! переменная для атрибута NAME
?AsciiBox, |                                ! номер объекта типа LIST
GlobalErrors, |                             ! объект класса ErrorClass
EnableSearch+EnablePrint)
IF ~ViewerActive THEN RETURN. !если инициализация не успешна, то не обращаться
                                           ! к другим методам объекта Viewer
Viewer.SetTranslator(Translator)           !установить перекодирование текста
!программный код
```

TakeEvent (обработать событие в цикле ACCEPT)

TakeEvent(событие **)**

TakeEvent	обрабатывает событие в цикле ACCEPT.
событие	Целочисленная константа, переменная , выражение или мнемоническая метка соответствия, содержащая код события, которое следует обрабатывать в цикле ACCEPT.

Метод **TakeEvent** устанавливает после объекта класса ASCIIViewerClass и возвращает значение, показывающее требуется ли переход на нечало цикла оператором CYCLE для правильного обновления окна.

Реализация: метод TakeEvent управляет изменением размеров, событиями RIGHT-CLICKS (щелчок правой кнопкой), LEFT-CLICKS (щелчок левой кнопкой) и событиями прокрутки списка.

Возвращаемое значение ноль (0) означает ненужность оператора CYCLE, а любое другое значение подразумевает необходимость использования этого оператора.

Тип возвращаемого значения: BYTE

Пример:

```
ACCEPT
CASE FIELD()
OF ?AsciiBox
IF ViewerActive
IF Viewer.TakeEvent(EVENT())
CYCLE
END
END
END
END
```

Свойства класса AsciiPrintClass

FileMgr (объект класса AsciiFileClass)

FileMgr &AsciiFileClass, PROTECTED

Свойство **FileMgr** представляет собой указатель на объект класса AsciiFileClass, управляющий процессом печати. Объект класса AsciiPrintClass использует объект FileMgr для чтения файла, при управлении печатью диапазона строк и при обработке ошибочных ситуаций и выдаче сообщений.

Это свойство имеет атрибут PROTECTED, поэтому на него могут ссылаться только методы класса ASCIIPrintClass или производных от него классов.

Реализация: свойство FileMgr инициализируется в методе Init.

Смотри также: Init

PrintPreview (переключатель предварительного просмотра)

PrintPreview BYTE

Свойство **PrintPreview** содержит используемую по умолчанию установку предварительного просмотра печатаемой страницы для объекта класса AsciiPrintClass. Единица (1) изначально включает кнопку "print preview" (просмотр предполагается по умолчанию), значение ноль (0) "очищает" отметку в этой кнопке (по умолчанию не просматривается страница)

Реализация: Метод Init устанавливает значение свойства PrintPreview в ложь.
Смотри также: Init

Translator (Объект класса TranslatorClass)

Translator &TranslatorClass, PROTECTED

Свойство **Translator** представляет собой указатель на объект класса TranslatorClass для данного объекта класса AsciiPrintClass. Объект класса AsciiPrintClass использует это свойство для перекодировки текста в окне на соответствующий язык. Это свойство имеет атрибут PROTECTED, поэтому его могут использовать методы данного класса AsciiPrintClass или производных от него классов.

Реализация: Класс AsciiPrintClass не инициализирует свойство Translator. Он только обращается к объекту Translator, если указатель на него не пустой. Для установки значения свойства Translator можно воспользоваться методом AsciiViewerClass.SetTranslator

Смотри также: AsciiViewerClass.SetTranslator

Методы класса AsciiPrintClass

Ask (запрос параметров печати)

Ask, VIRTUAL

Метод **Ask** запрашивает конечного пользователя относительно параметров печати.

Он имеет атрибут VIRTUAL, поэтому другие метды базового класса могут напрямую обращаться к виртуальному методу Ask в производном классе. Это

позволяет легко реализовать свою собственную версию данного метода.

Реализация: Метод Ask запрашивает конечного пользователя относительно параметров печати (включенных в стандартное диалоговое окно Print Setup), и дополнительно к этому запрашивает диапазон печатаемых строк. Если пользователь щелкает на кнопке Print, то этот метод выводит на печать требуемые строки на указанные пользователем принтер.

Пример:

```
ACCEPT
CASE FIELD()
OF ?PrintButton !кнопка “Print”
IF EVENT() = EVENT:Accepted                    !вызвать метод Printer.Ask
IF ViewerActive THEN Viewer.Printer.Ask.       !чтобы собрать спецификации и напечатать
строки.
END
END
END
```

Init (инициализировать объект класса ASCIIPrintClass)

Init(ASCIIFileMgr **), VIRTUAL**

Init	инициализирует объект класса ASCIIPrintClass
ASCIIFileMgr	имя объекта класса ASCIIFileClass, управляющего печатью данного файла. Объект класса ASCIIPrintClass использует *ASCIIFileMgr* для чтения данных из файла и обработки нумерации строк и ошибочных ситуаций.

Метод **Init** инициализирует объект класса ASCIIPrintClass. Он имеет атрибут VIRTUAL, поэтому другие метды базового класса могут напрямую обращаться к виртуальному методу Init в производном классе. Это позволяет легко реализовать свою собственную версию данного метода.

Пример:

```
MyViewerClass.Init FUNCTION(FILE AsciiFile,*STRING FileLine,*STRING Filename,|
UNSIGNED ListBox,ErrorClass ErrHandler,BYTE Enables)
CODE
!программный код
IF BAND(Enables,EnableSearch)                  !если установлен флаг Search
SELF.Searcher &= NEW AsciiSearchClass          !экземпляр объекта Searcher
SELF.Searcher.Init(SELF)                       !initialize Searcher
END
```

```
IF BAND(Enables,EnablePrint) !если установлен флаг Print
SELF.Printer &= NEW AsciiPrintClass !активизировать объект Printer
SELF.Printer.Init(SELF) !инициализировать объект Printer
END
```

PrintLines (печать заданных строк)

PrintLines(первая, последняя **), VIRTUAL**

PrintLines	печатает заданные строки.
первая	целочисленная константа, переменная, EQUATE или выражение, содержащее номер первой строки из диапазона печатаемых строк.
последняя	целочисленная константа, переменная, EQUATE или выражение, содержащее номер последней строки из диапазона печатаемых строк.

Метод **PrintLines** печатает заданные строки на активный принтер. Он имеет атрибут VIRTUAL, поэтому другие метды базового класса могут напрямую обращаться к виртуальному методу PrintLines в производном классе. Это позволяет легко реализовать свою собственную версию данного метода.

Пример:

```
IF EVENT() = EVENT:Accepted
IF ACCEPTED() = ?PrintButton
FirstLine=1
LastLine=HighestLine
SELF.PrintLines(FirstLine,LastLine)
POST(EVENT:CloseWindow)
END
END
```

Свойства класса AsciiSearchClass

FileMgr (объект класса AsciiFileClass)

FileMgr &AsciiFileClass, PROTECTED

Свойство **FileMgr** представляет собой ссылку на объект класса AsciiFileClass, который управляет поиском в файле. Объект класса AsciiSearchClass использует FileMgr для чтения файла и обработки ошибочных ситуаций и сообщений.

Это свойство имеет атрибут PROTECTED, поэтому его могут использовать методы данного класса ASCIISearchClass или производных от него классов.

Реализация: Свойство FileMgr инициализирует метод Init

Смотри также: Init

Translator (объект класса TranslatorClass)

Translator &TranslatorClass, PROTECTED

Свойство **Translator** представляет собой указатель на объект класса TranslatorClass для данного объекта класса ASCIISearchClass. Объект класса ASCIISearchClass использует это свойство для перекодировки текста в окне на соответствующий язык. Это свойство имеет атрибут PROTECTED, поэтому его могут использовать методы данного класса ASCIISearchClass или производных от него классов.

Реализация: Класс ASCIISearchClass не инициализирует свойство Translator. Он только обращается к объекту Translator, если указатель на него не пустой. Для установки значения свойства Translator можно воспользоваться методом AsciiViewerClass.SetTranslator

Смотри также: AsciiViewerClass.SetTranslator

Методы класса AsciiSearchClass

Ask (запрос спецификаций поиска)

Ask([нач.строка] **), VIRTUAL**

Ask	запрашивает конечного пользователя относительно контекста поиска, а затем позиционирует текущее положение в файле на первое вхождение контекста.
нач.строка	смещение (или положение) строки, на которой начинается поиск, обычно текущая строка. Если этот параметр опущен, то по умолчанию устанавливается в 1.

Метод **Ask** запрашивает у конечного пользователя контекст поиска, а затем позиционирует текущее положение в файле на первое вхождение контекста или выдает соответствующее сообщение, если искомый контекст не найден.

Метод Ask имеет атрибут VIRTUAL поэтому поэтому другие метды базового класса могут напрямую обращаться к виртуальному методу Ask в производном классе. Это позволяет легко реализовать свою собственную версию данного метода.

Реализация: метод Ask запрашивает у кончного пользователя контекст поиска, направление поиска и является ли поиск чувствительным к регистру букв. Если пользователь не отменяет поиск, метод Ask позиционирует текущее положение в файле на следующую строку, содержащую поисковое значение или выдает соответствующее сообщение, в случае, когда контекст не найден.

Пример:

```
ACCEPT
CASE FIELD()
OF ?PrintButton
IF EVENT() = EVENT:Accepted
IF ViewerActive THEN Viewer.Printer.Ask.
END
OF ?Search !по кнопке “search”
IF EVENT() = EVENT:Accepted
IF ViewerActive !вызвать метод Searcher.Ask
StartSearch=CHOOSE(CHOICE(?AsciiBox)>0, |    ! передавая выделенную в данный
                                             !момент
Viewer.TopLine+CHOICE(?AsciiBox)-1,1)        ! строку в качестве
Viewer.Searcher.Ask(StartSearch)             ! начальной точки поиска
END
END
END
END
```

Init (инициализировать объект класса ASCIISearchClass)

Init(ASCIIFileMgr **), VIRTUAL**

Init	инициализирует объект класса ASCIISearchClass.
ASCIIFileMgr	имя объекта класса ASCIIFileClass, управляющего поиском в данном файле. Объект класса ASCIISearchClass использует *ASCIIFileMgr* для чтения данных из файла.

Метод **Init** инициализирует объект класса ASCIISearchClass. Он имеет атрибут VIRTUAL, поэтому другие метды базового класса могут напрямую обращаться к виртуальному методу Init в производном классе. Это позволяет легко реализовать свою собственную версию данного метода.

Пример:

```
MyViewerClass.Init FUNCTION(FILE AsciiFile,*STRING FileLine,*STRING Filename,|
UNSIGNED ListBox,ErrorClass ErrHandler,BYTE Enables)
CODE
!program code
IF BAND(Enables,EnableSearch)              !если установлен флаг Search
SELF.Searcher &= NEW AsciiSearchClass      !активизировать объект Searcher
SELF.Searcher.Init(SELF)                   !инициализировать объект Searcher
END
IF BAND(Enables,EnablePrint)               !если установлен флаг Print
SELF.Printer &= NEW AsciiPrintClass        ! активизировать объект Printer
SELF.Printer.Init(SELF)                    ! инициализировать объект Printer
END
```

Next (найти следующее вхождение искомого контекста)

Next, VIRTUAL

Метод **Next** возвращает номер строки, содержащей следующее вхождение искомого контекста, заданного в методе Ask.

Он имеет атрибут VIRTUAL, поэтому другие метды базового класса могут напрямую обращаться к виртуальному методу Next в производном классе. Это позволяет легко реализовать свою собственную версию данного метода.

Реализация: Метод Ask обращется к методу Next. Метод Next выполняет поиск контекста в направлении, заданном методом Ask. Для установки ограничений поиска, можно использовать метод Setup.

Тип возвращаемого значения: LONG

Пример:

```
MyAsciiSearchClass.Ask PROCEDURE
CODE
!программный код
CASE EVENT()
OF EVENT:Accepted
CASE FIELD()
OF ?NextButton
SELF.LineCounter=SELF.Next()
IF SELF.LineCounter
SELF.FileMgr.SetLine(SELF.LineCounter)
```

```
END
!программный код
```

Смотри также: Ask, Setup

Setup (установить ограничения поиска)

Setup(ограничения [, нач.строка] **)**

Setup	устанавливает ограничения поиска.
ограничения	Имя структуры, содержащей ограничения поиска. Эта структура должна совпадать по полям со структурой FindGroup GROUP, объявленной в файле ABASCII.INC.
нач.строка	смещение (положение) строки, с которой начинается поиск, обычно это текущая строка. Если этот параметр опущен, то по умолчанию принимается 1.

Метод **Setup** устанавливает ограничения поиска. При поиске в текстовом файле объект AsciiSearchClass применяет эти ограничения.

Реализация: В библиотеке ABC нет обращений к этому методу. Он сущетвтует для того, чтобы вы могли реализовать пользовательский вариант поиска, расширяющий возможности обычного процесса AsciiViewerClass (без использования метода Ask)

Метод Next применяет поисковые ограничения, установленные методом Setup. Эти ограничения включают в себя искомый контекст, направление поиска, и является или нет поиск зависящим от регистра букв.

Объявленная в файле ABASCII.INC группа FindGroup GROUP такова:

```
FindGroup GROUP,TYPE
What PSTRING(64) !искомый текст
MatchCase BYTE !зависит от регистра букв?
Direction STRING(4) !вверх ‘Up ‘ или вниз ‘Down’
END
```

Пример:

```
MyAsciiSearchClass.Ask PROCEDURE
Constraints LIKE(FindGroup)
CODE
Constraints.MatchCase = False       ! всегда независит от регисста букв
Constraints.Direction = ‘Down’      !всегда вниз
                                    !запросить пользователя на предмет
                                    !поискового контекста.
```

```
SELF.Setup(Constraints,StartLine)        !установить ограничения
SELF.LineCounter=SELF.Next() !выполнить поиск
IF SELF.LineCounter
SELF.FileMgr.SetLine(SELF.LineCounter) !встать на следующее вхождение контекста
ELSE
MESSAGE(''''&CLIP(SELF.Constraints.What)&''' not found.')
END
```

Смотри также: Ask, Next

Класс Select File

Обзор

Концепции SelectFileClass

Объект SelectFileClass управляет файловым диалогом Windows для выбора одного или нескольких файлов, как в 16-разрядной, так и в 32- разрядной версиях (с длинными именами файлов).

Отношение к другим классам Application Builder

Класс AsciiViewerClass использует SelectFileClass, чтобы обеспечить выбор просматриваемого файла конечным пользователем. С другой стороны, класс SelectFileClass полностью независим от других классов Application Builder.

Реализация в шаблонах ABC

Шаблоны ABC объявляют локальный класс SelectFileClass *и* объект для каждого экземпляра диалогового шаблона SelectFile.

Класс получает название *procedure*:SelectFile#, где *procedure* – это имя процедуры, а # - номер экземпляра диалогового шаблона. Шаблоны обеспечивают производные классы, поэтому, используя закладку **Classes**, можно легко модифицировать поведение каждого экземпляра диалогового шаблона SelectFile.

Объект получает название SelectFile#, где # - номер экземпляра диалогового шаблона.

Исходные файлы SelectFileClass

Исходные тексты SelectFileClass стандартно устанавливаются в каталог \CLARION4\LIBSRC. Специфические файлы SelectFileClass и их соответствующие компоненты:

ABUTIL.INC	объявления SelectFileClass
ABUTIL.CLW	определения методов SelectFileClass
ABUTIL.TRN	стандартные тексты, маски, переключатели SelectFileClass

Концептуальный пример

Следующий пример показывает типичную последовательность операторов для объявления, конкретизации, инициализации, использования и завершения объекта SelectFileClass. Пример выводит диалог, который позволяет выбрать один или несколько файлов.

```
PROGRAM
INCLUDE('ABUTIL.INC')                    !объявитьSelectFileClass
MAP
END
SelectFile  SelectFileClass              !объявитьSelectFile объект
FileQ       SelectFileQueue              !объявить FileName QUEUE
FileQCount  USHORT,AUTO                  !объявить Q счетчик
FileNames CSTRING(255)                   !переменная для сохранения имени файла
FileMask    CSTRING('Text *.txt|*.txt|All =*.*|*.*')
MultiFiles  BYTE
GetFile     WINDOW('Select File'),AT(,,173,40),SYSTEM,GRAY,RESIZE
ENTRY(@s254),AT(6,6,144,12),USE(FileNames)
BUTTON('...'),AT(156,6,12,12),USE(?SelectFiles)
OPTION,AT(6,20,),USE(MultiFiles)
RADIO('One File'),AT(5,25),USE(?1File),VALUE('0')
RADIO('Multiple Files'),AT(45,25),USE(?MultiFile),VALUE('1')
END
BUTTON('Close'),AT(119,24),USE(?Close)
            END
            CODE
            OPEN(GetFile)
            ACCEPT
  IF EVENT() = EVENT:OpenWindow
SelectFile.Init
SelectFile.AddMask('Clarion source|*.clw;*.inc')
SelectFile.AddMask(FileMask)
END
CASE FIELD()
OF ?SelectFiles
IF EVENT() = EVENT:Accepted
IF MultiFiles
SelectFile.WindowTitle='Select multiple files'
SelectFile.Ask(FileQ,0)
LOOP FileQCount=1 TO RECORDS(FileQ)
GET(FileQ,FileQCount)
```

```
MESSAGE(FileQ.Name)
END
ELSE
SelectFile.WindowTitle = 'Select one file'
FileNames = SelectFile.Ask(1)
DISPLAY(?FileNames)
END
END
OF ?Close
    IF EVENT() = EVENT:Accepted
     POST(Event:CloseWindow)
              END
              END
              END
```

Свойства SelectFileClass

DefaultDirectory (первоначальный путь)

DefaultDirectory	CSTRING(File:MaxFilePath)

Свойство **DefaultDirectory** содержит каталог в котором первоначально открывается файловый диалог Windows. Если DefaultDirectory пусто, файловый диалог открывается в текущем каталоге.

DefaultFile (первоначальное имя/маска файла)

DefaultFile	CSTRING(File:MaxFilePath)

Свойство **DefaultFile** содержит имя файла, которое первоначально появляется в поле имени файла файлового диалога Windows. Имя файла может содержать символы расширения, такие как *, для фильтрации списка файлов в файловом диалоге.

Flags (поведение файлового диалога)

Flags	BYTE

Свойство **Flags** является битовой маской, которая определяет характер действий с файлами, которые выполняет файловый диалог Windows (выбор, выбор множества, сохранение, захват каталога, подавление ошибок). Свойство Flags действует идентично параметру *flag* в FILEDIALOG. См. дополнительные сведения о *FILEDIALOG* в *Language Reference*.

Реализация: Метод Init устанавливает в свойстве Flags его стандартное значение, объявленное в ABUTIL.TRN – выбор файла из любого каталога.

См. также: Init

WindowTitle (текст заголовока файлового диалога)

WindowTitleCSTRING(80)

Свойство **WindowTitle** содержит строку, которая формирует текст заголовка окна файлового диалога Windows.

Реализация: Метод Init устанавливает в свойстве WindowTitle его стандартное значение, объявленное в ABUTIL.TRN. SelectFileClass использует свойство WindowTitle в качестве параметра *title* в функции FILEDIALOG. См. дополнительные сведения о *FILEDIALOG* в *Language Reference*.

См. также: Init

Методы SelectFileClass

AddMask (добавить маски файлов к файловому диалогу)

AddMask(| *description, masks* |**)**
| *mask string* |

AddMask	Добавляет маски к выпадающему списку **List Files of Type (Тип файлов:)** файлового диалога.
description	Строковая константа, переменная, EQUATE или выражение, которое содержит описание маски файлов, как например 'Все файлы - *.*' или 'Исходные тексты - .inc;.clw'. Значение маски может быть включено в описание исключительно для сведения.
masks	Строковая константа, переменная, EQUATE или выражение, которое содержит маски, соответствующие *description*, как например '*.*' или '.inc;.clw'. Несколько масок разделяются точкой с запятой (;).
mask string	Строковая константа, переменная, EQUATE или выражение, которое содержит как маски, так и их описания.

Метод **AddMask** добавляет маски и их описания к выпадающему списку **List Files of Type (Тип файлов:)** файлового диалога. Первая маска определяет стандартное значение для файлового диалога.

Метод AddMask добавляет маски файлов и их описания.

Параметр *mask string* должен содержать одно или более описаний, за которыми следуют соответствующие им файловые маски в виде описание|маски|описание|маски. Элементы строки разделяются вертикальной чертой (|). Например 'Все файлы - *.*|*.*|Исходные тексты Clarion - .inc;.clw|.inc;.clw' определяет два варианта в выпадающем списке **List Files of Type (Тип файлов:)** файлового диалога. См. дополнительные сведения о параметре *extensions* функции FILEDIALOG в *Language Reference*.

Пример:

```
FileMask   CSTRING('Text *.txt|*.txt|All *.*|*.*')
           CODE
  IF EVENT() =3D EVENT:OpenWindow
SelectFile.Init
           SelectFile.SetMask('Clarion source','*.clw;*.inc')
           SelectFile.AddMask(FileMask)
           END
```

См. также: SetMask

Ask (вывести файловый диалог Windows)

Ask([*file queue*] [, *restore path*] **)**

Ask	Выводит файловый диалог Windows.
file queue	Метка структуры QUEUE, которая долучает данные о выбранных файлах. Стуктура должна быть такой же, как и объявленная в ABUTIL.INC структура SelectFileQueue. Если параметр опущен, конечный пользователь может выбрать только один файл, для которого метод Ask возвращает полный путь.
restore path	Строковая константа, переменная, EQUATE или выражение, которое определяет нужно ли восстановить текущий каталог в состояние предшествовавшее файловому диалогу. Значение один (1) в *restore path* восстанавливает текущий путь, значение ноль (0) - не восстанавливает текущий путь. Если опущено, для *restore path* используется стандартное значение ноль (0).

Метод **Ask** выводит файловый диалог Windows и возвращает информацию – в основном полный путь выбранного файла или файлов.

Реализация: Параметр *file queue* должет именовать QUEUE, структура которой начинается также, как и у структуры SelectFileQueue из ABUTIL.INC:

```
SelectFileQueue  QUEUE,TYPE
Name
STRING(File:MaxFilePath)
```

```
ShortName
STRING(File:MaxFilePath)
        END
```

Тип возвращаемых данных: STRING

Пример:

```
FileQ       SelectFileQueue
FileQCount  BYTE
        CODE
        !код программы
        SelectFile.Ask(FileQ,0)
        LOOP FileQCount=1 TO RECORDS(FileQ)
        GET(FileQ,FileQCount)
        MESSAGE(FileQ.Name)
        END
        FileNames = SelectFile.Ask(1)
```

Init (инициализировать объект SelectFileClass)

Init

Метод **Init** инициализирует объект SelectFileClass.

Реализация: Метод Init устанавливает свойства WindowTitle и Flags в их стандартные значения, заданные в ABUTIL.TRN.

Пример:

```
IF EVENT() = EVENT:OpenWindow
        SelectFile.Init
        SelectFile.AddMask('Clarion source|*.clw;*.inc')
        SelectFile.AddMask(FileMask)
        END
```

См. также: Flags, WindowTitle

SetMask (задать маски файлов для файлового диалога)

SetMask(| *description, masks* | **)**
| *mask string* |

SetMask Задает маски для выпадающего списка **List Files of Type (Тип файлов:)** файлового диалога.

description Строковая константа, переменная, EQUATE или выражение, которое содержит описание маски файлов, как например 'Все файлы - *.*' или 'Исходные тексты - .inc;.clw'. Значение маски может быть включено в описание исключительно для сведения.

masks	Строковая константа, переменная, EQUATE или выражение, которое содержит маски, соответствующие *description*, как например '*.*' или '.inc;.clw'. Несколько масок разделяются точкой с запятой (;).
mask string	Строковая константа, переменная, EQUATE или выражение, которое содержит как маски, так и их описания.

Метод **SetMask** задает маски и их описания для выпадающего списа **List Files of Type (Тип файлов:)** файлового диалога. Первая маска определяет стандартное значение для файлового диалога.

Метод SetMask заменяет маски файлов и их описания.

Параметр *mask string* должен содержать одно или более описаний, за которыми следуют соответствующие им файловые маски в виде описание|маски|описание|маски. Элементы строки разделяются вертикальной чертой (|). Например 'Все файлы - *.*|*.*|Исходные тексты Clarion - .inc;.clw|.inc;.clw' определяет два варианта в выпадающем списку **List Files of Type (Тип файлов:)** файлового диалога. См. дополнительные сведения о параметре *extensions* функции FILEDIALOG в *Language Reference*.

Пример:

```
FileMask  CSTRING('Text *.txt|*.txt|All *.*|*.*')
          CODE
          !программный код
          IF EVENT() = EVENT:OpenWindow
          SelectFile.Init
          SelectFile.SetMask('Clarion source','*.clw;*.inc')
          SelectFile.AddMask(FileMask)'
                                    END
```

См. также: AddMask

Класс INI

Обзор

Концепция INI-класса

Объект INI-класса для данной конфигурации обрабатывается централизованно (.INI файл). По принятому соглашению INI-файл является текстовым файлом в ASCII-кодах, который загружает информацию между сеансами работы и содержит данные о конфигурации:

```
[SECTION1]
ENTRY1=value
ENTRYn=value
[SECTIONn]
ENTRY1=value
ENTRYn=value
```

INI-класс автоматически создает INI-файл, а также разделы и строки данных внутри него. INI-класс также корректирует и удаляет разделы и строки данных. В частности, INI-класс значительно упрощает процедуру сохранения и восстановления размеров окна и его положения на экране в промежутке между сеансами работы; вдобавок ко всему, он обеспечивает хранение всех данных по настройке в одном месте, так что Вам придется разместить INI-файл только в одном месте.

Связь с другими Классами Разработки Приложений

PopupClass опционально использует INI-класс, в то же время он сам является полностью независимым относительно других Классов Разработки Приложений.

Реализация ABC шаблона

ABC Шаблоны генерируют код для обработки глобального объекта INI-класса, именуемого INIMgr. Если Вы установите опцию **Использовать INI-файл для сохранения и восстановления программных установок** в разделе **Глобальных свойств**, то впоследствии каждая процедура, основанная на шаблоне оконной процедуры (Frame, Browse и Form) будет обращаться к INIMgr для сохранения и восстановления положения и размера своего окна.

Исходные файлы INI-класса

Исходный код INI-класса по умолчанию находится в \CLARION4\LIBSRC. Ниже представлены собственно файлы, относящиеся к INI-классу:

ABUTIL.INC	объявление INI-класса
ABUTIL.CLW	определения метода INI-класса

Поясняющий Пример

Представленный ниже пример показывает типовую последовательность операторов для объявления, присвоения значения, инициализации, использования и завершения объекта INI-класса.

```
PROGRAM

INCLUDE('ABUTIL.INC')                  !объявление INI-класса
MAP
END
INIMgr     INI-класс                   ! объявление INIMgr объекта
Sound      STRING('ON ')               ! пользовательский выбор звука
Volume     BYTE(3)                     !пользовательский выбор громкости
PWindow    WINDOW('Preferences'),AT(,,89,34),MAX,RESIZE
CHECK('&Sound'),AT(8,6),USE(Sound),VALUE('ON','OFF')
PROMPT('&Volume'),AT(31,19),USE(?VolumePrompt)
SPIN(@s20),AT(8,20,21,7),USE(Volume),HVSCROLL,RANGE(0,9),STEP(1)
BUTTON('OK'),AT(57,3,30,10),USE(?OK)
END
CODE
INIMgr.Init('.\MyApp.INI')             !инициализация INIMgr объекта
INIMgr.Fetch('Preferences','Sound',Sound)
                                       !получить значение звука, по умолчанию 'ON'
Volume=INIMgr.TryFetch('Preferences','Volume')
                                       ! получить значение громкости, нет значения
                                       !по умолчанию
IF Volume  Sound=INIMgr.FetchField('Preferences','Sound&Vol',1)
                                       !получить отделенное запятой значение звука
Volume=INIMgr.FetchField('Preferences','Sound&Vol',2)
                                       !получить отделенное запятой значение
                                       !громкости
END
OPEN(PWindow)
INIMgr.Fetch('Preferences',PWindow)    !восстановить размеры и положение окна
          ACCEPT
          IF EVENT() = EVENT:Accepted
```

```
IF FIELD() = ?OK
INIMgr.Update('Preferences','Sound',Sound)          !сохранить звук
INIMgr.Update('Preferences','Volume',Volume)        !сохранить громкость
INIMgr.Update('Preferences','Sound&Vol',|
                                    !сохранить разделенные запятой значения

CLIP(Sound)&','&Volume)             !например., Sound&Vol=ON,3
POST(EVENT:CloseWindow)
        END
        END
        END
INIMgr.Update('Preferences',PWindow)    !сохранение размеров и положения окна
```

СвойстваINI-класса

FileName

FileName CSTRING(File:MaxFilePath)

Свойство **FileName** содержит имя управляемого INI-файла. Методы INI-класса используют свойство **FileName** для идентификации INI-файла.

Если задан полный путь, то INI-класс пытается найти файл по указанному пути. Если же путь не указан, INI-класс ищет файл в директории Windows. Если не задано вообще никакого имени (''), то INI-класс использует файл WIN.INI.

Например:

Свойство FileName	**Полученный INI-файл**
''	c:\Windows\WIN.INI
'invoice.cfg'	c:\Windows\invoice.cfg
'.\invoice.cfg'	*текущая директория*\invoice.cfg
'c:\invoice\invoice.cfg'	c:\invoice\invoice.cfg

Метод Init устанавливает содержание свойства **FileName**.

Реализация: Методы INI-класса используют свойство **FileName** как параметр файла в операторах GETINI и PUTINI. За дополнительной информацией обращайтесь к книге *Описание Языка*.

Смотри Также: Init

Методы INI-класса

Fetch (получить запись из INI-файла)

Fetch(*раздел*,|*запись* [, *значение*]|**)**
|*окно* |

Fetch	Получает или возвращает значение из INI-файла.
раздел	Константа типа string, переменная, метка соответствия или выражение, содержащее наименование раздела INI-файла.
запись	Константа типа string, переменная, метка соответствия или выражение, содержащее наименование записи INI-файла.
значение	Метка переменной, содержащей значение, полученное по умолчанию и получающей реальное полученное значение. Если пропущено, то в INI-файле должен быть соответствующий раздел и запись, которую смог бы возврвтить метод Fetch.
окно	Метка соответствующего WINDOW или APPLICATION, у которого следует восстановить его прежний размер и положение. Если этот параметр определен, метод Fetch не возвращает значение, а восстанавливает размер и положение соответствующего окна.

Метод **Fetch** Получает или возвращает значение из INI-файла.

Fetch(*раздел,запись*[,*значение*])
Возвращает единственное значение, определенное для конкретной записи в конкретном разделе. Если определен параметр *значение*, то метод Fetch обновит его и не возвратит ничего. Если же нет, то метод Fetch возвратит запрошенную величину.

Fetch(*раздел,окно*)
Восстанавливает различные атрибуты окна, сохраненные предшествующим вызовом для коррекции Update(раздел,окно). Восстановление значений атрибутов приводит к тому, что окно принимает свое исходное положение и размер.

Реализация: Если имеется окно, метод Fetch получает пять записей с данными из соответствующего раздела INI-файла: Данные о максимизации, Положение X, Положение Y, Высота, Ширина. Затем он, в соответствии с извлеченными величинами, изменяет параметры соответствующего окна или приложения.

Возвращаемый тип данных: STRING

Пример:

```
Sound      STRING('ON ')
PWindow    WINDOW('Preferences'),AT(,,89,34),IMM,MAX,RESIZE
CHECK('&Sound'),AT(8,6),USE(Sound),VALUE('ON','OFF')
BUTTON('OK'),AT(57,3,30,10),USE(?OK)
           END
           CODE
INIMgr.Fetch('Preferences','Sound',Sound)        !получить 'Sound', по умолчанию ON
Sound=INIMgr.Fetch('Preferences','Sound')        !возвратить 'Sound', нет значения
                                                 !по умолчанию
           OPEN(PWindow)
           INIMgr.Fetch('Preferences',PWindow)   !восстановить размеры и положение
                                                 !окна PWindow
```

Смотри также: Update

FetchField (возвратить разделенные запятыми значения из INI-файла)

FetchField(*раздел, запись, поле* **)**

FetchField Возвратить отделенное запятой значение из INI-файла.

раздел Константа типа string, переменная, метка соответствия или выражение, содержащее имя раздела INI-файла.

запись Константа типа string, переменная, метка соответствия или выражение, содержащее имя записи INI-файла.

поле Константа целого типа, переменная, метка соответствия или выражение, определяющее то отделенное запятой значение, которое надо вернуть из INI-файла.

FetchField метод возвращает одно из нескольких отделенных друг от друга запятой значений получаемой величины из INI-файла.

Например:

```
[MySection]
MyEntry=M,35,Blue,Brown,160
```

Если значение поля равно единице (1), то возвращается значение, находящееся до первой запятой в строке; при значении два (2) возвращается значение, находящееся между первой и второй запятыми; при трех (3) - между вторыми и третьими запятыми, и т.п.

Возвращаемый тип данных: STRING

Пример:

```
Sound     STRING('ON ')
Volume    BYTE(3)
 CODE
INIMgr.Update('Preferences','Sound&Volume',    |!создает запись в INI-файле
CLIP(Sound)&','&Volume)                        !со значениямиSound&Volume=ON,3
                                               !program code
Sound=INIMgr.FetchField('Preferences','Sound&Volume',1)
                                               !получить первое значение - 'ON'
Volume=INIMgr.FetchField('Preferences','Sound&Volume',2)
                                               ! получить второе значение - 3
```

FetchQueue (получить ряд записей из INI-файла)

FetchQueue(*раздел, запись, очередь, поле [,поле] [,поле]* **)**

FetchQueue	Добавляет ряд величин из INI-файла в очередь.
раздел	Константа типа string, переменная, метка соответствия или выражение, содержащее имя раздела INI-файла.
запись	Константа типа string, переменная, метка соответствия или выражение, содержащее имя записи INI-файла.
очередь	Имя предназначенной для записи значений очереди.
поле	Имя поля в очереди, в которое записывается значение. В очереди должно присутствовать как минимум одно поле, максимальное же количество полей – три.

FetchQueue метод добавляет целый ряд величин из INI-файла в соответствующие поля выбранной очереди.

Реализация: Метод **FetchQueue** получает ряд значений из следующей последовательности:

```
[section]
entry=ItemsInQueue
entry_n=value,optionalvalue,optionalvalue
```

Например:

```
[Users]
User=3
User_1=Fred,1
User_2=Barney,0
User_3=Wilma,1
```

Пример:

```
UserQ     QUEUE
Name      STRING(20)
```

```
Auth        BYTE
END
CODE
INIMgr.FetchQueue('Users','User',UserQ,UserQ.Name,UserQ.Auth)
                                                    !получить UserQ
                                                    !program code
INIMgr.Update('Users','User',RECORDS(UserQ))
                                                    !установить счетчик UserQ LOOP i# =
                                                    !1 TO RECORDS(UserQ)
                                                    !разместить UserQ строку
GET(UserQ,i#)
INIMgr.Update('Users','User_'&i#,CLIP(UserQ.Name)&','&UserQ.Auth)
        END
```

Init (инициализировать объект INI-класса)

Init(*имя файла***)**

Init Инициализирует объект **INI-класса**.

имя файла Константа целого типа, переменная, метка соответствия или выражение, содержащее имя INI-файла. Если имя INI-файла включает в себя полный путь, то INI-класс ищет файл в заданной директории. Если путь не задан, то INI-класс ищет INI-файл в директории Windows. Если имя INI-файла не задано вовсе (задано как ''), то INI-класс обращается к файлу WIN.INI.

Метод **Init** инициализирует объект INI-класса.

Реализация: Метод **Init** присваивает свойству **FileName** *имя файла*.

Пример:

```
INCLUDE('UTILITY.INC')
INIMgr          INI-класс
                CODE
INIMgr.Init('c:\MyApp\MyApp.INI')       !читать & писать из/в файл
c:\MyApp\MyApp.INI
                INIMgr.Init('.\MyApp.INI')  ! читать & писать из/в файл
                                            !текущая_директория\MyApp.INI
                INIMgr.Init('')             ! читать & писать из/в файл c:\Windows\WIN.INI
                INIMgr.Init('MyApp.INI')                                              !
                                            !читать & писать из/в файл Windows\MyApp.INI
```

Смотри также: **FileName**

TryFetch (получить значение из INI-файла)

TryFetch(*раздел, запись* **)**

TryFetch Возвратить значение из INI-файла.
раздел Константа типа string, переменная, метка соответствия или выражение, содержащее имя раздела INI-файла.
запись Константа типа string, переменная, метка соответствия или выражение, содержащее имя записи INI-файла.

Метод **TryFetch** возвращает значение из INI-файла. Если заданный раздел и запись не существуют, **TryFetch** возвращает пустую строку. Это позволяет проверять возвращаемое значение и производить соответствующие действия, в случае отсутствия строки в INI-файле.

Возвращаемый тип данных: STRING

Пример:

```
Color        BYTE
DefaultColor  EQUATE(5)
             CODE
             Color=INIMgr.TryFetch('Preferences','Color')
                                          !вернуть 'Color', нет значения по умолчанию
             IF NOT Color
             Color=DefaultColor
             END
```

TryFetchField (возвратить отделенное запятой значение из INI-файла)

TryFetchField(*раздел, запись, поле* **)**

TryFetchField Возвращает отделенное запятой значение из INI-файла.
раздел Константа типа string, переменная, метка соответствия или выражение, содержащее имя раздела INI-файла.
запись Константа типа string, переменная, метка соответствия или выражение, содержащее имя записи INI-файла.
поле Константа целого типа, переменная, метка соответствия или выражение, определяющее то отделенное запятой значение, которое надо вернуть из INI-файла.

TryFetchField метод возвращает одно из нескольких отделенных друг от друга запятой значений получаемой величины из INI-файла. Если заданный раздел и запись не существуют, **TryFetchField** возвращает пустую строку. Это позволяет проверять возвращаемое значение и производить соответствующие действия в случае отсутствия строки в INI-файле.

TryFetchField принимает значение только одной из отделенных запятой значений из следующего ряда V1,V2,...,Vn. Если значение поля равно единице (1), то возвращается значение, находящееся до первой запятой в строке; при значении два (2) возвращается значение, находящееся между первой и второй запятыми; при трех (3) - между вторыми и третьими запятыми, и т.п.

Возвращаемый тип данных: STRING

Пример:

```
Sound                   STRING(3)
Volume          BYTE
 CODE
  Sound=INIMgr.TryFetchField('Preferences','Sound&Volume',1)
!получить значение Sound
  IF NOT Sound

      !если отсутствует
      Sound='ON'

  !по умолчанию «on»
  END
  Volume=INIMgr.TryFetchField('Preferences','Sound&Volume',2)    !
получить значение Volume
  IF NOT Volume

  ! если отсутствует
      Volume=3

      ! по умолчанию «3»
  END
  !program code
  INIMgr.Update('Preferences','Sound&Volume', |
          !создать в INI-файле запись вида
                                        CLIP(Sound)&','&Volume)

!Sound&Volume=ON,3
```

Update (записать строку в INI-файл)

Update(раздел, |*запись* [, *значение*] | **)**
|*окно* |

Update Записывает данные в INI-файл.

раздел	Константа типа string, переменная, метка соответствия или выражение, содержащее наименование раздела INI-файла.
запись	Константа типа string, переменная, метка соответствия или выражение, содержащее наименование записи INI-файла.
значение	Метка переменной, содержащей значение, полученное по умолчанию и получающей реальное полученное значение. Константа, переменная, метка соответствия или выражение, содержащее значение, предназначенное для сохранения в соответствующем разделе и записи.
окно	Метка соответствующего WINDOW или APPLICATION, параметры размера и положения которого сохраняет метод Update.

Метод **Update** записывает в INI-файл строки данных. Если задано нулевое *значение* (''), то существующая запись удаляется.

Update(*раздел,запись,значение*)
Пишет единственное значение, определенное разделом и записью.

Update(*раздел,окно*)
Записывает в INI-файл атрибуты окна, определяющие его положение и размер, которые могут быть впоследствии получены с помощью метода Fetch(*раздел,окно*). Восстановление величин приводит к возвращению выбранного окна в свое прежнее сохраненное положение и восстановление его размеров.

Реализация: Если имеется окно, метод Update записывает пять строк данных в соответствующий *раздел* INI-файла: Данные о максимизации, Положение X, Положение Y, Высота, Ширина. Эти записи извлекаются и применяются методом Fetch для восстановления размеров и положения соответствующего окна.

Пример:

```
Sound     STRING('ON ')
PWindow   WINDOW('Preferences'),AT(,,89,34),IMM,MAX,RESIZE
CHECK('&Sound'),AT(8,6),USE(Sound),VALUE('ON','OFF')
BUTTON('OK'),AT(57,3,30,10),USE(?OK)
          END
          CODE
          OPEN(PWindow)
INIMgr.Fetch('Preferences',PWindow)  !восстановить размер и положение окна
                                     !PWindow size & position
INIMgr.Fetch('Preferences','Sound',Sound)
                                     !получить запись 'Sound'
```

```
                                        !program code
INIMgr.Update('Preferences','Sound',Sound)
                                        !сохранить запись 'Sound'
INIMgr.Update('Preferences',PWindow)
                                        ! сохранить размер и положение окна PWindow
```

Смотри также: Fetch

Класс Popup

Обзор

Концепция Popup-класса

Объект Popup-класса определяет полностью готовое к применению popup-меню (всплывающее меню) и управляет им. При этом он значительно упрощает добавление этого popup-меню в ваши процедуры.

Вы можете назначить различным пунктам popup-меню те же действия, что и соответствующим кнопкам на окне, то есть разрешить выбор какого-то пункта меню (текст меню не затенен) только в том случае, если доступна соответствующая кнопка, тогда при выборе этого пункта меню будет производится действие, аналогичное действию кнопки.

С другой стороны, можно заставить пункт Popup-меню посылать какое-нибудь собственное событие или просто возвращать его ID, так что можно отлавливать событие выбора пункта Popup-меню и писать «под него» свой собственный код (обрабатывать его по собственному желанию).

Объект Popup-класса опционально использует преимущества Translator-класса, так что текст меню легко может быть переведен на другие языки без правки кода вашей программы.

Связь с другими Классами Разработки Приложений

Popup-класс опционально использует Translator-класс для перевода текста меню в процессе работы, а также INI-класс, чтобы сохранить и восстановить параметры меню в файле конфигурации (.INI). Для работы Popup-класса нет необходимости в использовании какого-либо другого класса; однако, если применяются какие-либо другие свойства, необходимо описывать их в вашей программе. Смотри Пояснительный Пример.

ASCIIVIEWER-класс, Browse-класс, и PrintPreview-класс – все они используют Popup-класс для управления их собственными popup-меню. Такое использование Popup-класса происходит автоматически, когда Вы включаете заголовок класса (ABASCII.INC, ABBROWSE.INC или ABPRINT.INC) в секцию данных вашей программы.

Реализация ABC шаблонов

Шаблоны ABC объявляют локальный класс Popup-класса и объект для каждого случая применения шаблона Popup-кода.

Класс называется procedure:PopupMgr#, где procedure - имя процедуры, а # - номер случая применения шаблона Popup- кода. Полученный класс обслуживается шаблонами, так что Вы можете использовать шаблон Popup-кода **Classes** для того, чтобы облегчить себе труд по модификации поведения Popup-меню в зависимости от конкретного случая применения.

Объект же называется PopupMgr#, где # - номер случая применения шаблона Popup- кода.

Файлы-источники Popup-класса

Файлы с исходным кодом Popup-класса установлены по умолчанию в директорию \CLARION4\LIBSRC. Собственно файлы с кодом Popup-класса и другие необходимые компоненты представлены ниже:

ABPOPUP.INC	Объявления Popup-класса
ABPOPUP.CLW	Определения метода Popup-класса
ABPOPUP.TRN	Строки перевода Popup-класса

Поясняющий Пример

Следующий пример показывает типичную последовательность применения операторов, выполняющих действия по объявлению, приписыванию значений, инициализации, использованию и закрытию объекта Popup-класса.

Этот пример демонстрирует диалоговое окно, имеющее свое Popup-меню и появляющееся по правому щелчку мыши. Пункты Popup-меню дублируют действия кнопок, которые выполняют различные действия (с тремя различными методами Popup-класса). Кнопки диалога демонстрируют способность Popup-класса сохранять/восстанавливать меню в/из INI файла.

```
                PROGRAM
                MAP
                END
                INCLUDE('ABPOPUP.INC')          !declare PopupClass
                INCLUDE('ABUTIL.INC')           !declare INIClass & Translator
                INCLUDE('KEYCODES.CLW')         !declare right-click EQUATE
PopupString     STRING(20)                      !to receive menu selection
PopupMgr        PopupClass                      !declare PopupMgr object
Translator      TranslatorClass                 !declare Translator object
```

```
INIMgr      INIClass                    !declare INIMgr object
INIFile     EQUATE('.\Popup.ini')       !declare INI pathname EQUATE
PopupWin    WINDOW('Popup Demo'),AT(,,184,50),ALRT(MouseRight),GRAY
BUTTON('&Save Popup'),AT(17,16),USE(?Save)
BUTTON('&Restore Popup'),AT(74,16),USE(?Restore),DISABLE
BUTTON('Close'),AT(140,16),USE(?Close)
        END
        CODE
        OPEN(PopupWin)  Translator.Init   !initialize Translator object
        INIMgr.Init(INIFile)              !initialize INIMgr object
        PopupMgr.Init(INIMgr)             !initialize PopupMgr object
        PopupMgr.AddItemMimic('Save',?Save)     !Save item mimics ?Save button
        PopupMgr.AddItem('Restore Popup','Restore')       !add menu item: Restore
        PopupMgr.SetItemEnable('Restore',False)  !initially disable Restore item
        PopupMgr.AddItem('-','Separator1')        !add a menu item separator
        PopupMgr.AddItem('Disable Save','Disable')!add a menu item: Disable
        PopupMgr.AddItem('-','Separator2')        !add a menu item separator
        PopupMgr.AddItem('Close (EVENT:Accepted)','Close')     !add menu item: Close
        PopupMgr.AddItemEvent('Close',EVENT:Accepted,?Close)
                                                  !Close POSTs event to a control
        PopupMgr.AddItem('Close (EVENT:CloseWindow)','Close2')
                                                  !add a menu item: Close2
        PopupMgr.AddItemEvent('Close2',EVENT:CloseWindow,0)
                                                  !Close2 POSTs independent event
        PopupMgr.SetTranslator(Translator)        !enable popup text translation
        ACCEPT
        CASE EVENT()
        OF EVENT:AlertKey                         !trap for alerted keys
        IF KEYCODE() = MouseRight                 !if right-click
        PopupString=PopupMgr.Ask()                !display popup menu
        CASE PopupString                          !check for selected item
        OF 'Disable'                              !if Disable item selected
        IF PopupMgr.GetItemChecked('Disable')
        PopupMgr.SetItemCheck('Disable',False)    !toggle the menu check mark
        ENABLE(?Save)                             !toggle ?Save button state
        ELSE                                      !which automatically toggles
        PopupMgr.SetItemCheck('Disable',True)     !the Save menu item, because
        DISABLE(?Save)                            !t mimics the ?Save button
        END
        OF 'Restore'                              !if Restore item selected
        POST(EVENT:Accepted,?Restore)             !code your own functionality
        ELSE                                      !if any other item selected
        END                                       !Ask automatically handled it
```

```
            END
        END
        CASE FIELD()
        OF ?Save                    !Save button mimiced by Save item
        CASE EVENT()
        OF EVENT:Accepted
        PopupMgr.Save('MyPopup')  !save menu definition to INI
        RUN('NotePad '&INIFile)   !display/edit menu definition
        ENABLE(?Restore)          !enable the Restore button
        PopupMgr.SetItemEnable('Restore',True)   !enable the Restore item
            END
        OF ?Restore
        CASE EVENT()
        OF EVENT:Accepted
        PopupMgr.Restore('MyPopup')      !restore/define menu from INI
            END
        OF ?Close                        !Close btn Accepted by Close item
            CASE EVENT()
            OF EVENT:Accepted
            POST(Event:CloseWindow)
            END
        END
    END
    PopupMgr.Kill
```

Свойства Popup-класса

Popup-класс содержит свойства, описанные ниже.

ClearKeycode (очистить «буфер» KEYCODE)

ClearKeycode	BYTE

Свойство ClearKeycode определяет, очищает ли объект Popup-класса значение из "буфера" KEYCODE() перед совершением действия, соответствующего выбранному пункту меню (MouseRight). Значение 1(« Истина») устанавливает "буфер" KEYCODE() в ноль; значение 0(«Ложь») оставляет "буфер" KEYCODE() неизменным. См. KEYCODE и SETKEYCODE в Описании Языка для получения дополнительной информации.

Совет: Неочищенное значение KEYCODE() может привести к тому, что Popup-меню вновь появляться при некотором стечении обстоятельств; поэтому рекомендуется присваивать свойству ClearKeycode значение Истина.

Реализация: Init метод устанавливает свойство ClearKeycode равным нулю. Ask метод выполняет действие, определенное свойством ClearKeycode.

Смотри также: Ask, Init

Методы Popup-класса

Popup-класс содержит методы, перечисленные ниже.

Функциональная Организация - Возможное Применение

Для лучшего понимания работы Popup-класса полезно будет объединить его методы в две больших группы согласно их возможному применению: первичный интерфейс и виртуальные методы. Эта организация отражает то, как видится использование методов Popup-класса в общем случае.

Основные Интерфейсные Методы

Основные интерфейсные методы, которые вы скорее всего будете вызывать из программы, могут быть далее разделены на три категории:

Применяются один раз:

Init	Инициализация объекта Popup-класса
AddMenu	добавить меню
AddItem	добавить пункт меню
AddItemEvent	определить действия для пункта меню
AddItemMimic	связать пункт меню с кнопкой
AddSubMenu	добавить подменю
Kill	закрыть объект Popup-класса

В основном используются в работе:

Ask	показать и обработать popup-меню
GetItemChecked	возвратить статус переключаемого пункта меню
GetItemEnabled	возвратить статус пункта меню
SetItemCheck	установить статус переключаемого пункта меню
SetItemEnable	установить статус пункта меню

Применяются изредка:

DeleteItem	удалить пункт меню
GetLastSelection	получить последний выбранный пункт меню
SetTranslator	установить транслятор
Save	сохранить меню для последующего восстановления
Restore	восстановить меню

Виртуальные Методы

Popup-класс не имеет виртуальных методов.

AddItem (добавить пункт меню)

AddItem(*текст [, имя] [, положение] [, уровень]* **)**

AddItem Добавляет пункт к popup-меню.

текст Константа типа string, переменная, метка соответствия или выражение, содержащее текст пункта меню. Одиночный дефис приводит к созданию невыбираемого пункта меню - сепаратора (трехмерная горизонтальная полоска) на меню.

имя Константа типа string, переменная, метка соответствия или выражение, содержащее Имя пункта меню. Другие методы Popup-класса обращаются к пункту меню по имени, а не по его тексту. Это позволяет Вам применять runtime translation (перевод во время выполнения) или динамическую перестройку меню без изменения кода. Если параметр *имя* опущен, AddItem использует в качестве имени текст.

положение Константа типа string, переменная, метка соответствия или выражение, содержащее *имя* того пункта меню, после которого новый пункт будет добавлен. Если параметр *положение* опущен, AddItem метод добавляет новый пункт в конец popup-меню

уровень Целочисленная константа, переменная, метка соответствия или выражение, содержащее степень вложенности или глубину нового пункта меню. Если *уровень* опущен, то по умолчанию принимается равным единице. Если параметр *положение* опущен или пусто, то *уровень* должен быть или опущен, или равен единице (1).

Метод **AddItem** добавляет пункт к popup-меню.

Каждому пункту меню приводится в соответствие свое действие или с помощью методов AddItemMimic или AddItemEvent, или путем написания собственного программного кода. Эти методы (и написанный вами код) должны ссылаться на пункты меню по имени (а не по тексту).

Пример:

```
PopupMgr.AddItem('Save Popup','Save')          !добавить пункт меню.: Save
PopupMgr.AddItemMimic('Save',?Save)            !Уподобить действие пункта
                                               !действию кнопки ?Save button
```

```
PopupMgr.AddItem('Restore Popup','Restore')    ! добавить пункт меню.: Restore
PopupMgr.AddItem('-','Separator1')             ! добавить разделитель.
PopupMgr.AddItem('Disable Save','Disable')     ! добавить пункт меню.: Disable
PopupMgr.AddItem('-','Separator2')             ! добавить разделитель
PopupMgr.AddItem('Close (control event)','Close')! добавить пункт меню.: Close
PopupMgr.AddItemEvent('Close',EVENT:Accepted,?Close)      ! пункт меню Close
                                        !посылает событие элементу управления
PopupMgr.AddItem('Close (window event)','Close2')      ! добавить пункт меню.:
                                                       !Close2
PopupMgr.AddItemEvent('Close2',EVENT:CloseWindow,0)
        ! пункт меню Close2 посылает независимое событие
```

Смотри также: AddItemEvent, AddItemMimic

AddItemEvent (установить исполняемое действие для пункта меню)

AddItemEvent(*имя, событие* [, *элемент управления*] **), PROC**

AddItemEvent	Связывает событие с пунктом меню.
имя	Константа типа string, переменная, метка соответствия или выражение, содержащее *имя* пункта меню, которое связано с событием. Если указанный раздел не существует, AddItemEvent добавляет его в конец popup-меню.
событие	Целочисленная константа, переменная, метка соответствия или выражение, содержащее номер события, которое будет послано в том случае, когда конечный пользователь выберет этот пункт меню.
элемент управления	Целочисленная константа, переменная, метка соответствия или выражение, содержащее номер элемента управления, которому будет послано событие, когда конечный пользователь выбирает этот пункт меню. Чтобы посылать событие, не зависящее от элемента управления, используйте значение *элемента управления,* равное нулю (0). Если же этот параметр пропущен, номер элемента управления по умолчанию принимается равным нулю (0).

Метод **AddItemEvent** привязывает событие к конкретному пункту меню и возвращает имя пункта. Когда конечный пользователь выбирает пункт меню, объект Popup-класса посылает событие, соответствующему элементу управления.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Метод Ask определяет выбранный пункт меню и посылает *событие*.

Возвращаемый тип данных: STRING

Пример:

```
PopupMgr.AddItem('Close (control event)','Close')        !добавить пункт меню: Close
PopupMgr.AddItemEvent('Close',EVENT:Accepted,?Close)
        !пункт Close посылает событие элементу управления
PopupMgr.AddItem('Close (window event)','Close2')        ! добавить пункт меню: Close2
PopupMgr.AddItemEvent('Close2',EVENT:CloseWindow,0)
        ! пункт Close2 посылает независимое событие
```

Смотри также: AddItem, AddItemMimic, AddMenu, Ask

AddItemMimic (связать пункт меню с кнопкой)

AddItemMimic(*имя, кнопка [, текст]* **), PROC**

AddItemMimic	Связывает пункт меню с кнопкой.
имя	Константа типа string, переменная, метка соответствия или выражение, содержащее *имя* пункта меню, которое связано с кнопкой. Если указанный пункт не существует, AddItemEvent добавляет его в конец popup-меню. Для добавления нового пункта на кнопке должен быть текст, или же у пункта меню дожен быть задан параметр *текст*.
кнопка	Числовая константа, переменная, метка соответствия или выражение, содержащее номер связанного элемента управления (связанной кнопки). Если на кнопке нет никакого текста, необходимо задать параметр *текст*.
текст	Константа типа string, переменная, метка соответствия или выражение, содержащее текст пункта меню. Другие методы Popup-класса связаны с пунктом меню по его имени, а не по тексту. Это позволяет производить трансляцию (runtime translation) во время выполнения или динамическую перестройку меню без правки кода. Если параметр *текст* опущен, AddItemMimic использует в качестве текста пункта меню текст кнопки.

AddItemMimic метод связывает пункт меню с кнопкой и возвращает имя пункта меню. AddItemMimic может добавлять новый пункт меню или добавлять *связь* к существующему пункту. Соответствующий пункту меню текст аналогичен тексту кнопки, при этом он доступен для выбора только тогда, когда доступна для выбора соответствующая кнопка, когда же этот пункт выбран, он вызывает то же действие, которое производится по нажатию кнопки.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Ask метод отлавливает выбранный пункт меню и посылает *кнопке* событие EVENT:Accepted.

Если параметр *кнопка* не соответствует реальному элементу управления, AddItemMimic не производит никаких действий.

Возвращаемый тип данных: STRING

Пример:

```
PopupMgr.AddItem('Save Popup','Save')  ! добавить пункт меню: Save
PopupMgr.AddItemMimic('Save',?Save)    !сохранить пункт меню с действием,
                                       !подобным ?Save button
PopupMgr.AddItemMimic('Insert',?Insert) ! добавить пункт меню Insert с
                                       !действием, подобным кнопке ?Insert
```

Смотри также: AddItem, AddMenu, Ask

AddMenu (добавить меню)

AddMenu(*выборы* [, *позиция*] **)**

AddMenu Добавляет popup-меню.

выборы Константа типа string, переменная, метка соответствия или выражение, содержащее текст из выбранного пункта popup-меню.

позиция Константа типа string, переменная, метка соответствия или выражение, содержащее какой-нибудь из существующих пунктов меню Popup-класса, чтобы добавить новые. Если параметр опущен или равен нулю (0), AddMenu очищает любые существующие пункты меню.

AddMenu метод добавляет все popup-меню или добавляет дополнительные пункты к существующему меню. AddMenu метод создает пункт Popup-меню с уникальным именем для каждого текста, специфицированного параметром *выборы*. Параметр *выборы* идентичен параметру *выборы* для команды POPUP. См. POPUP в *Описании Языка* для получения дополнительной информации.

Вы устанавливаете действие, принятое для каждого пункта меню с помощью методов AddItemMimic или AddItemEvent или используя ваш собственный программный код. Эти методы (и ваш код) должны быть связаны с пунктами меню по имени, а не по тексту.

Реализация: AddMenu метод опционально заменяет любое предварительно определенное меню для данного объекта Popup-класса.

Ask метод отображает popup-меню и возвращает имя выбранного пункта.

Имя пункта меню – это тот же самый его текст минус любые специальные символы. То есть имя содержит только символы 'A-Z', 'a-z', и '0-9'. Если полученное в результате отбрасывания специальных символов имя не уникально, то, чтобы делать его уникальным, Popup-класс добавляет в конец его порядковый номер.

Пример:

```
MenuChoices EQUATE('&Save Menu|&Restore Menu|-|&Close')
                                        !объявить строку описания меню
        CODE
        PopupMgr.AddMenu(MenuChoices)   !добавить Popup-меню
        PopupMgr.AddItemMimic('SaveMenu',?Save)
                                        !уподобить SaveMenu действию кнопки ?Save
        PopupMgr.AddItemEvent('Close',EVENT:Accepted,?Close)
                                        !пунктClose посылает событие
                                        !соответствующей кнопке
        !текст программы
        IF PopupMgr.Ask()= 'RestoreMenu'
                                        !если выбран пункт RestoreMenu
PopupMgr.Restore('MyMenu')
                                        !то задать свою обработку
        ELSE                            !если выбран другой пункт меню
        END                             !он автоматически обрабатывается методом
                                        !Ask
```

Смотри также: AddItemEvent, AddItemMimic, Ask

AddSubMenu (добавить подменю)

AddSubMenu(*[Текст], выборы, пункт-родитель* **)**

AddSubMenu Добавляет к существующему меню подменю.

текст Константа типа string, переменная, метка соответствия, или выражение, содержащее текст подменю. Если параметр опущен, текст подменю должен быть определен в параметре *выборы*.

выборы Константа типа string, переменная, метка соответствия, или выражение, содержащее текст пункта подменю. Пунктам подменю должна предшествовать двойная открытая фигурная скобка ({{), а закрываться они должны одиночной фигурной скобкой (}).

пункт-родитель Константа типа string, переменная, метка соответствия или выражение, содержащее имя меню или имя пункта меню, после которого надо вставить подменю.

AddSubMenu метод добавляет подменю к существующему меню. AddSubMenu метод добавляет подменю и его пункты, включая уникальное имя для каждого пункта, указанного параметром *выборы*. Параметр *выборы* идентичен секции подменю параметра *выборы* для команды POPUP. См. POPUP в Описании Языка для получения дополнительной информации.

Вы устанавливаете действие для каждого пункта меню или с помощью методов AddItemMimic или AddItemEvent, или используя ваш собственный программный код. Эти методы (и ваш код) должны быть связаны с пунктами меню по имени (а не по тексту).

Реализация: Ask метод отображает popup-меню и возвращает имя выбранного раздела.

Имя пункта меню – это тот же самый его текст минус любые специальные символы. То есть имя содержит только символы 'A-Z', 'a-z', и '0-9'. Если полученное в результате отбрасывания специальных символов имя не уникально, то, чтобы делать его уникальным, Popup-класс добавляет в конец его порядковый номер.

Пример:

```
MenuChoices EQUATE('&Insert|&Change|&Delete')
                                        ! Объявить строку определения меню
SubChoices EQUATE('{{by &name|by &ZIP code}')
                                        ! Объявить определение подменю
CODE
PopupMgr.AddMenu(MenuChoices)     ! Добавить Popup-меню
PopupMgr.AddSubMenu('&Print',SubChoices,'Delete')
                                        !добавить подменю Print после удаления
          ! Код программы
CASE PopupMgr.Ask()                     ! Показать popup меню
OF ('Insert')       ;DO Update(1)       ! Обработать выбор конечного пользователя
OF ('Change')     ;DO Update(2)         ! Обработать выбор конечного пользователя
OF ('Delete')       ;DO Update(3)       ! Обработать выбор конечного пользователя
OF ('byname')     ;DO PrintByName       ! Обработать выбор конечного пользователя
OF ('byZIPcode') ;DO PrintByZIP         ! Обработать выбор конечного пользователя
      END
```

Смотри также: AddItemEvent, AddItemMimic, AddMenu, Ask

Ask (отобразить popup-меню)

Ask([*x*] [,*y*]), PROC

Ask Возвращает имя выбранного пункта Popup-меню.

x Целочисленная константа, переменная, метка соответствия, или выражение, которое определяет горизонтальное положение левого верхнего угла меню. Если параметр опущен, левый верхний угол появившегося меню будет находится в текущей позиции курсора.

y Целочисленная константа, переменная, метка соответствия, или выражение, которое определяет вертикальное положение левого верхнего угла меню. Если параметр опущен, левый верхний угол появившегося меню будет находится в текущей позиции курсора.

Ask метод отображает popup-меню, исполняет любое действие, определенное методами AddItemEvent или AddItemMimic для выбранного пункта меню, затем возвращает имя выбранного пункта меню. Методы AddItem, AddItemMimic или AddMenu задают имя пункта меню.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Возвращаемый тип данных: STRING

Пример:

```
MenuChoices EQUATE('&Save Menu|&Restore Menu|-|&Close')
                                    ! Объявить строку определения меню
CODE
PopupMgr.AddMenu(MenuChoices)    ! Добавить Popup-меню
PopupMgr.AddItemMimic('SaveMenu',?Save)
                                    !Действие пункта SaveMenu подобно кнопке
                                    !?Save
PopupMgr.AddItemEvent('Close',EVENT:Accepted,?Close)
                                    !Close посылает событие элементу управления
!Код программы
IF PopupMgr.Ask()= 'RestoreMenu'    !если выбран пункт RestoreMenu
PopupMgr.Restore('MyMenu')          !написать собственную обработку этого
                                    !пункта
ELSE                                !если выбран какой-нибудь другой пункт меню
        END                         !Ask автоматически его обработает
```

Смотри также: AddItem, AddItemMimic, AddMenu

DeleteItem (удалить пункт меню)

DeleteItem(*имя*)

DeleteItem Удалить пункт меню
имя Константа типа string, переменная, метка соответствия или выражение, содержащее *имя* пункта меню. Методы AddItem, AddItemMimic или AddMenu определяют имя пункта меню.

Метод **DeleteItem** удаляет пункт popup-меню и все пункты связанного подменю.

Пример:

```
PopupMgr.AddItem('&Insert','Insert')          !Добавить пункт меню
PopupMgr.AddItem('&Change','Change')          !Изменить его
PopupMgr.AddItem('&Delete','Delete')          !Добавить пункт меню
PopupMgr.AddItem('&Select','Select')          !Добавить пункт меню
IF No_Records_Found
PopupMgr.DeleteItem('Change')                 !Удалить пункт меню Изменить
PopupMgr.DeleteItem('Delete')                 !Удалить пункт меню Удалить
PopupMgr.DeleteItem('Select')                 !Удалить пункт меню Выбрать
END
```

Смотри также: AddItem, AddItemMimic, AddMenu

GetItemChecked (возвратить статус переключаемого пункта меню)

GetItemChecked(*имя*)

GetItemChecked Возвращает статус переключаемого пункта меню.
имя Константа типа string, переменная, метка соответствия или выражение, содержащее *имя* пункта меню. Методы AddItem, AddItemMimic или AddMenu устанавливают имя пункта меню.

GetItemChecked метод возвращает единицу (1), если пункт меню «включен», и ноль (0), если пункт меню «не включен». Метод SetItemCheck устанавливает состояние переключаемого пункта меню.

Возвращаемый тип данных: BYTE

Пример:

```
IF PopupMgr.Ask() = 'Disable'                 !если выбран пункт меню Disable
IF PopupMgr.GetItemChecked('Disable')         ! если пункт меню «включен»
PopupMgr.SetItemCheck('Disable',False)        ! выключить его
ENABLE(?Save)                                 ! совершить соответствующие
                                              !действия
```

```
        ELSE                                ! если пункт меню «не включен»
PopupMgr.SetItemCheck('Disable',True)       ! включить его
DISABLE(?Save)                              ! совершить соответствующие действия
        END
        END
```

Смотри также: AddItem, AddItemMimic, AddMenu, SetItemCheck

GetItemEnabled (вернуть статус пункта меню)

GetItemEnabled(*имя*)

GetItemEnabled Возвращает статус (доступно/недоступно для выбора) пункта меню.
имя Целочисленная константа, переменная, метка соответствия, или выражение, содержащее имя пункта меню. Методы AddItem, AddItemMimic или AddMenu устанавливают имя пункта меню.

Метод **GetItemEnabled** возвращает единицу (1), если пункт меню доступен для выбора, и ноль (0), если пункт меню недоступен. Метод SetItemEnable устанавливает состояние (доступно/недоступно для выбора) переключаемого пункта меню.

Возвращаемый тип данных: BYTE

Пример:

```
IF PopupMgr.GetItemEnabled('Save')      !если пункт менюдоступен
PopupMgr.SetItemEnable('Save',False)    ! сделать его недоступным
ELSE                                    !если пункт меню недоступен
PopupMgr.SetItemEnable('Save',True)     ! сделать его доступным
END
```

Смотри также: AddItem, AddItemMimic, AddMenu, SetItemEnable

GetLastSelection (вернуть выбранный пункт меню)

GetLastSelection

GetLastSelection метод возвращает имя последнего выбранного пункта меню.

Методы AddItem, AddItemMimic, AddMenu или AddSubMenu устанавливают имя пункта меню.

Возвращаемый тип данных: STRING

Пример:

```
MenuChoices  EQUATE('Fred|Barney|Wilma')
                                        ! Объявить строку определения меню
        CODE
```

```
PopupMgr.AddMenu(MenuChoices)        !добавить Popup menu
                                     !program code
PopupMgr.Ask()                       !отобразить меню
MESSAGE('Thank you for choosing '&PopupMgr.GetLastSelection)
```

Смотри также: AddItem, AddItemMimic, AddMenu, AddSubMenu

Init (initialize the PopupClass object)

Init([*INI-класс*] **)**

Init	Инициализирует объект Popup-класса
INI-класс	Метка объекта INI-класса для этого объекта Popup-класса. Save метод использует объект INI- класса для сохранения описания меню в INI файле; Restore метод использует его для восстановления сохраненного описания меню. Если параметр опущен, Save и Restore методы не производят никаких действий.

Метод **Init** инициализирует объект Popup-класса.

Пример:

```
PopupMgr PopupClass                  !объявить объект PopupMgr
INIMgr   INIClass                    ! объявить объект INIMgr
CODE
PopupMgr.Init(INIMgr)                ! инициализировать объект PopupMgr
PopupMgr.AddItem('Save Popup','Save')        !добавить пункт меню: Save
PopupMgr.AddItemMimic('Save',?Save)          !уподобить действие пункта меню
                                             !Save кнопке ?Save
```

Смотри также: Restore, Save

Kill (закрыть объект Popup-класса)

Kill

Метод **Kill** освобождает любую память, размещенную в течение жизни объекта Popup-класса и исполняет любой другой требуемый код завершения.

Пример:

```
PopupMgr.Init                        ! инициализировать объект PopupMgr
                                     !Код программы
PopupMgr.Kill                        !закрыть объект PopupMgr
```

Restore (восстановить сохраненное меню)

Restore(*имя* **)**

Restore	Восстанавливает меню, ранее сохраненное методом Popup-класса.
имя	Целочисленная константа, переменная, метка соответствия или выражение,

содержащее имя пункта меню, подлежащее восстановлению.

Restore метод восстанавливает меню, сохраненное методом Save. Restore метод восстанавливает все атрибуты меню, включая связанные с пунктом меню действия, о которых известно объекту Popup-класса.

Реализация: Для Restore метода необходим объект INI-класса. Объект INI-класса определяется методом Init.

Пример:

```
PopupMgr PopupClass                    !объявить объект PopupMgr
INIMgr   INIClass                      ! объявить объект INIMgr
MenuChoices  EQUATE('&Save Menu|&Restore Menu|-|&Close')
                                       !объявить описание меню
CODE
PopupMgr.Init(INIMgr)                  ! инициализировать объект PopupMgr
PopupMgr.AddMenu(MenuChoices)          !добавить Popup-меню
ACCEPT
CASE FIELD()
OF ?Save
CASE EVENT()
OF EVENT:Accepted
PopupMgr.Save('MyPopup')               !сохранить описание меню в INI-файле
END
OF ?Restore
CASE EVENT()
OF EVENT:Accepted
PopupMgr.Restore('MyPopup')            ! восстановить описание меню из INI-файла
          END
          END
          END
```

Смотри также: Init, Save

Save (сохранить меню для последующего восстановления)

Save(*имя* **)**

Save	Сохраняет меню для восстановления методом PopupClass.Restore.
имя	Целочисленная константа, переменная, метка соответствия или выражение, содержащее имя пункта меню, подлежащее сохранению.

Метод **Save с**охраняет меню для последующего восстановления методом Restore. Save метод сохраняет все атрибуты меню, включая связанные с пунктом меню

действия, о которых известно объекту Popup-класса.

Реализация: Для Save метода необходим объект INI-класса. Объект INI-класса определяется методом Init.

Пример:

```
PopupMgr PopupClass                    !объявить объект PopupMgr
INIMgr     INIClass                    ! объявить объект INIMgr
MenuChoices EQUATE('&Save Menu|&Restore Menu|-|&Close')
                                       !объявить описание меню
CODE
PopupMgr.Init(INIMgr)                  ! инициализировать объект PopupMgr
PopupMgr.AddMenu(MenuChoices)          !добавить Popup-меню
ACCEPT
CASE FIELD()
OF ?Save
CASE EVENT()
OF EVENT:Accepted
PopupMgr.Save('MyPopup')               !сохранить описание меню в INI-файле
END
OF ?Restore
CASE EVENT()
OF EVENT:Accepted
PopupMgr.Restore('MyPopup')            ! восстановить описание меню из INI-файла
END
END
          END
```

Смотри также: Init, Restore

SetItemCheck (установить статус переключаемого пункта меню)

SetItemCheck(*имя, статус***)**

SetItemCheck	Установить статус переключаемого пункта меню.
имя	Целочисленная константа, переменная, метка соответствия, или выражение, содержащее имя пункта меню. Методы AddItem, AddItemMimic или AddMenu устанавливают имя пункта меню.
Статус	Булева константа, переменная, метка соответствия или выражение, содержащее значение, определяющее статус пункта меню. Значение *статуса* равное единице (1) отображает «галочку» на пункте меню («включено»); ноль (0) приводит к тому, что пункт меню отображается без «галочки».

SetItemCheck метод устанавливает статус переключаемого пункта меню. GetItemChecked метод возвращает статус переключаемого пункта меню.

Пример:

```
IF PopupMgr.Ask() = 'Disable'                    !если выбран пункт меню Disable
IF PopupMgr.GetItemChecked('Disable')            !если пункт меню «включен»
PopupMgr.SetItemCheck('Disable',False)           ! выключить его
ENABLE(?Save)                          ! произвести соответствующие действия
ELSE                                   ! если пункт меню «выключен»
PopupMgr.SetItemCheck('Disable',True)            ! включить его
DISABLE(?Save)                         ! произвести соответствующие действия
        END
        END
```

Смотри также: AddItem, AddItemMimic, AddMenu, GetItemChecked

SetItemEnable (установить статус пункта меню)

SetItemEnable(*имя* **)**

SetItemEnable Установить статус (доступен/недоступен для выбора) у пункта меню.

имя Целочисленная константа, переменная, метка соответствия или выражение, содержащее имя пункта меню. Методы AddItem, AddItemMimic или AddMenu устанавливают имя пункта меню.

статус Булева константа, переменная, метка соответствия, или выражение, содержащее значение, определяющее статус пункта меню. Значение *статуса* равное единице (1) включает пункт меню; ноль (0) выключает.

SetItemEnable метод устанавливает статус (доступен/недоступен для выбора) у пункта меню. Метод GetItemEnabled возвращает статус (доступен/недоступен) пункта меню.

Пример:

```
IF PopupMgr.GetItemEnabled('Save')       !если пункт меню доступен
PopupMgr.SetItemEnable('Save',False)     ! сделать его недоступным
ELSE                                     ! если пункт меню недоступен
PopupMgr.SetItemEnable('Save',True)      ! сделать его доступным
END
```

Смотри также: AddItem, AddItemMimic, AddMenu, GetItemEnabled

SetTranslator (set run-time translator)

SetTranslator(*транслятор* **)**

SetTranslator Устанавливает объект Translator-класса для объекта Popup-класса.

транслятор Метка объекта Translator-класса для этого объекта Popup-класса.

Метод **SetTranslator** определяет объект Translator-класса для объекта popup-класса. Определяя объект Translator-класса для объекта popup-класса, Вы можете автоматически перевести текст popup-меню, однако объект Translator-класса не сможет перевести различные popup-меню в обратном направлении, потому что они не являются частью структуры окна.

Реализация: Метод Ask использует объект Translator-класса для перевода текста popup-меню перед его отображением.

Пример:

```
PopupMgr PopupClass                  !объявить объект PopupMgr
Translator  TranslatorClass          ! объявить объект Translator
MenuChoicesEQUATE('&Save Menu|&Restore Menu|&Close')
                                     ! объявить описание меню
CODE
  Translator.Init                    !инициализировать объект Translator
PopupMgr.Init(INIMgr)                ! инициализировать объект PopupMgr
PopupMgr.AddMenu(MenuChoices)        !добавить Popup-меню
PopupMgr.SetTranslator(Translator)   !разрешить перевод текста popup-меню
                                     !program code
  PopupMgr.Ask()
                                    !показать переведенное popup-меню
```

Смотри также: Ask

ViewMenu (отладчик Popup-меню)

ViewMenu

Метод ViewMenu отображает информацию о структуре popup-меню, созданного различными ' Add ' методами.

Реализация: ViewMenu метод работает только тогда, когда программа компилируется со включенной отладочной информацией. Смотри *Отладчики* в *Руководстве Пользователя* для дополнительной информации.

Класс изменения размеров окна

Краткий обзор

Понятие Изменения Размера Окна

Класс-Изменения-Размеров-Окна позволяет конечному пользователю изменять размеры окна, которые первоначально имели размер, позволяющий видеть на окне все принадлежащие ему средства управления (окна списка, поля ввода, кнопки, вложенные элементы управления и т.д.). Класс- Изменения-Размеров-Окна *интеллектуально* переразмещает элементы управления, изменяет размеры элементов управления, или делает и то, и другое, когда конечный пользователь изменяет размеры окна.

Интеллектуальное переразмещение выполняется с учетом того, что имеется много различных типов элементов управления, каждый из которых имеет уникальную «политику переразмещения и изменения размеров» (т.е. свои требования по изменению параметров размера и размещения). Изменение-Размеров-Окна-Класс также учитывает то, что элементы управления зачастую оказываются вложенными друг в друга, и рассматривает, с чем координаты данного элемента управления более тесно связаны: с координатами окна или с координатами другого элемента управления. То есть интеллектуальное переразмещение правильно распознает родителя каждого элемента управления. См. *SetParentControl* для получения дополнительной информации о сути концепции отношений «родитель-ребенок».

Интеллектуальное переразмещение включает несколько общих стратегий, которые применяются ко всем элементам управления на окне, а также стратегии изменения размеров и переразмещения конкретного средства управления. Общие стратегии включают:

Поверхность	Размещает элементы управления (окно списка, лист, панель, изображение) на окне таким образом, чтобы максимизировать их размеры (т.е. использовать доступное количество пикселов на окне по максимуму). Эта стратегия рекомендуется для всех окон, сгенерированных с использованием шаблона.
Распространение	Осуществляет изменение размеров окна таким образом, чтобы сохранялся первоначальный его вид (заданный в процессе разработки), что достигается путем применения к каждому элементу

управления своей стратегии переразмещения и изменения размеров. Например, размеры кнопки не изменяются, но ее положение привязано к ближайшему краю окна. Или, напротив, размеры окна списка и его положение масштабируются пропорционально окну.

Изменение размеров Все элементы управления масштабируются пропорционально окну.

См. *SetStrategy* для получения дополнительной информации о стратегиях изменения размеров для конкретного элемента управления.

Обратите внимание: Чтобы разрешить окну изменять свои размеры, необходимо установить тип структуры окна «Изменяющий размеры». Также рекомендуется добавить атрибут MAX. См. *Конструктор Окна — Диалог Свойств Окна* в *Руководстве Пользователя* для получения дополнительной информации об этих установках.

Связь с другими классами разработки приложений

Класс -Изменения-Размеров-Окна независим от других классов разработки приложений. Он не связан с другими ABC классами, и другие ABC классы также не связаны с ним.

Реализация ABC шаблона

Шаблоны ABC обрабатывают объект Изменение-Размеров-Окна-Класса для каждого шаблона WindowResize в приложении (обычно по одному для каждой управляющей окном процедуры). Шаблоны могут также получать класс из Изменение-Размеров-Окна-Класса. Полученный класс (и его объект) называется Resize. Полученный класс обслуживают шаблоны ABC, так что можно использовать WindowResize шаблон **Classes,** чтобы легко менять от случая к случаю поведение Resize.

Объект, порожденный от полученного класса, называется Resize. Этот объект поддерживает все функциональные возможности, определенные в WindowResize шаблоне. См., *Другие Шаблоны — Изменение Размеров Окна* для получения дополнительной информации о реализации шаблона этого класса.

Изменение Размеров Окна: Исходные Файлы

По умолчанию файлы-источники Изменение-Размеров-Окна-Класса установлены в директории \CLARION4\LIBSRC. Предназначенные для Изменение-Размеров-Окна-Класса файлы и их соответствующие компоненты представлены здесь:

ABRESIZE.INC	Объявление Изменение-Размеров-Окна-Класса
ABRESIZE.CLW	Определения метода Изменение-Размеров-Окна-Класса

Пояснительный Пример

Следующий пример показывает типовую последовательность операторов, предназначенных для объявления, обработки, инициализации, использования и завершения объекта Изменение-Размеров-Окна-Класса. Этот пример иллюстрирует стратегию «Поверхность» плюс некоторые специфические стратегии для конкретного средства управления. Этот программный код не производит никаких действий, кроме как демонстрирует окно с типовым набором средств управления.

```
PROGRAM
INCLUDE('ABRESIZE.INC')                !объявить Изменение-Размеров-Окна-Класс
MAP
END
Resizer     WindowResizeClass          ! объявить объект Изменяющий Размеры
ClientQ     QUEUE,PRE(CLI)             ! объявить очередь списка
Name        STRING(20)
State       STRING(2)
END                                    ! для окна нужны атрибуты IMM и RESIZE
window      WINDOW('Client Information'),AT(,,185,100),IMM,GRAY,MAX,RESIZE
SHEET,AT(3,3,180,78),USE(?Sheet1)
TAB('Client List'),USE(?ListTab)
LIST,AT(10,20,165,55),USE(?List1),FROM(ClientQ),|
FORMAT('87L~Name~@s20@8L~State Code~@s2@')
END
TAB('Client Logo'),USE(?LogoTab)
IMAGE('TopSpeed.gif'),AT(50,35),USE(?CLI:Logo)
END
END
PROMPT('Locate:'),AT(7,87),USE(?LocatorPrompt)
ENTRY(@s20),AT(33,86,61,12),USE(CLI:Name)
BUTTON('Restore'),AT(110,84),USE(?Restore)
BUTTON('Close'),AT(150,84),USE(?Close)
END
CODE
OPEN(window)
window{PROP:MinWidth}=window{PROP:Width}
                                       !установить минимальную ширину окна
window{PROP:MinHeight}=window{PROP:Height}
                                       !установить минимальную высоту окна
Resizer.Init(AppStrategy:Surface)
                                       !инициализация объекта Изменяющего
                                       !Размеры
Resizer.SetStrategy(?LocatorPrompt,  |
```

```
        !установить специфическую стратегию элемента:
Resize:FixLeft+Resize:FixBottom,Resize:LockSize)
        ! оставаться слева внизу & фиксированный размер
Resizer.SetStrategy(?CLI:Name,          |
        ! установить специфическую стратегию элемента:
Resize:FixLeft+Resize:FixBottom,Resize:LockHeight)
        !оставаться слева внизу & фиксированная высота
ACCEPT
CASE EVENT()
OF EVENT:CloseWindow                    !при закрытии окна,
Resizer.Kill                            !погасить объект Изменяющий Размеры
OF EVENT:Sized                          !на окне с заданными размерами,
Resizer.Resize                          ! изменить размеры и положение элементов
                                        !управления
END                                     ! применить указанные выше стратегии
CASE ACCEPTED()
OF ?Restore
Resizer.RestoreWindow                   !восстановить исходные размеры окна
OF ?Close
POST(Event:CloseWindow)
        END
        END
```

Свойства Изменение-Размеров-Окна-Класса

Изменение-Размеров-Окна-Класс обладает следующими свойствами:

AutoTransparent (оптимизировать перерисовку)

AutoTransparent BYTE

Свойство **AutoTransparent** в течение процесса изменения размеров окна указывает, сделан ли прозрачным элемент управления, который является носителем этого атрибута (атрибут TRN). Прозрачные элементы управления дают меньшее мерцание при перерисовке и более плавное изменение размеров, а также снижают вероятность ошибок Windows на некоторых окнах.

Значение единицы (1) делает средство управления прозрачным; значение нуля (0) - наоборот.

DeferMoves (оптимизировать изменение размеров)

DeferMoves BYTE

DeferMoves свойство указывает, следует ли отсрочить движение элемента управления до конца ACCEPT-цикла (см. *PROP:DEFERMOVE* в *Описании Языка*). Это позволяет исполняющей библиотеке исполнять все движения элемента управления сразу, что приводит к более чистому, мгновенному изменению размеров, а также к снижению вероятности ошибок Windows на некоторых окнах.

Значение единицы (1) производит отсрочку движения элемента управления; значение ноля (0) - нет.

Методы Изменение-Размеров-Окна-Класса

Изменение-Размеров-Окна-Класс содержит методы, описанные ниже.

Функциональная Организация — Предполагаемое Использование

С целью лучшего понимания Изменение-Размеров-Окна-Класса, полезно структурировать различные его методы в две больших категории согласно их предполагаемому использованию — первичный интерфейс и виртуальные методы. Эта организация отражает то, что, на наш взгляд, является типовым применением методов Изменение-Размеров-Окна-Класса.

Первичные Интерфейсные Методы

Первичные интерфейсные методы, которые, скорее всего, Вы будете вызывать из своей программы, могут быть далее разделены на три категории:

Используются один раз:

Init	инициализирует объект Изменение-Размеров-Окна-Класса
Kill	закрывает объект Изменение-Размеров-Окна-Класса

Используются в основном:

Resize[V]	изменяет размеры и переразмещает все элементы управления

Используются изредка:

SetParentControl	установить родителя элемента управления
SetStrategy	установить стратегию переразмещения и изменения размеров элемента управления

[V] Эти методы также являются виртуальными.

Виртуальные Методы

Обычно не приходиться вызывать эти методы непосредственно, поскольку они вызываются первичными интерфейсными методами. Однако, вполне вероятно, что Вы захотите отказаться от использования этих методов, и, поскольку они виртуальные, сделать это будет довольно легко. Эти методы обеспечивают поведение по умолчанию в том случае, если Вы не хотите их переопределить.

SetParentDefaults	установить родителей для всех элементов управления
RestoreWindow	восстановить исходные размеры окна
GetParentControl	вернуть родителя элемента управления
Resize	изменить размеры и положение всех элементов управления

GetParentControl (вернуть родителя элемента управления)

GetParentControl(*элемент управления* **), VIRTUAL**

GetParentControl возвращает родителя для оконного *элемента управления*.

элемент управления Целочисленная константа, переменная, метка соответствия, или выражение, содержащее номер элемента управления. Метод Resize заново масштабирует *элемент управления* по мере изменения координат родителя.

GetParentControl метод возвращает родителя для *элемента управления* окна. Нулевое возвращаемое значение указывает, что родителем является текущее окно. В противном случае возвращаемое значение - метка соответствия другого оконного элемента управления.

SetParentDefaults метод интеллектуально устанавливает соответствующего родителя для всех средств управления окна, а метод SetParentControl устанавливает родителя для конкретного элемента управления. Метод Resize заново масштабирует *элемент управления* по мере изменения координат родителя.

GetParentControl - виртуальный метод, так что другие основные методы класса могут прямо вызывать метод GetParentControl в полученном классе. Это значительно облегчает реализацию вашей программной собственной версии этого метода.

Возвращаемый тип данных: SIGNED

Пример:

```
window     WINDOW('Nested Controls'),AT(,,165,97),IMM,GRAY,MAX,RESIZE
GROUP('OuterGroup'),AT(5,3,154,92),USE(?OuterGroup),BOXED
                                                    BUTTON('Button
1'),AT(14,23),USE(?Button1)
```

```
ENTRY(@s20),AT(60,24),USE(Entry1)
GROUP('InnerGroup'),AT(11,49,141,38),USE(?InnerGroup),BOXED
CHECK('Check 1'),AT(32,64),USE(Check1)
CHECK('Check 2'),AT(91,64),USE(Check2)
.                .                .
CODE
OPEN(window)
Resizer.Init(AppStrategy:Spread)     !инициализация объекта, изменяющего размеры
Resizer.SetParentDefaults       !установить родителей для всех элементов управления
Resizer.SetParentControl(?Button1,?OuterGroup)          !поменять родителя для
                                                        !элемента управления
Resizer.SetParentControl(?Check1,?InnerGroup)           !поменять родителя для
                                                        !элемента управления
Resizer.SetParentControl(?Check2,?InnerGroup)           !поменять родителя для
                                                        !элемента управления
```

Смотри также: Resize, SetParentControl, SetParentDefaults

GetPositionStrategy (возвратить стратегию переразмещения для типа элемента управления)

GetPositionStrategy(*Тип элемента управления* [, *стратегия*] **)**

GetPositionStrategy	Возвращает стратегию переразмещения и изменения размеров.
тип элемента управления	Целочисленная константа, переменная, метка соответствия, или выражение, показывающая тип элемента управления (кнопка, поле ввода, окно списка и т.д.).
стратегия	Целочисленная константа, переменная, метка соответствия, или выражение, отображающая общую стратегию переразмещения и изменения размеров всех оконных элементов управления. Если параметр пропущен, за *стратегию* по умолчанию принимается та, которая задана методом Init.

Метод **GetPositionStrategy в**озвращает соответствующую стратегию переразмещения для конкретного типа элемента управления, основанную на общей стратегии.

Реализация: Метод Reset вызывает метод GetPositionStrategy для задания стратегии размещения динамически созданных элементов управления.

Метки соответствия для параметра *тип элемента управления* объявлены в файле EQUATES.CLW. Метка соответствия каждого типа элемента управления определяется оператором CREATE:.

Метки соответствия для возвращаемого значения объявлены в файле

ABRESIZE.INC. Метка соответствия каждой стратегии определяется оператором Resize.

Пример:

```
GET(SELF.ControlQueue,SELF.ControlQueue.ID)
        !получить информацию об изменении размеров элемента управления
IF ERRORCODE()                          !если этой информации нет, то добавить ее
SELF.ControlQueue.Type=FieldCounter{PROP:Type}      ! установить тип элемента
                                                    !управления
SELF.ControlQueue.ParentID=0                        ! установить родителя
SELF.ControlQueue.HasChildren=False
                                        ! установить порожденные элементы
                                        !управления
SELF.ControlQueue.ID=FieldCounter   ! установить ID
GetSizeInfo(FieldCounter,SELF.ControlQueue.Pos)     ! установить координаты
                                                    ! установить стратегию
                                        !переразмещения и изменения размеров
SELF.ControlQueue.PositionalStrategy=SELF.GetPositionStrategy(SELF.ControlQueue.Type)

SELF.ControlQueue.ResizeStrategy=SELF.GetResizeStrategy(SELF.ControlQueue.Type)
ADD(SELF.ControlQueue,SELF.ControlQueue.ID)                                    !
        !добавить информацию об изменении размеров элемента управления
ASSERT(~ERRORCODE())
END
```

Смотри также: Init, Reset

GetResizeStrategy (вернуть стратегию изменения размеров для типа элементов)

GetResizeStrategy(*Тип элемента управления* [, *стратегия*] **)**

GetResizeStrategy	Возвращает стратегию изменения размеров для *типа элемента управления.*
GetResizeStrategy	Возвращает стратегию изменения размеров для *типа элемента управления.*
Тип элемента управления	Целочисленная константа, переменная, метка соответствия, или выражение, указывающее тип элемента управления (кнопка, поле ввода, окно списка и т.д.).
Стратегия	Целочисленная константа, переменная, метка соответствия, или выражение, указывающее общую стратегию изменения размеров и переразмещения всех элементов управления на окне. Если этот параметр опущен, *стратегией* по умолчанию является та, которая определена методом Init.

Метод **GetResizeStrategy** возвращает соответствующую стратегию изменения размеров для конкретного типа элемента управления.

Реализация: Метод Reset вызывает метод GetPositionStrategy для задания стратегии изменения размеров динамически созданных элементов управления.

Метки соответствия для параметра *тип элемента управления* объявлены в файле EQUATES.CLW. Метка соответствия каждого типа элемента управления определяется оператором CREATE:.

Метки соответствия для возвращаемого значения объявлены в файле ABRESIZE.INC. Метка соответствия каждой стратегии определяется оператором Resize.

Возвращаемый тип данных: USHORT

Пример:

```
GET(SELF.ControlQueue,SELF.ControlQueue.ID)
                                    !получить информацию об изменении
                                    !размеров элемента управления
IF ERRORCODE()                      !если этой информации нет, то добавить ее
SELF.ControlQueue.Type=FieldCounter{PROP:Type}      ! установить тип элемента
                                                    !управления
SELF.ControlQueue.ParentID=0        ! установить родителя
SELF.ControlQueue.HasChildren=False
                                    ! установить порожденные элементы
                                    !управления
SELF.ControlQueue.ID=FieldCounter
                                    ! установить ID
GetSizeInfo(FieldCounter,SELF.ControlQueue.Pos)     ! установить координаты
                                                    ! установить стратегию

SELF.ControlQueue.PositionalStrategy=SELF.GetPositionStrategy(SELF.ControlQueue.Type)

SELF.ControlQueue.ResizeStrategy=SELF.GetResizeStrategy(SELF.ControlQueue.Type)
ADD(SELF.ControlQueue,SELF.ControlQueue.ID)
                                                ! добавить информацию об
                                    !изменении !размеров элемента управления
ASSERT(~ERRORCODE())
    END
```

Смотри также: Init, Reset

Init (Инициализировать объект Изменение-Размеров-Окна-Класса)

Init ([*стратегия*] [, *минимальный размер*] [, *максимальный размер*]**))**

Init Инициализирует объект Изменение-Размеров-Окна-Класса.

Стратегия Целочисленная константа, переменная, метка соответствия, или выражение, указывающее общую стратегию изменения размеров и переразмещения всех элементов управления на окне. Если параметр опущен, *стратегией* по умолчанию является **AppStrategy:Resize**, которая изменяет размеры всех элементов управления пропорционально родителю.

Минимальный размер Целочисленная константа, переменная, метка соответствия, или выражение, указывающее минимальный размер окна. Значение единица (1) устанавливает минимальный размер окна равным его размеру в проекте. Если параметр опущен, *минимальный размер* по умолчанию считается равным нулю (0), что означает отсутствие минимального размера.

Максимальный размер Целочисленная константа, переменная, метка соответствия, или выражение, указывающее максимальный размер окна. Значение единица (1) устанавливает максимальный размер окна равным его размеру в проекте. Если параметр опущен, *максимальный размер* по умолчанию считается равным нулю (0), что означает отсутствие максимального размера.

Init метод инициализирует объект Изменение-Размеров-Окна-Класса и устанавливает общую стратегию изменения размеров и переразмещения элементов управления окна. Вы можете использовать метод SetStrategy, чтобы отказаться от использования общей стратегии в отношении конкретного элемента управления.

Реализация: Если параметр *стратегия* присутствует, Init применяет стратегию к каждому элементу управления, основываясь на значении соответствующего параметра. Если же параметр *стратегия* отсутствует, Init по умолчанию применяет стратегию к каждому элементу управления. *Стратегия* по умолчанию пересчитывает все координаты элемента управления (координаты x и y, ширина, и высота) пропорционально родительским.

Родителем может быть окно, содержащее элемент управления, или другой элемент управления на этом окне. Методы SetParentControl и SetParentDefaults определяют родителя для данного элемента управления.

Метки соответствия параметра *стратегия* объявлены в файле RESIZE.INC следующим образом:

```
ITEMIZE(0),PRE(AppStrategy)
```

```
Resize      EQUATE                  !пропорционально изменить размеры всех
элементов управления
Spread      EQUATE                  !Сохранить у окна тот вид, который оно
имело при проектировании
Surface     EQUATE                  !Максимизировать количество доступных
пикселов
            END
```

Цель и последствия применения этих стратегий:

Изменение размеров	Масштабирует все оконные координаты в соответствии с изменением размеров и местоположения родителя, сохраняя таким образом относительные размеры и положение всех средств управления. Эта стратегия применяется по умолчанию.
Поверхность	Использует доступные пикселы окна по максимуму, размещая элементы управления так, чтобы максимизировать размеры окна списка, панели, листа, изображения.
Распространение	Сохраняет у окна тот вид, который был ему задан в процессе проектирования, путем применения следующих стратегий (по типам элементов управления):
BUTTON	Горизонтальное и Вертикальное положение (координаты X и Y) зафиксированы относительно края самого близкого родительского элемента; ширина и высота неизменны.
RADIO	Горизонтальное и вертикальное положение изменяются по мере изменения координат родительского элемента, но ширина и высота неизменны.
CHECK	Горизонтальное и вертикальное положение изменяются по мере изменения координат родительского элемента, но ширина и высота неизменны.
ENTRY	Горизонтальное, вертикальное положение и ширина изменяются по мере изменения координат родительского элемента, но высота неизменна.
COMBO+DROP	Горизонтальное, вертикальное положение и ширина изменяются по мере изменения координат родительского элемента, но высота неизменна.
LIST+DROP	Горизонтальное, вертикальное положение и ширина изменяются по мере изменения координат родительского элемента, но высота неизменна.
SPIN	Горизонтальное, вертикальное положение и ширина изменяются по мере изменения координат родительского элемента, но высота неизменна.
Остальные	Все координаты изменяются по мере изменения координат родительского элемента.

Совет: Несмотря на то, что такие элементы управления, как ОКНО СПИСКА и КОМБИНИРОВАННОЕ ОКНО СПИСКА могут менять размеры, ширина колонок в них остается неизменной. Тем не менее, самая правая колонка расширяется или сжимается (в зависимости от наличия свободного места).

Пример:

```
OPEN(window)
Resizer.Init(AppStrategy:Surface)      !Ининициализация объекта Resizer
ACCEPT
CASE EVENT()
OF EVENT:CloseWindow                   !при закрытии окна,
Resizer.Kill                           ! закрыть объект Resizer
OF EVENT:Sized                         !при изменении размеров окна,
Resizer.Resize
                    ! изменить размеры и положение элемента управления
END
END
```

Смотри также: SetParentControl, SetParentDefaults, SetStrategy

Kill (Закрыть объект Изменение-Размеров-Окна-Класса)

Kill

Kill метод освобождает любую память, размещенную в течение жизни объекта и выполняет любой другой требуемый код завершения.

Пример:

```
OPEN(window)
Resizer.Init(AppStrategy:Surface)      !инициализация объекта Resizer
ACCEPT
CASE EVENT()
OF EVENT:CloseWindow                   !при закрытии окна,
Resizer.Kill                           ! закрыть объект Resizer
OF EVENT:Sized                         !при изменении размеров окна,
Resizer.Resize                         ! изменить размеры и положение
                                       !элемента управления
   END
END
```

Reset (Переустановить объект Изменение-Размеров-Окна-Класса)

Reset, VIRTUAL

Reset метод переустанавливает объект Изменение-Размеров-Окна-Класса с целью его соответствия окну в текущем состоянии.

Reset является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать Reset метод в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Init метод вызывает Reset метод. Reset метод хранит начальные координаты окна и его элементов управления. Объект Изменение-Размеров-Окна-Класса использует сохраненные координаты для восстановления окна, установления связи родитель - ребенок между элементами управления и т.д.

Пример:

```
ThisWindow.Init PROCEDURE()
ReturnValue       BYTE,AUTO
 CODE                                    !procedure code
 Resizer.Init(AppStrategy:Surface,Resize:SetMinSize)
 SELF.AddItem(Resizer)
 Resizer.AutoTransparent=True
 Resizer.SetParentDefaults
 INIMgr.Fetch('BrowseMembers',QuickWindow)
 Resizer.Resize                          ! Resize необходим, если окно изменено
                                         !объектом INIMgr
 Resizer.Reset                           !Reset необходим, если окно изменено
                                         !объектом INIMgr
 SELF.SetAlerts()
 RETURN ReturnValue
```

Смотри также: Init

Resize (Изменение размеров и местоположения элементов управления)

Resize, VIRTUAL, PROC

Resize метод изменяет размеры и местоположение каждого элемента управления на окне, применяя указанную стратегию к каждому элементу управления, а также возвращает значение, указывающее, продолжается ли обработка ACCEPT-цикла или надо остановиться.

Resize возвращает Level:Benign для того, чтобы указать, что обработка события (обычно это событие EVENT:Sized) должна продолжаться в нормальном режиме; Level:Notify, чтобы указать, что обработка события закончена и управление ACCEPT-циклом передается в начало; и Level:Fatal, если нужно указать, что событие не может быть обработано, и ACCEPT-цикл должен быть прерван.

Init метод и SetStrategy метод определяют те стратегии, которые будут применены к каждому элементу управления. Все стратегии изменения размеров учитывают новые координаты "родителя" каждого элемента управления. По умолчанию родителем каждого элемента управления считается окно. Однако, с помощью метода SetParentControl можно определить любой элемент управления как родитель для любого другого элемента управления.

Resize метод является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод Resize в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Возвращаемый тип данных: BYTE

Пример:

```
OPEN(window)
Resizer.Init(AppStrategy:Surface)       !инициализация объекта Resizer(общая
                                        !стратегия)
Resizer.SetStrategy(?CloseButton,       |
                                        !установить свою стратегию изменения
                                        !размеров для элемента управления:
Resize:FixRight+Resize:FixBottom,Resize:LockSize)      ! удерживать внизу справа и
                                        !зафиксировать размер
        ACCEPT
        CASE EVENT()
        OF EVENT:CloseWindow            !при закрытии окна,
        Resizer.Kill                    ! закрыть объект Resizer
        OF EVENT:Sized                  !при изменении размеров окна,
        Resizer.Resize                  ! изменить размеры и положение элемента
                                        !управления
        END
        END
```

Смотри также: Init, SetStrategy, SetParentControl

RestoreWindow (восстановить первоначальные размеры окна)

RestoreWindow, VIRTUAL

RestoreWindow метод возвращает окну и всем элементам управления на нем их размеры на момент выполнения метода Init.

RestoreWindow метод является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод RestoreWindow в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Пример:

```
OPEN(window)
Resizer.Init(AppStrategy:Surface)        !инициализация общей стратегии для
                                         !Resizer
        ACCEPT
        CASE EVENT()
        OF EVENT:CloseWindow
        Resizer.Kill                     ! закрыть объект Resizer
        OF EVENT:Sized
        Resizer.Resize                   ! изменить размеры и местоположение
                                         !элемента управления
        END
        CASE ACCEPTED()
        OF ?RestoreButton
        Resizer.RestoreWindow            !восстановить у окна первоначальные
                                         !размеры

        END
        END
```

Смотри также: Init

SetParentControl (установить родительский элемент управления)

SetParentControl(*Элемент управления* [, *родитель*] **)**

SetParentControl
Элемент управления

Устанавливает *родителя* для *элемента управления* на окне. Целочисленная константа, переменная, метка соответствия, или выражение, содержащее номер элемента управления.

Метод Resize масштабирует *элемент управления* в зависимости от координат *родителя*.

Родитель Целочисленная константа, переменная, метка соответствия, или выражение, содержащее номер элемента управления. Метод Resize масштабирует *элемент управления* в зависимости от координат *родителя*. Если параметр опущен, то по умолчанию *родителем* является окно.

SetParentControl метод устанавливает *родителя* для оконного *элемента управления*. Метод Resize масштабирует *элемент управления* в зависимости от координат *родителя*.

Это позволяет изменять размеры конкретного элемента управления, основываясь на координатах связанного элемента управления, а не на координатах окна. Это применимо в тех случаях, когда стратегия, приложенная к родительскому элементу управления, приводит к тому, что его размеры меняются непропорционально размерам окна. Например, элемент управления в пределах структуры ГРУППА, размер которой зафиксирован, может быть высчитан относительно координат ГРУППЫ, а не окна.

SetParentDefaults метод интеллектуально устанавливает соответствующего родителя для каждого элемента управления на окне, так что использовать метод SetParentControl придется только в том случае, если SetParentDefaults установит какому-либо элементу управления несоответствующего родителя.

GetParentControl метод возвращает номер родительского элемента управления.

Пример:

```
window     WINDOW('Nested Controls'),AT(,,165,97),IMM,GRAY,MAX,RESIZE
GROUP('OuterGroup'),AT(5,3,154,92),USE(?OuterGroup),BOXED
BUTTON('Button 1'),AT(14,23),USE(?Button1)
ENTRY(@s20),AT(60,24),USE(Entry1)
GROUP('InnerGroup'),AT(11,49,141,38),USE(?InnerGroup),BOXED
CHECK('Check 1'),AT(32,64),USE(Check1)
CHECK('Check 2'),AT(91,64),USE(Check2)
.          .                    .
CODE
OPEN(window)
Resizer.Init(AppStrategy:Spread)        ! инициализация объекта Resizer
Resizer.SetParentDefaults               !установить родителей для всех элементов
                                        !управления
Resizer.SetParentControl(?Button1,?OuterGroup)        !сменить родителя у
                                        !элемента управления
```

```
Resizer.SetParentControl(?Check1,?InnerGroup)    ! сменить родителя у
                                                 !элемента управления
Resizer.SetParentControl(?Check2,?InnerGroup)    ! сменить родителя у
                                                 !элемента управления
```

Смотри также: GetParentControl, Resize, SetParentDefaults

SetParentDefaults (set default parent controls)

SetParentDefaults, VIRTUAL

SetParentDefaults метод интеллектуально устанавливает соответствующего родителя для каждого элемента управления на окне. Метод Resize масштабирует элемент управления в зависимости от координат родителя.

Это позволяет изменять размеры конкретного элемента управления, основываясь на координатах связанного элемента управления, а не на координатах окна. Это применимо в тех случаях, когда стратегия, приложенная к родительскому элементу управления, приводит к тому, что его размеры меняются непропорционально размерам окна. Например, элемент управления в пределах структуры ГРУППА, размер которой зафиксирован, может быть высчитан относительно координат ГРУППЫ, а не окна.

Вы можете использовать SetParentControl метод для того, чтобы установить родителя для конкретного элемента управления.

SetParentDefaults метод является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод SetParentDefaults в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: SetParentDefaults метод учитывает координаты каждого элемента управления. Если координаты одного элемента управления находятся в пределах координат другого элемента управления, SetParentDefaults метод определяет “внешний” элемент управления как родитель для “внутреннего” элемента управления.

Метод Init вызывает метод SetParentDefaults тогда, когда стратегией изменения размеров является AppStrategy:Surface.

Пример:

```
window     WINDOW('Nested Controls'),AT(,,165,97),IMM,GRAY,MAX,RESIZE
GROUP('OuterGroup'),AT(5,3,154,92),USE(?OuterGroup),BOXED
BUTTON('Button 1'),AT(14,23),USE(?Button1)
ENTRY(@s20),AT(60,24),USE(Entry1)
GROUP('InnerGroup'),AT(11,49,141,38),USE(?InnerGroup),BOXED
CHECK('Check 1'),AT(32,64),USE(Check1)
CHECK('Check 2'),AT(91,64),USE(Check2)
.          .                    .
CODE
OPEN(window)
Resizer.Init(AppStrategy:Spread)                    ! инициализация объекта Resizer
Resizer.SetParentDefaults
                                                    !установить родителей для всех
                                                    !элементов управления
Resizer.SetParentControl(?Button1,?OuterGroup)      !сменить родителя у
                                                    !элемента управления
Resizer.SetParentControl(?Check1,?InnerGroup)       ! сменить родителя у
                                                    !элемента управления
Resizer.SetParentControl(?Check2,?InnerGroup)       ! сменить родителя у
                                                    !элемента управления
```

Смотри также: Resize, SetParentControl

SetStrategy (установить стратегию изменения размеров элемента управления)

SetStrategy(|[*элемент управления*] ,*стратегия размещения*, *стратегия изменения размеров*| **)** |*исходный элемент управления*, *целевой элемент управления*|

SetStrategy Устанавливает *стратегию размещения* и *стратегию изменения размеров* для применения к конкретному элементу управления.

Элемент управления Целочисленная константа, переменная, метка соответствия, или выражение, содержащее номер элемента управления. Если параметр опущен, SetStrategy метод применяет *стратегию размещения* и *стратегию изменения размеров* ко всем оконным элементам управления.

стратегия размещения Целочисленная константа, переменная, метка соответствия, или выражение, указывающее стратегию размещения, которую следует применить к *элементу управления*.

стратегия изменения размеров Целочисленная константа, переменная, метка соответствия, или выражение, указывающее стратегию изменения размеров, которую следует применить к *элементу управления*.

исходный элемент управления	Целочисленная константа, переменная, метка соответствия, или выражение, определяющее элемент управления, чьи *стратегия размещения* и *стратегия изменения размеров* применяются к *целевому элементу управления*.
целевой элемент управления	Целочисленная константа, переменная, метка соответствия, или выражение, определяющее элемент управления, чьи *стратегия размещения* и *стратегия изменения размеров* скопированы с *исходного элемента управления*.

SetStrategy метод устанавливает *стратегию размещения* и *стратегию изменения* для применения к конкретному элементу управления. Resize метод же применяет указанные стратегии.

Реализация: Метки соответствия для параметров *стратегия размещения* и *стратегия изменения размеров* объявлены в файле ABRESIZE.INC как показано ниже. Для применения двух или больше стратегий надо просто все их добавить.

Пример:

```
!стратегии изменения размеров
Resize:Resize EQUATE(0000b)              !масштабирует ширину и высоту
Resize:LockWidth EQUATE(0001b)           !фиксирует ширину
Resize:LockHeight EQUATE(0010b)          ! фиксирует высоту
Resize:LockSize EQUATE(0011b)            ! фиксирует ширину и высоту
Resize:ConstantRightEQUATE(0100b)        ! фиксирует положение правого края,
                                         !перемещает левый
Resize:ConstantBottom EQUATE(1000b)      ! фиксирует положение нижнего края,
                                         !перемещает вершину
          !стратегия переразмещения – горизонтальная позиция
Resize:Reposition EQUATE(0000h)          !перемасштабирует X и Y
Resize:LockXPos EQUATE(0001h)            ! фиксирует положение левого края
                                         !(полностью)
Resize:FixRight EQUATE(0002h)            ! фиксирует положение правого края
                                         !(относительно)
Resize:FixLeftEQUATE(0003h)              ! фиксирует положение левого края
                                         !(относительно)
Resize:FixXCenter EQUATE(0004h)          ! фиксирует горизонтальное положение
                                         !центра (относительно)
Resize:FixNearestX EQUATE(0005h)         !FixRight or FixLeft
          !стратегия переразмещения – вертикальная позиция
Resize:LockYPos EQUATE(0100h)            ! фиксирует положение верхнего края
                                         !(полностью)
```

```
Resize:FixBottom EQUATE(0200h)          ! фиксирует положение нижнего края
                                        !(относительно)
Resize:FixTop EQUATE(0300h)             ! фиксирует положение верхнего края
                                        !(относительно)
Resize:FixYCenter EQUATE(0400h)         ! фиксирует вертикальное положение
                                        !центра (относительно)
Resize:FixNearestY EQUATE(0500h)        !FixTop или FixBottom
```

Пример:

```
window     WINDOW('Client Information'),AT(,,185,100),IMM,GRAY,MAX,RESIZE
SHEET,AT(3,3,180,78),USE(?Sheet1)
TAB('Client List'),USE(?ListTab)
LIST,AT(10,20,165,55),USE(?List1),FROM(ClientQ),|
FORMAT('87L~Name~@s20@8L~State Code~@s2@')
END
TAB('Client Logo'),USE(?LogoTab)
IMAGE,AT(10,20,165,55),USE(?CLI:Logo)
            END
            END
PROMPT('Locate:'),AT(7,87),USE(?LocatorPrompt)
ENTRY(@s20),AT(33,86,61,12),USE(CLI:Name)
BUTTON('Close'),AT(150,84),USE(?Close)
            END
            CODE
            OPEN(window)
Resizer.Init(AppStrategy:Surface)       ! инициализация общей статегии для Resizer
Resizer.SetStrategy(?LocatorPrompt,  |
            !установить индивидуальную стратегию для элемента управления:
Resize:FixLeft+Resize:FixBottom,Resize:LockSize)              ! зафиксировать
                                        !положение левого нижнего края и
                                        !зафиксировать размер
            Resizer.SetStrategy(?CLI:Name,                    |
                                        ! установить индивидуальную стратегию
                                        !для элемента управления:

Resize:FixLeft+Resize:FixBottom,Resize:LockHeight)
                                        ! зафиксировать положение
                                        !левого нижнего края и зафиксировать
                                        !высоту
```

Смотри также: Resize

Классы Панели управления

Краткий обзор

Концепции Toolbar-класса

Классы Панели управления позволяют установить одну-единственную глобальную панель управления для всей программы, которая в дальнейшем и будет управлять целым рядом локальных объектов, таких, как формы модернизации, линейные окна списка, окна списка (окна списка с древовидным представлением отношений) и т.д. Причем Классы Панели управления обеспечивают при этом бесконфликтную работу, несмотря на то, что одна процедура вполне может управлять несколькими управляющимися и с панели управления объектами. При этом внутри многопроцессной программы могут быть активны сразу несколько таких процедур.

Суть всей концепции состоит в перехвате, т.е. в том, чтобы, как это и делают Классы Панели управления, "преобразовывать" событие, связанное с кнопкой панели управления в событие, связанное с определенным элементом управления или объектом.

Имеются два типа объектов панели управления, работающих совместно для обеспечения такой возможности: объекты Toolbar-класса и объекты ToolbarTarget-класса.

Объекты Toolbar-класса

Как правило, имеется единственный объект Toolbar-класса для каждой процедуры в вашей программе. Объект Toolbar-класса составляет список всех возможных адресатов или целей для событий панели управления. Когда происходит событие на панели управления, объект Toolbar-класса направляет событие для обработки к активному объекту ToolbarTarget-класса.

Объекты ToolbarTarget-класса

Каждый объект ToolbarTarget-класса связан с определенным объектом, таким как окно списка, окно дерева отношений или форма модернизации. В настоящее время имеется три класса, полученных из ToolbarTarget-класса: ToolbarListboxClass, ToolbarReltreeClass, и ToolbarUpdateClass.

Эти объекты ToolbarTarget-класса производят обработку события, уже конкретизированную для соответствующего объекта. Внутри процедуры может не быть вовсе, а может иметься и несколько объектов ToolbarTarget-класса; однако в конкретный момент времени *активен только один*. Активный объект ToolbarTarget-класса определяется текстом Вашей программы.

Связь с Другими Классами разработки приложений

WindowManager опционально использует Классы Панели управления, так же, как это делает Browse-класс. Поэтому, если ваша программа использует объект WindowManager'а или объект Browse-класса, то может также потребоваться применение и Классов Панели управления. Большинство этих действий происходит автоматически, стоит лишь включить заголовки WindowManager или Browse-класса (ABWINDOW.INC и ABBROWSE.INC) в секцию данных Вашей программы. См. *Концептуальный Пример*.

И WindowManager, и Browse-класс – оба они запрограммированы на использование объектов Класса Панели управления. Поэтому большая часть механизма взаимодействия между этими объектами скрыта внутри исходного кода Класса разработки приложений (ABC), и механика их взаимодействия отражена в произведенном шаблонами ABC коде по минимуму.

Реализация шаблона ABC

Такие ABC шаблоны, как шаблон BrowseBox'a, FormVCRControls'a и RelationTree, порождают объект Toolbar-класса, так называемый Toolbar, внутри каждой процедуры, содержащей шаблон, запрашивающий у панели управления глобальный управляющий сигнал. Каждый шаблон, который запрашивает глобальный управляющий сигнал панели управления, также порождает собственный объект ToolbarTarget-класса.

Объект ToolbarTarget-класса для шаблона FormVCRControls называется ToolBarForm; объект ToolbarTarget-класса для шаблона RelationTree называется REL#::Toolbar, где # представляет собой номер случая применения шаблона RelationTree; объект же ToolbarTarget-класса для шаблона BrowseBox'a полностью встроен внутрь объекта Browse-класса, а потому и не упоминается в ссылках, содержащихся внутри сгенерированного с помощью шаблонов текста.

Когда происходит событие на панели управления, сгенерированный с помощью шаблонов текст использует объект Toolbar-класса для того, чтобы вызвать обработчик события для активного объекта ToolbarTarget-класса

Исходные Файлы Классов Панели управления

Исходные Файлы Классов Панели управления по умолчанию установлены в директорию \CLARION4\LIBSRC. Собственно файлы Классов Панели управления и их соответствующие компоненты представлены ниже:

ABTOOLBA.INC	объявления Toolbar-класса
	объявления ToolbarTarget-класса
	объявления ToolbarListbox-класса
	объявления ToolbarReltree-класса
	объявления ToolbarUpdate-класса
ABTOOLBA.CLW	описания методов Toolbar-класса
	описания методов ToolbarTarget-класса
	описания методов ToolbarListbox-класса
	описания методов ToolbarReltree-класса
	описания методов ToolbarUpdate-класса

Пояснительный Пример

Следующий пример показывает типовую последовательность операторов, предназначенных для того, чтобы объявлять, порождать, инициализировать, использовать и завершать ToolbarClass и связанные с ним объекты ToolbarTarget-класса.

В этом примере Классы Панели управления применены для того, чтобы обеспечить управление двумя разными окнами списка, связанными между собой и принадлежащими одной MDI процедуре, с одной общей глобальной панели управления. При этом в первом окне списка отображается информация по клиенту, а в подчиненном, связанном окне списка показывается список телефонных номеров, относящихся к выбранному клиенту. Панель же управления работает в любом случае, какое бы окно списка ни владело в данный момент фокусом ввода.

Программа, используя свойство SYSTEM{Prop:Active}, посылает активному MDI окну произошедшее на панели управления событие. Местный же объект Toolbar-класса, в свою очередь, вызывает для обработки пришедшего события активный объект ToolbarTarget-класса.

```
PROGRAM
INCLUDE('ABBROWSE.INC')          ! объявить BrowseClass
INCLUDE('ABTOOLBA.INC')          ! объявить Классы Панели управления
INCLUDE('ABWINDOW.INC')          ! объявить WindowManager
CODE                             !текст программы
```

```
Main        PROCEDURE                   !содержит глобальную панель управления
AppFrame  APPLICATION('Toolbars'),AT(,,275,175),SYSTEM,MAX,RESIZE,IMM
MENUBAR
ITEM('Browse Customers'),USE(?BrowseCustomer)
END
TOOLBAR,AT(0,0,400,22)                  !must use ABTOOLBA.INC EQUATES:

BUTTON,AT(4,2),USE(?Top,Toolbar:Top),DISABLE,ICON('VCRFIRST.ICO'),FLAT
BUTTON,AT(16,2),USE(?PageUp,Toolbar:PageUp),DISABLE,ICON('VCRPRIOR.ICO'),FLAT
BUTTON,AT(28,2),USE(?Up,Toolbar:Up),DISABLE,ICON('VCRUP.ICO'),FLAT
BUTTON,AT(40,2),USE(?Down,Toolbar:Down),DISABLE,ICON('VCRDOWN.ICO'),FLAT
BUTTON,AT(52,2),USE(?PageDown,Toolbar:PageDown),DISABLE,ICON('VCRNEXT.ICO'),FLAT
BUTTON,AT(64,2),USE(?Bottom,Toolbar:Bottom),DISABLE,ICON('VCRLAST.ICO'),FLAT
END
END
Frame       CLASS(WindowManager)
Init
PROCEDURE(),BYTE,PROC,VIRTUAL
TakeAccepted   PROCEDURE(),BYTE,PROC,VIRTUAL
END
Toolbar     ToolbarClass                    ! объявить объект Toolbar
            CODE
            Frame.Run()
Frame.Init  PROCEDURE()
ReturnValue                             BYTE,AUTO
            CODE
            ReturnValue = PARENT.Init()
            SELF.VCRRequest &= VCRRequest
            SELF.Errors &= GlobalErrors
            SELF.AddItem(Toolbar)           !зарегистрировать Toolbar для
                                            !WindowManager
            OPEN(AppFrame)
            SELF.Opened=True
            SELF.SetAlerts()
            RETURN ReturnValue
Frame.TakeAccepted PROCEDURE()
ReturnValue BYTE,AUTO
Looped      BYTE
            CODE
            LOOP
IF Looped THEN RETURN Level:Notify ELSE Looped=1.
            CASE ACCEPTED()
            OF Toolbar:First TO Toolbar:Last        !при возникновении события
```

```
EVENT:Accepted на панели управления
POST(EVENT:Accepted,ACCEPTED(),SYSTEM{Prop:Active})
                                        ! переслать его в активный процесс
            CYCLE                       ! и остановиться
            END
            ReturnValue = PARENT.TakeAccepted()
            IF ACCEPTED() = ?BrowseCustomer
            START(BrowseCustomer,050000)
            END
            RETURN ReturnValue
            END
BrowseCustomer PROCEDURE            !содержит местный Toolbar и ряд целей
CusView     VIEW(Customer)
            END
CusQ        QUEUE
CUS:CUSTNO      LIKE(CUS:CUSTNO)
CUS:NAME        LIKE(CUS:NAME)
ViewPosition    STRING(512)
END
PhView      VIEW(Phones)
END
PhQ         QUEUE
PH:NUMBER           LIKE(PH:NUMBER)
PH:ID       LIKE(PH:ID)
ViewPosition        STRING(512)
END         CusWindow  WINDOW('Browse
Customers'),AT(,,246,131),IMM,SYSTEM,GRAY,MDI
LIST,AT(8,7,160,100),USE(?CusList),IMM,HVSCROLL,FROM(CusQ),|
FORMAT('51R(2)|M~CUSTNO~C(0)@n-14@80L(2)|M~NAME~@s30@')
BUTTON('&Insert'),AT(17,111,45,14),USE(?InsertCus),SKIP
BUTTON('&Change'),AT(66,111,45,14),USE(?ChangeCus),SKIP,DEFAULT
BUTTON('&Delete'),AT(115,111,45,14),USE(?DeleteCus),SKIP
LIST,AT(176,7,65,100),USE(?PhList),IMM,FROM(PhQ),|
FORMAT('80L~Phones~L(1)@s20@')
BUTTON('&Insert'),AT(187,41,42,12),USE(?InsertPh),HIDE
BUTTON('&Change'),AT(187,54,42,12),USE(?ChangePh),HIDE
BUTTON('&Delete'),AT(187,67,42,12),USE(?DeletePh),HIDE
END
ThisWindow  CLASS(WindowManager)                ! объявить объект ThisWindow
Init
PROCEDURE(),BYTE,PROC,VIRTUAL
Kill
PROCEDURE(),BYTE,PROC,VIRTUAL
```

```
TakeSelected PROCEDURE(),BYTE,PROC,VIRTUAL
             END
Toolbar      ToolbarClass                ! объявить принимающий объект Toolbar
                                         ! и обработать событие на панели управления,
                                         !пришедшее из процедуры Main
CusBrowse CLASS(BrowseClass)             ! объявить объект CusBrowse
Q            &CusQ
             END
PhBrowse  CLASS(BrowseClass)             ! объявить объект PhBrowse
Q            &PhQ
             END
             CODE
             ThisWindow.Run()
ThisWindow.Init    PROCEDURE()
ReturnValue        BYTE,AUTO
             CODE
             ReturnValue = PARENT.Init()
             SELF.FirstField = ?CusList        !CusList получает начальный фокус
 SELF.VCRRequest &= VCRRequest
 SELF.Errors &= GlobalErrors
 SELF.AddItem(Toolbar)                       !регистрация Toolbar для WindowManager
 Relate:Customer.Open
  CusBrowse.Init(?CusList,CusQ.ViewPosition,CusView,CusQ,Relate:Customer,SELF)
  PhBrowse.Init(?PhList,PhQ.ViewPosition,PhView,PhQ,Relate:Phones,SELF)
 OPEN(CusWindow)
 SELF.Opened=True
 CusBrowse.Q &= CusQ
 CusBrowse.AddSortOrder(,CUS:BYNUMBER)
 CusBrowse.AddField(CUS:CUSTNO,CusBrowse.Q.CUS:CUSTNO)
 CusBrowse.AddField(CUS:NAME,CusBrowse.Q.CUS:NAME)
 PhBrowse.Q &= PhQ
 PhBrowse.AddSortOrder(,PH:IDKEY)
 PhBrowse.AddRange(PH:ID,Relate:Phones,Relate:Customer)
 PhBrowse.AddField(PH:NUMBER,PhBrowse.Q.PH:NUMBER)
 PhBrowse.AddField(PH:ID,PhBrowse.Q.PH:ID)
 CusBrowse.InsertControl=?InsertCus
 CusBrowse.ChangeControl=?ChangeCus
 CusBrowse.DeleteControl=?DeleteCus
 CusBrowse.AddToolbarTarget(Toolbar)          !нацелить панель управления на
                                              !работу с CusBrowse
 PhBrowse.InsertControl=?InsertPh
 PhBrowse.ChangeControl=?ChangePh
 PhBrowse.DeleteControl=?DeletePh
```

```
  PhBrowse.AddToolbarTarget(Toolbar)          ! нацелить панель управления на
                                              !работу с PhBrowse
  SELF.SetAlerts()
  RETURN ReturnValue
ThisWindow.Kill   PROCEDURE()
ReturnValue       BYTE,AUTO
  CODE
  ReturnValue = PARENT.Kill()
  Relate:Customer.Close
  RETURN ReturnValue
ThisWindow.TakeSelected    PROCEDURE()
ReturnValue
BYTE,AUTO
Looped
BYTE
  CODE
  LOOP
    IF Looped THEN RETURN Level:Notify ELSE Looped=1.
    ReturnValue = PARENT.TakeSelected()
    CASE FIELD()
    OF ?CusList
!при выборе этого варианта,
      Toolbar.SetTarget(?CusList)
! сделать текущей целью ?CusList
    OF ?PhList
! при выборе этого варианта
      IF RECORDS(PhBrowse.Q) > 1
                !и наличия в очереди более одной записи,
        Toolbar.SetTarget(?PhList)
! сделать текущей целью ?PhList
      END
    END
    RETURN ReturnValue
  END
```

Методы Toolbar-класса

Toolbar-класс содержит методы, описанные ниже.

Функциональная Организация — Предполагаемое Использование

Для лучшего понимания Toolbar-класса будет полезно организовать его различные методы в две больших категории согласно их предполагаемому использованию — первичный интерфейс и виртуальные методы. Такая организация, по мнению разработчиков, наиболее полно отражает типовое использование методов Toolbar-класса.

Первичные Интерфейсные Методы

Первичные интерфейсные методы, вызов которых, скорее всего, будет иметь место в Вашей программе, могут быть далее разделены на три категории:

Одноразовое применение:

Init[V]	инициализировать объект Toolbar-класса
AddTarget	зарегистрировать объект, управляемый с панели управления
Kill[V]	завершить объект Toolbar-класса

Используются в основном:

SetTarget	установить активную цель & соответствующее состояние панели управления
TakeEvent[V]	обработать событие на панели управления для активной цели

Используются изредка:

DisplayButtons[V]	сделать недоступными для выбора соответствующие кнопки панели управления

[V] Эти методы также являются виртуальными.

Виртуальные Методы

В основном Вам не придется прямо вызывать эти методы, поскольку они вызываются другими основными методами. Однако у Вас наверняка часто будет возникать потребность в переопределении этих методов, а поскольку они являются виртуальными, то сделать это достаточно просто. Эти методы уже по умолчанию обеспечивают «разумное» поведение, в случае, если Вы не хотите переопределить их

DisplayButtons	сделать доступными для выбора соответствующие кнопки панели управления
TakeEvent	обработать событие на панели управления для активной цели
Kill	закрыть объект Toolbar-класса

AddTarget (зарегистрировать объект, управляемый с панели управления)

AddTarget(*Цель, элемент управления* **)**

AddTarget	Д обавляет очередную цель к списку потенциальных целей панели управления, которым владеет объект Toolbar-класса.
Цель	Метка объекта ToolbarTarget-класса.
Элемент управления	Целочисленная константа, переменная, метка соответствия, или выражение, содержащее идентификационный номер *цели*. Для целей, связанных с элементом управления, он представляет собой номер элемента управления (обычно совпадает с меткой соответствия поля).

AddTarget метод добавляет очередную цель для панели управления (объект ToolbarTarget-класса) к списку потенциальных целей панели управления, которым владеет объект Toolbar-класса.

Последняя добавленная цель является активной до тех пор, пока она не будет вытеснена последующим запросом к AddTarget или SetTarget.

Пример:

```
CODE
Toolbar.Init                    ! инициализация объекта Toolbar
ToolBar.AddTarget( ToolBarForm, -1 )
                                !регистрация в качестве цели Формы
                                !Модернизации
Toolbar.AddTarget( REL1::Toolbar, ?RelTree )
                                ! регистрация в качестве цели Дерева
                                !отношений
BRW1.AddToolbarTarget( Toolbar )
                                ! регистрация в качестве цели Окна просмотра
...
                                ! Метод BrowseClass'а вызывает AddTarget
```

Смотри также: SetTarget

DisplayButtons (сделать доступными для выбора соответствующие кнопки панели управления)

DisplayButtons, VIRTUAL

DisplayButtons метод делает соответствующие кнопки панели управления доступными или нет для выбора в зависимости от активной цели.

SetTarget метод устанавливает активную цель панели управления.

Метод DisplayButtons является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод DisplayButtons в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: DisplayButtons метод вызывает для активной цели панели управления метод ToolbarTargetClass.DisplayButtons.

Пример:

```
CODE
Toolbar.Init                          ! инициализация объекта Toolbar
ToolBar.AddTarget( ToolBarForm, -1 )  !зарегистрировать в качестве цели Форму
                                      !Модернизации
Toolbar.DisplayButtons                !и сделать недоступными для выбора
                                      !соответствующие кнопки панели управления
                                      !для текущей цели
```

Смотри также: SetTarget

Init (инициализация объекта Toolbar-класса)

Init

Init метод инициализирует объект Toolbar-класса.

Реализация: Init метод размещает новый список потенциальных целей панели управления.

Пример:

```
CODE
Toolbar.Init                          ! инициализация объекта Toolbar
                                      ! текст программы
ACCEPT                                !текст программы
        END
        Toolbar.Kill                  !закрытие объекта Toolbar
```

Kill (закрытие объекта Toolbar-класса)

Kill, VIRTUAL

Kill метод освобождает любую память, размещенную в течение жизни объекта, а также исполняет любой другой требуемый код завершения.

Метод Kill является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод Kill в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: В процессе своей работы Kill метод распоряжается всем списком потенциальных целей панели управления.

Пример:

```
CODE
Toolbar.Init                    ! инициализация объекта Toolbar
                                ! текст программы
        ACCEPT                  !текст программы
        END
        Toolbar.Kill            !закрытие объекта Toolbar
```

SetTarget (установить активную цель)

SetTarget([*идентификатор*] **)**

SetTarget Устанавливает на панели управления активную цель объекта Toolbar-класса.

идентификатор Целочисленная константа, переменная, метка соответствия, или выражение, содержащее идентификационный номер *цели*. Для целей, связанных с элементом управления, он представляет собой номер элемента управления (обычно совпадает с меткой соответствия поля). Если параметр опущен или равен нулю (0), SetTarget устанавливает наиболее подходящую (близкую по значению *идентификатора*) цель.

SetTarget метод устанавливает на панели управления активную цель объекта Toolbar-класса (объект ToolbarTarget-класса) и приводит состояние панели управления в соответствие с выбранным в качестве цели объектом.

Реализация: SetTarget метод вызывает метод ToolbarTargetClass.TakeToolbar или ToolbarTargetClass.TryTakeToolbar для переопределения атрибута TIP каждой кнопки панели управления и приведения статуса «доступно / недоступно для выбора» в соответствие с активной целью панели управления.

Пример:

```
ACCEPT
CASE EVENT()
OF EVENT:OpenWindow             ! при открытии окна
DO RefreshWindow                ! загрузить очереди данных для просмтра
OF EVENT:Accepted               ! при возникновении событий EVENT:Accepted
 CASE FOCUS()                   ! также пересланы сюда с глобальной панели
```

```
управления)
OF ?ClientList                          ! обработать объект, владеющий фокусом
                                        !ввода,
Toolbar.SetTarget(?ClientList)
OF ?PhoneList                           ! а также сделать доступными
                                        !соответствующие кнопки
Toolbar.SetTarget(?PhoneList)           ! панели управления и перенастроить
                                        !атрибуты TIP
END
Toolbar.TakeEvent(VCRRequest, WM)       ! объект Toolbar вызывает обработчик событий
END                                     ! активной в данный момент цели
END                                     ! который уже и осуществляет действия
                                        ! (прокрутка списка, вставка, удаление,
                                        ! помощь и т.д.)
                                        ! Зачастую обработчик событий просто
                                        ! посылает другое событие к конкретному
                                        ! элементу управления, например,
                                        ! событие Event:Accepted – кнопке ?Insert,
                                        ! а событие Event:PageUp – окну
                                        ! списка ?ClientList
```

Смотри также: ToolbarTargetClass.TakeToolbar, ToolbarTargetClass.TryTakeToolbar

TakeEvent (обработать событие панели управления)

TakeEvent([*vcr*], *Менеджер окна* **), VIRTUAL**

TakeEvent Обрабатывает события на панели управления для активной цели панели управления.

vcr Целочисленная переменная, предназначенная для получения номера нажатой кнопки навигации *vcr*-панели (панель, которая для навигации по просматриваемым в окне записям использует кнопки, по своему виду и функционированию схожие с аналогичными кнопками на панели управления видеомагнитофона). В результате этого метод TakeEvent определяет необходимое последующее действие. Если этот параметр пропущен, объект ToolbarTarget-класса не осуществляет никакой навигации типа «послать обработку».

Менеджер окна Метка WindowManager-объекта для объекта ToolbarTarget-класса. См. *Менеджер Окна* для получения дополнительной информации.

TakeEvent метод обрабатывает события на панели управления для активной цели панели управления (объекта ToolbarTarget-класса).

Параметр *vcr* позволяет методу TakeEvent определить необходимое последующее или послеследующее действие. Наример, метод ToolbarUpdateClass.TakeEvent (для Формы Модернизации), может интерпретировать кнопку прокрутки вниз видеомагнитофона как "сохранить и затем прокрутить". Он, во-первых, производит необходимые действия по сохранению текущей позиции и, во-вторых, «прокручивает» список в направлении, определенном параметром *vcr*.

Метод SetTarget устанавливает активную цель панели управления.

Метод TakeEvent является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeEvent в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод WindowManager.TakeEvent вызывает метод TakeEvent. Метод же TakeEvent, в свою очередь, вызывает метод ToolbarTargetClass.TakeEvent для активной цели панели управления.

Пример:

```
MyWindowManager.TakeAccepted PROCEDURE
 CODE
 IF ~SELF.Toolbar &= NULL
  SELF.Toolbar.TakeEvent(SELF.VCRRequest,SELF)
 END
!текст процедуры
```

Смотри также: SetTarget, WindowManager.TakeEvent

Свойства ToolbarTarget-класса

Хотя ToolbarTarget-класс полезен и сам по себе, из него могут быть получены и другие более полезные классы, такие, например, как ToolbarListboxClass, ToolbarRelTreeClass или ToolbarUpdateClass, а также другие структуры, типа Toolbar-класса, использующие ToolbarTarget-класс для ссылки на любой из этих полученных классов.

Классы, полученные из ToolbarTarget-класса, позволяют устанавливать состояние глобальной панели управления в соответствии с тем объектом, которым она сейчас управляет, а затем уже обрабатывать глобальные события панели управления, посланные этому объекту.

ToolbarTarget-класс содержит следующие свойства:

ChangeButton (номер элемента управления «Изменить»)

ChangeButton SIGNED

ChangeButton свойство содержит номер оконного элемента управления (обычно представляемый меткой соответствия этого элемента управления), который вызывает действие по изменению записи для этого объекта ToolbarTarget-класса.

Значение ноль (0) делает кнопку «Изменить» на глобальной панели управления недоступной для выбора.

Реализация: Объект ToolbarTarget-класса использует это свойство для того, чтобы сделать кнопку «Изменить» на глобальной панели управления доступной или недоступной для выбора, а также как целевой элемент управления при посылке некоторых событий. См. POST в *Описании Языка* для получения дополнительной информации. Когда конечный пользователь нажимает на кнопку «Изменить» на глобальной панели управления, объект ToolbarTarget-класса посылает элементу управления ChangeButton событие EVENT:Accepted.

Control (оконный элемент управления)

Control SIGNED

Свойство **Control** содержит номер оконного элемента управления (обычно представляемый меткой соответствия этого элемента управления), который связан с этим объектом ToolbarTarget-класса. Для того же объекта ToolbarTarget-класса, который не имеет ни одного связанного элемента управления (Форма модернизации), свойство Control может содержать любой идентификационный номер.

Объект ToolbarTarget-класса использует это свойство в качестве целевого элемента управления при посылке некоторых событий. См. POST в *Описании Языка*.

Метод ToolbarClass.AddTarget устанавливает значение этого свойства.

Реализация: В соответствии с принятым соглашением, формы модернизации имеют значение свойства Control, равное отрицательной единице (-1).

Смотри также: ToolbarClass.AddTarget

DeleteButton (номер элемента управления «Удалить»)

DeleteButton SIGNED

DeleteButton свойство содержит номер оконного элемента управления (обычно представляемый меткой соответствия этого элемента управления), который вызывает действие по удалению записи для этого объекта ToolbarTarget-класса.

Значение ноль (0) делает кнопку «Удалить» на глобальной панели управления недоступной для выбора.

Реализация: Объект ToolbarTarget-класса использует это свойство для того, чтобы сделать кнопку «Удалить» на глобальной панели управления доступной или недоступной для выбора, а также как целевой элемент управления при посылке некоторых событий. См. POST в *Описании Языка* для получения дополнительной информации. Когда конечный пользователь нажимает на кнопку «Удалить» на глобальной панели управления, объект ToolbarTarget-класса посылает элементу управления DeleteButton событие EVENT:Accepted.

HelpButton (номер элемента управления «Помощь»)

HelpButton SIGNED

Свойство **HelpButton** содержит номер оконного элемента управления (обычно представляемый меткой соответствия этого элемента управления), который вызывает справочную систему Windows для этого объекта ToolbarTarget-класса.

Значение ноль (0) делает кнопку «Помощь» на глобальной панели управления недоступной для выбора.

Реализация: Объект ToolbarTarget-класса использует это свойство для того, чтобы сделать кнопку «Помощь» на глобальной панели управления доступной или недоступной для выбора. Когда конечный пользователь нажимает на кнопку «Помощь» на глобальной панели управления, объект ToolbarTarget-класса «нажимает» на кнопку вызова помощи (F1).

InsertButton (номер элемента управления «Добавить»)

InsertButton SIGNED

InsertButton свойство содержит номер оконного элемента управления (обычно представляемый меткой соответствия этого элемента управления), который вызывает добавление записи для этого объекта ToolbarTarget-класса.

Значение ноль (0) делает кнопку «Добавить» на глобальной панели управления недоступной для выбора.

Реализация: Объект ToolbarTarget-класса использует это свойство для того, чтобы сделать кнопку «Добавить» на глобальной панели управления доступной или недоступной для выбора, а также как целевой элемент управления при посылке некоторых событий. См. POST в *Описании Языка* для получения дополнительной информации. Когда конечный пользователь нажимает на кнопку «Добавить» на глобальной панели управления, объект ToolbarTarget-класса посылает элементу управления InsertButton событие EVENT:Accepted.

SelectButton (номер элемента управления «Выбрать»)

SelectButtonSIGNED

InsertButton свойство содержит номер оконного элемента управления (обычно представляемый меткой соответствия этого элемента управления), который вызывает действие по выбору записи для этого объекта ToolbarTarget-класса.

Значение ноль (0) делает кнопку «Выбрать» на глобальной панели управления недоступной для выбора.

Реализация: Объект ToolbarTarget-класса использует это свойство для того, чтобы сделать кнопку «Выбрать» на глобальной панели управления доступной или недоступной для выбора, а также как целевой элемент управления при посылке некоторых событий. См. POST в *Описании Языка* для получения дополнительной информации. Когда конечный пользователь нажимает на кнопку «Выбрать» на глобальной панели управления, объект ToolbarTarget-класса посылает элементу управления SelectButton событие EVENT:Accepted.

Методы ToolbarTarget-класса

Хотя ToolbarTarget-класс полезен и сам по себе, из него могут быть получены и другие более полезные классы, такие, например, как ToolbarListboxClass, ToolbarRelTreeClass или ToolbarUpdateClass, а также другие структуры, типа Toolbar-класса, использующие ToolbarTarget-класс для ссылки на любой из этих полученных классов.

Классы, полученные из ToolbarTarget-класса, позволяют устанавливать состояние глобальной панели управления в соответствии с тем объектом, которым она сейчас управляет, затем обрабатывать глобальные события панели управления, посланные этому объекту.

ToolbarTarget-класс содержит следующие методы:

Функциональная Организация — Предполагаемое Использование

Для лучшего понимания Toolbar-класса будет полезно признать все его методы виртуальными Такая организация, по мнению разработчиков, наиболее полно отражает типовое использование методов Toolbar-класса. В основном Вам не придется прямо вызывать эти методы, поскольку они вызываются другими методами Toolbar-класса. Однако у Вас наверняка часто будет возникать потребность в переопределении этих методов, а поскольку они являются виртуальными, то сделать это достаточно просто. Эти методы уже по умолчанию обеспечивают «разумное» поведение, в случае, если Вы не хотите переопределить их.

Виртуальные Методы

DisplayButtons	сделать доступными для выбора соответствующие кнопки панели управления
TryTakeToolbar	восстановить индикатор панели управления
TakeToolbar	взять управление
TakeEvent	обработать события на панели управления

DisplayButtons (сделать доступными для выбора соответствующие кнопки панели управления)

DisplayButtons, VIRTUAL

DisplayButtons метод делает доступными или недоступными для объекта ToolbarTarget-класса соответствующие кнопки глобальной панели управления, в зависимости от значений таких свойств, как HelpButton, InsertButton, ChangeButton, DeleteButton или SelectButton.

Метод DisplayButtons является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод DisplayButtons в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Методы ToolbarListboxClass.TakeToolbar, ToolbarRelTreeClass.TakeToolbar и ToolbarUpdateClass.TakeToolbar вызывают метод DisplayButtons.

Пример:

```
MyToolbarListboxClass.DisplayButtons PROCEDURE
CODE
```

```
PARENT.DisplayButtons                !Вызов метода DisplayButtons
!здесь исходный код Вашей программы
```

Смотри также: HelpButton, InsertButton, ChangeButton, DeleteButton, SelectButton, ToolbarListboxClass.TakeToolbar, ToolbarRelTreeClass.TakeToolbar, ToolbarUpdateClass.TakeToolbar

TakeEvent (обработать событие панели управления)

TakeEvent([*vcr*], *Менеджер_окна* **), VIRTUAL**

TakeEvent	Обрабатывает событие, произошедшее на глобальной панели управления, применительно к текущей цели.
vcr	Целочисленная переменная, предназначенная для получения номера нажатой навигационной кнопки *vcr*-панели. Если этот параметр пропущен, объект ToolbarTarget-класса не осуществляет никакой навигации типа «послать обработку».
Менеджер_окна	Метка WindowManager-объекта для объекта ToolbarTarget-класса. См. *Менеджер Окна* для получения дополнительной информации.

TakeEvent метод обрабатывает события на панели управления для этой цели панели управления.

Параметр *vcr* позволяет методу TakeEvent определить необходимое последующее или послеследующее действие. Наример, метод ToolbarUpdateClass.TakeEvent (для Формы Модернизации), может интерпретировать кнопку прокрутки вниз видеомагнитофона как “сохранить и затем прокрутить”. Он, во-первых, производит необходимые действия по сохранению текущей позиции и, во-вторых, «прокручивает» список в направлении, определенном параметром *vcr*.

Метод TakeEvent является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeEvent в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод ToolbarClass.TakeEvent вызывает метод TakeEvent для активного объекта ToolbarTarget-класса. Метод же ToolbarClass.SetTarget определяет активный объект ToolbarTarget-класса. А метод TakeEvent посылает соответствующему местному элементу управления событие EVENT:Accepted (чтобы добавить, изменить, удалить запись или вызвать помощь).

Пример:

```
REL1::Toolbar.TakeEvent PROCEDURE(<*LONG VCR>,WindowManager WM)
 CODE
 CASE ACCEPTED()
 OF Toolbar:Bottom TO Toolbar:Up
  SELF.Control{PROPLIST:MouseDownRow} = CHOICE(SELF.Control)
  EXECUTE(ACCEPTED()-Toolbar:Bottom+1)
   DO REL1::NextParent
   DO REL1::PreviousParent
   DO REL1::NextLevel
   DO REL1::PreviousLevel
   DO REL1::NextRecord
   DO REL1::PreviousRecord
  END
 OF Toolbar:Insert TO Toolbar:Delete
  SELF.Control{PROPLIST:MouseDownRow} = CHOICE(SELF.Control)
  EXECUTE(ACCEPTED()-Toolbar:Insert+1)
   DO REL1::AddEntry
   DO REL1::EditEntry
   DO REL1::RemoveEntry
  END
 ELSE
  PARENT.TakeEvent(VCR,ThisWindow)
 END
```

Смотри также: ToolbarClass.SetTarget, ToolbarClass.TakeEvent

TakeToolbar (принять управление панелью управления)

TakeToolbar, VIRTUAL

TakeToolbar метод предназначен для приведения состояния глобальной панели управления в соответствие с текущим объектом ToolbarTarget-класса, что подразумевает перенастройку значений атрибутов MSG и TIP, запрещение или разрешение соответствующих кнопок и т.д.

Метод TakeToolbar является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeToolbar в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: The TakeToolbar method is a placeholder method for derived classes.

Реализация: Метод TakeToolbar - placeholder метод для полученных классов.

Пример:

```
MyToolbarReltreeClass.TakeToolbar PROCEDURE          !работа полученного класса
        CODE
        SELF.DisplayButtons                          !сделать доступными для выбора
!здесь – текст Вашей программы
```

Смотри также: ToolbarListboxClass.TakeToolbar, ToolbarRelTreeClass.TakeToolbar, ToolbarUpdateClass.TakeToolbar

TryTakeToolbar (возвратить индикатор контроля над панелью управления)

TryTakeToolbar, VIRTUAL

TryTakeToolbar метод - это виртуальный placeholder метод, предназначенный для возврата значения, сигнализирующего о том, успешно ли объект ToolbarTarget-класса принял контроль над панелью управления. Возвращаемое значение единица (1 или Истина) означает успех; значение ноль (0 или Ложь) говорит о неудачной попытке взять под свой контроль панель управления.

Метод TryTakeToolbar является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TryTakeToolbar в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод TryTakeToolbar - placeholder метод для полученных классов.

Возвращаемый тип данных: BYTE

Пример:

```
MyToolbarListBoxClass.TryTakeToolbar PROCEDURE     ! работа полученного класса
 CODE
  IF SELF.Browse.ListControl{PROP:Visible}
   SELF.TakeToolbar
   RETURN 1                                        !управление получено
  ELSE
   RETURN 0                                        ! управление не получено
      END
```

Смотри также: ToolbarListboxClass.TryTakeToolbar, ToolbarUpdateClass.TryTakeToolbar

Свойства ToolbarListbox-класса

ToolbarListbox-класс наследует все свойства ToolbarTarget-класса, из которого он получен. См. *Свойства ToolbarTarget-класса* и *Концепции Toolbar-класса* для получения дополнительной информации.

В дополнение к унаследованным, ToolbarListbox-класс содержит следующие свойства.

Browse (Объект Browse-класса)

Browse	&BrowseClass

Свойство **Browse** представляет собой ссылку на объект Browse-класса объекта ToolbarListbox-класса. Объект ToolbarListbox-класса использует это свойство для получения доступа к свойствам и методам объекта Browse-класса.

Реализация: Метод BrowseClass.AddToolbarTarget устанавливает значение свойства Browse.

Метод TryTakeToolbar использует свойство Browse для того, чтобы определить, является ли соответствующий элемент управления окно списка видимым.

Методы ToolbarListbox-класса

ToolbarListbox-класс наследует все методы ToolbarTarget-класса, из которого он получен. См. *Методы ToolbarTarget-класса* и *Концепции Toolbar-класса* для получения дополнительной информации.

В дополнение к унаследованным, ToolbarListbox-класс содержит следующие методы.

DisplayButtons (сделать доступными для выбора соответствующие кнопки панели управления)

DisplayButtons, VIRTUAL

Метод **DisplayButtons** делает доступными или недоступными для объекта ToolbarListbox-класса соответствующие кнопки глобальной панели управления, в зависимости от значений таких свойств, как HelpButton, InsertButton, ChangeButton, DeleteButton или SelectButton.

Метод DisplayButtons является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод DisplayButtons в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод TakeToolbar вызывает метод DisplayButtons.

Пример:

```
CODE
Toolbar.Init                              !инициализация объекта Toolbar
 BRW1.AddToolbarTarget( Toolbar )         !регистрация цели BrowseBox
 ToolBar.SetTarget( ?Browse:1)            !вызывает DisplayButtons путем
                                          !использования TakeToolbar
MyToolbarListboxClass.DisplayButtons PROCEDURE
                                          !работа полученного класса
CODE
DISABLE(Toolbar:History)                  !сделать недоступной для выбора как панель,
                                          !так и кнопку
ENABLE(Toolbar:Locate)                    ! сделать доступной для выбора кнопку
                                          !локатора
PARENT.DisplayButtons                     ! Вызов DisplayButtons
!здесь исходный код Вашей программы
```

Смотри также: HelpButton, InsertButton, ChangeButton, DeleteButton, SelectButton, TakeToolbar

TakeEvent (обработать событие на панели управления)

TakeEvent([*vcr*], *Менеджер_окна* **), VIRTUAL**

TakeEvent	Обрабатывает событие, произошедшее на глобальной панели управления, применительно к объекту ToolbarListbox-класса.
vcr	Целочисленная переменная, предназначенная для получения номера нажатой навигационной кнопки *vcr*-панели. Это позволяет методу TakeEvent определить соответствующее последующее действие. Если этот параметр пропущен, объект ToolbarTarget-класса не осуществляет никакой навигации типа «послать обработку».
Менеджер_окна	Метка WindowManager-объекта для объекта ToolbarListbox-класса. См. *Менеджер Окна* для получения дополнительной информации.

TakeEvent метод обрабатывает для объекта ToolbarListbox-класса события, произошедшие на глобальной панели управления.

Параметр *vcr* позволяет методу TakeEvent определить необходимое последующее или послеследующее действие. Наример, метод ToolbarListboxClass.TakeEvent, может интерпретировать кнопку прокрутки вниз видеомагнитофона как "сохранить и затем прокрутить". Он, во-первых, производит необходимые действия по сохранению текущей позиции и, во-вторых, «прокручивает» список в направлении, определенном параметром *vcr*.

Метод TakeEvent является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeEvent в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод ToolbarClass.TakeEvent вызывает метод TakeEvent для активного объекта ToolbarTarget-класса. Метод же ToolbarClass.SetTarget определяет активный объект ToolbarTarget-класса.

Пример:

```
ToolbarClass.TakeEvent PROCEDURE(<*LONG VCR>,WindowManager WM)
 CODE
ASSERT(~SELF.List &= NULL)
IF RECORDS(SELF.List)
SELF.List.Item.TakeEvent(VCR,WM)
END
```

Смотри также: ToolbarClass.SetTarget, ToolbarClass.TakeEvent

TakeToolbar (принять управление панелью управления)

TakeToolbar, VIRTUAL

TakeToolbar метод определяет состояние глобальной панели управления, приводя его в соответствие с объектом ToolbarListbox-класса.

Метод TakeToolbar является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeToolbar в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод TakeToolbarустанавливает для кнопок глобальной панели управления соответствующие значения атрибута, а также делает доступными или нет для выбора кнопки панели управления в зависимости от соответствующего

объекта ToolbarListbox-класса. ToolbarClass.SetTarget метод и TryTakeToolbar метод вызывают TakeToolbar метод.

Пример:

```
MyToolbarClass.SetTarget PROCEDURE(SIGNED Id)
I USHORT,AUTO
Hit USHORT
 CODE
  ASSERT(~ (SELF.List &= NULL))
  IF Id                                  ! определить явно запрашиваемую цель
SELF.List.Id = Id
    GET(SELF.List,SELF.List.Id)
    ASSERT (~ERRORCODE())
    SELF.List.Item.TakeToolbar
  ELSE                                   !определить какую-нибудь (последнюю)
                                         !пригодную цель
   LOOP I = 1 TO RECORDS(SELF.List)
    GET(SELF.List,I)
    IF SELF.List.Item.TryTakeToolbar() THEN Hit = I.
   END
   IF Hit THEN GET(SELF.List,Hit).
  END
```

Смотри также: TryTakeToolbar, ToolbarClass.SetTarget

TryTakeToolbar (возвратить индикатор контроля над панелью управления)

TryTakeToolbar, VIRTUAL

TryTakeToolbar метод возвращает значение, сигнализирующее о том, успешно ли объект ToolbarTarget-класа принял контроль над панелью управления. Возвращаемое значение единица (1 или Истина) означает успех; значение ноль (0 или Ложь) говорит о неудачной попытке взять под свой контроль панель управления.

Метод TryTakeToolbar является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TryTakeToolbar в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод ToolbarClass.SetTarget вызывает метод TryTakeToolbar. Метод же TryTakeToolbar вызывает метод TakeToolbar только в том случае, если элемент управления окно списка объекта ToolbarListbox-класса является видимым.

Возвращаемый тип данных: BYTE

Пример:

```
ToolbarClass.SetTarget PROCEDURE(SIGNED Id)
I USHORT,AUTO
Hit USHORT
 CODE
  ASSERT(~ (SELF.List &= NULL))
  IF Id                                  ! определить явно запрашиваемую цель
   SELF.List.Id = Id
   GET(SELF.List,SELF.List.Id)
   ASSERT (~ERRORCODE())
   SELF.List.Item.TakeToolbar
  ELSE                                   ! определить какую-нибудь пригодную цель
   LOOP I = 1 TO RECORDS(SELF.List)
    GET(SELF.List,I)
    IF SELF.List.Item.TryTakeToolbar() THEN Hit = I.
   END
   IF Hit THEN GET(SELF.List,Hit).
  END
```

Смотри также: TakeToolbar, ToolbarClass.SetTarget

Свойства ToolbarReltree-класса

ToolbarReltree-класс наследует все свойства ToolbarTarget-класса, из которого он получен. См. *Свойства ToolbarTarget-класса* и *Концепции Toolbar-класса* для получения дополнительной информации.

Методы ToolbarReltree-класса

ToolbarReltree-класс наследует все методы ToolbarTarget-класса, из которого он получен. См. *Методы ToolbarTarget-класса* и *Концепции Toolbar-класса* для получения дополнительной информации.

В дополнение к унаследованным (или вместо них), ToolbarReltree-класс содержит следующие методы.

DisplayButtons (сделать доступными для выбора соответствующие кнопки панели управления)

DisplayButtons, VIRTUAL

Метод **DisplayButtons** делает доступными или недоступными для объекта ToolbarReltree-класса соответствующие кнопки глобальной панели управления, в

зависимости от значений таких свойств, как HelpButton, InsertButton, ChangeButton, DeleteButton или SelectButton.

Метод DisplayButtons является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод DisplayButtons в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод TakeToolbar вызывает метод DisplayButtons.

Пример:

```
CODE
Toolbar.Init                                !инициализация объекта Toolbar
 ToolBar.AddTarget( ToolBarForm, -1 ) ! регистрация цели Форма Модернизации
 Toolbar.AddTarget( REL1::Toolbar, ?RelTree )! регистрация цели Дерево отношений
 ToolBar.SetTarget( ?RelTree )          !вызов DisplayButtons через TakeToolbar
!program code
MyToolbarReltreeClass.DisplayButtons PROCEDURE      !работа полученного класса
CODE
DISABLE(Toolbar:History)
                                            !сделать недоступной для выбора как панель,
                                            !так и кнопку
ENABLE(Toolbar:Locate)                      ! сделать доступной для выбора кнопку
                                            !локатора
          PARENT.DisplayButtons
                                                ! Вызов DisplayButtons
!здесь исходный код Вашей программы
```

Смотри также: HelpButton, InsertButton, ChangeButton, DeleteButton, SelectButton, TakeToolbar

TakeToolbar (принять управление панелью управления)

TakeToolbar, VIRTUAL

TakeToolbar метод определяет состояние глобальной панели управления, приводя его в соответствие с объектом ToolbarReltree-класса.

Метод TakeToolbar является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeToolbar в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод TakeToolbarустанавливает для кнопок глобальной панели управления соответствующие значения атрибута, а также делает доступными или нет для выбора кнопки панели управления в зависимости от соответствующего объекта ToolbarReltree-класса. ToolbarClass.SetTarget метод и TryTakeToolbar метод вызывают TakeToolbar метод.

Пример:

```
CODE
Toolbar.Init                                    !инициализация объекта Toolbar
 ToolBar.AddTarget( ToolBarForm, -1 )           ! регистрация цели Форма Модернизации
 Toolbar.AddTarget( REL1::Toolbar, ?RelTree )
                                                ! регистрация цели Дерево Отношений
 ToolBar.SetTarget( ?RelTree )                  !вызов TakeToolbar
!program code
MyToolbarReltreeClass.TakeToolbar PROCEDURE     ! работа полученного класса
CODE                                            !your custom code here
SELF.DisplayButtons                             ! сделать доступными для
                                                !выбора соответствующие
                                                !кнопки
```

Смотри также: ToolbarClass.SetTarget

Свойства ToolbarUpdate-класса

ToolbarUpdate-класс наследует все свойства ToolbarTarget-класса, из которого он получен. См. *Свойства ToolbarTarget-класса* и *Концепции Toolbar-класса* для получения дополнительной информации.

В дополнение к унаследованным, ToolbarUpdate-класс содержит следующие свойства.

Request (запрашиваемая операция с базой данных)

Request BYTE

Свойство **Request** указывает, для каких целей используется объект ToolbarUpdate-класса. ToolbarUpdate-класс использует это значение для того, чтобы установить у кнопок глобальной панели управления соответствующие значения атрибута TIP, а также делает доступными или нет для выбора кнопки панели управления.

Реализация: Методы TakeToolbar и DisplayButtons определяют состояние глобальной панели управления, основываясь на значении свойства Request. Метки соответствия для различных значений свойства Request объявлены в файле

TPLEQU.CLW следующим образом:

```
InsertRecord    EQUATE (1)    !Добавить запись
ChangeRecord    EQUATE (2)    !Изменить текущую запись
DeleteRecord    EQUATE (3)    !Удалить текущую запись
SelectRecord    EQUATE (4)    !Выбрать текущую запись
```

Смотри также: DisplayButtons, TakeToolbar

History (сделать доступной для выбора кнопку истории на глобальной панели управления)

History	BYTE

Свойство **History** указывает, следует ли сделать доступной для выбора кнопку истории на глобальной панели управления для этого объекта ToolbarUpdate-класса. ToolbarUpdate-класс использует это значение, чтобы установить у кнопок глобальной панели управления соответствующие значения атрибута TIP, а также делает доступными или нет для выбора кнопки панели управления.

В соответствии с принятым соглашением кнопка истории восстанавливает поле или запись в его предыдущем значении. См. *Шаблоны Элементов управления — Кнопка «Сохранить»* для получения дополнительной информации.

Реализация: Методы TakeToolbar и DisplayButtons определяют состояние глобальной панели управления, основываясь на значении свойства History. Значение History, равное единице (1), делает кнопку истории на глобальной панели управления доступной для выбора; значение же ноль (0) делает ее недоступной.

Смотри также: DisplayButtons, TakeToolbar

Методы ToolbarUpdate-класса

ToolbarUpdate-класс наследует все методы ToolbarTarget-класса, из которого он получен. См. *Методы ToolbarTarget-класса* и *Концепции Toolbar-класса* для получения дополнительной информации.

В дополнение к унаследованным, ToolbarUpdate-класс содержит следующие методы.

DisplayButtons (сделать доступными для выбора соответствующие кнопки панели управления)

DisplayButtons, VIRTUAL

Метод **DisplayButtons** делает доступными или недоступными для объекта ToolbarUpdate-класса соответствующие кнопки глобальной панели управления, в зависимости от значений таких свойств, как HelpButton, InsertButton, ChangeButton, DeleteButton или SelectButton.

Метод DisplayButtons является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод DisplayButtons в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод TakeToolbar вызывает метод DisplayButtons.

Пример:

```
CODE
Toolbar.Init                                    !инициализация объекта Toolbar
 ToolBar.AddTarget( ToolBarForm, -1 )           ! регистрация цели Форма Модернизации
 Toolbar.AddTarget( REL1::Toolbar, ?RelTree )
                                                ! регистрация цели Дерево отношений
 ToolBar.SetTarget( -1 )
                                                !вызов DisplayButtons через TakeToolbar
!program code
MyToolbarUpdateClass.DisplayButtons PROCEDURE
                                                ! работа полученного класса
CODE
DISABLE(Toolbar:History)
                                                !сделать недоступной для выбора как
                                                !панель, так и кнопку
ENABLE(Toolbar:Locate)                          ! сделать доступной для выбора кнопку
                                                !локатора
PARENT.DisplayButtons                           ! Вызов DisplayButtons
                                              !здесь исходный код Вашей программы
```

Смотри также: HelpButton, InsertButton, ChangeButton, DeleteButton, SelectButton, TakeToolbar

TakeEvent (обработать событие, произошедшее на панели управления)

TakeEvent([*vcr*], *Менеджер_окна*), VIRTUAL

TakeEvent	Обрабатывает событие, произошедшее на глобальной панели управления, применительно к объекту ToolbarUpdate-класса.
vcr	Целочисленная переменная, предназначенная для получения номера нажатой навигационной кнопки *vcr*-панели. Это позволяет методу TakeEvent определить соответствующее последующее действие. Если этот параметр пропущен, объект ToolbarUpdate-класса не осуществляет никакой навигации типа «послать обработку».
Менеджер_окна	Метка WindowManager-объекта для объекта ToolbarUpdate-класса. См. *Менеджер Окна* для получения дополнительной информации.

TakeEvent метод обрабатывает для объекта ToolbarUpdate-класса события, произошедшие на глобальной панели управления.

Параметр *vcr* позволяет методу TakeEvent определить необходимое последующее или послеследующее действие. Наример, метод ToolbarUpdateClass.TakeEvent, может интерпретировать кнопку прокрутки вниз видеомагнитофона как “сохранить и затем прокрутить”. Он, во-первых, производит необходимые действия по сохранению текущей позиции и, во-вторых, «прокручивает» список в направлении, определенном параметром *vcr*.

Метод TakeEvent является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeEvent в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод ToolbarClass.TakeEvent вызывает метод TakeEvent для активного объекта ToolbarTarget-класса. Метод же ToolbarClass.SetTarget определяет активный объект ToolbarTarget-класса.

Пример:

```
ToolbarClass.TakeEvent PROCEDURE(<*LONG VCR>,WindowManager WM)
  CODE
ASSERT(~SELF.List &= NULL)
IF RECORDS(SELF.List)
SELF.List.Item.TakeEvent(VCR,WM)
END
```

Смотри также: ToolbarClass.SetTarget, ToolbarClass.TakeEvent

TakeToolbar (принять контроль над панелью управления)

TakeToolbar, VIRTUAL

TakeToolbar метод определяет состояние глобальной панели управления, приводя его в соответствие с объектом ToolbarUpdate-класса.

Метод TakeToolbar является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeToolbar в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод TakeToolbar устанавливает для кнопок глобальной панели управления соответствующие значения атрибута TIP, а также делает доступными или нет для выбора кнопки панели управления в зависимости от соответствующего объекта ToolbarUpdate-класса. Методы ToolbarClass.SetTarget и TryTakeToolbar вызывают метод TakeToolbar.

Пример:

```
CODE
Toolbar.Init                                    !инициализация объекта Toolbar
ToolBar.AddTarget( ToolBarForm, -1 )
                                                ! регистрация цели Форма Модернизации
 Toolbar.AddTarget( REL1::Toolbar, ?RelTree )
                                                ! регистрация цели Дерево отношений
 ToolBar.SetTarget( -1 )                        !вызов DisplayButtons через TakeToolbar
!текст программы

MyToolbarUpdateClass.TakeToolbar PROCEDURE
                                                ! работа полученного класса
        CODE
                                                !здесь исходный код Вашей програмы
        SELF.DisplayButtons
                                                ! сделать доступными для выбора
                                                !соответствующие кнопки
```

Смотри также: ToolbarClass.SetTarget, TryTakeToolbar

TryTakeToolbar (возвратить индикатор контроля над панелью управления)

TryTakeToolbar, VIRTUAL

TryTakeToolbar метод возвращает значение, сигнализирующее о том, успешно ли объект ToolbarTarget-класа принял контроль над панелью управления. Возвращаемое значение единица (1 или Истина) означает успех; значение ноль (0 или Ложь) говорит о неудачной попытке взять под свой контроль панель управления.

Метод TryTakeToolbar является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TryTakeToolbar в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод ToolbarClass.SetTarget вызывает метод TryTakeToolbar. Метод же TryTakeToolbar вызывает метод TakeToolbar и возвращает значение Истина, поскольку, по умолчанию, объект ToolbarUpdate-класса обычно принимает на себя управление панелью управления.

Возвращаемый тип данных: BYTE

Пример:

```
ToolbarClass.SetTarget PROCEDURE(SIGNED Id)
I USHORT,AUTO
Hit USHORT
 CODE
  ASSERT(~ (SELF.List &= NULL))
  IF Id
! определить явно запрашиваемую цель
    SELF.List.Id = Id
    GET(SELF.List,SELF.List.Id)
    ASSERT (~ERRORCODE())
    SELF.List.Item.TakeToolbar
  ELSE
! определить какую-нибудь пригодную цель
    LOOP I = 1 TO RECORDS(SELF.List)
     GET(SELF.List,I)
     IF SELF.List.Item.TryTakeToolbar() THEN Hit = I.
    END
    IF Hit THEN GET(SELF.List,Hit).
        END
```

Смотри также: TakeToolbar, ToolbarClass.SetTarget

Translator-класс

Краткий обзор

По умолчанию, ABC шаблоны, ABC библиотека и визуальный редактор исходного кода языка Clarion используют в пользовательском интерфейсе американский английский. Тем не менее, Clarion позволяет довольно легко сделать интерфейс пользователя неанглоязычным.

Translator-класс обеспечивает очень быстрый перевод текста пользовательского интерфейса во время выполнения. Translator-класс позволяет устанавливать всем вашим клиентам одно-единственное приложение, независимо от используемого ими языка общения. То есть Вы можете использовать Translator-класс для демонстрации многоязыкового интерфейса пользователя, выбор языка общения в котором основан на предпочтениях конечного пользователя или некоторых других критериях, задаваемых во время работы приложения, таких, например, как содержимое INI файла или управляющего файла.

В то же время, Вы можете использовать файлы перевода языка Clarion (*.TRN) для реализации одного неанглоязычного интерфейса пользователя в период компиляции.

Концепции Translator-класса

Translator-класс и ABUTIL.TRN файл предоставляют возможность выполнения перевода языка общения на этапе работы приложения. То есть Вы можете заставить вашу программу демонстрировать один или более неанглоязычных интерфейсов пользователя, выбор языка общения в которых основан на предпочтениях конечного пользователя или некоторых других критериях, задаваемых во время работы приложения, таких, например, как содержимое INI файла или управляющего файла. Вы можете также использовать Translator-класс для того, чтобы настроить одно-единственное приложение для нескольких клиентов, работающих на разных языках. Translator-класс функционирует на всех элементах пользовательского интерфейса, включая оконные элементы управления, панели заголовка окна, всплывающие подсказки, заголовки окон списка и элементы управления статическими отчетами.

Файл ABUTIL.TRN

ABUTIL.TRN файл содержит пары перевода для всего текста интерфейса пользователя, сгенерированного ABC шаблонами и ABC библиотекой. Пара

перевода представляет собой просто две строки текста: одна строка текста - та, которую надо заменить, другая же –которой надо заменить найденный текст. На этапе выполнения программы Translator-класс применяет пары перевода к каждому элементу интерфейса пользователя.

Вы можете непосредственно редактировать ABUTIL.TRN файл с целью добавления дополнительных пунктов перевода. Этот метод рекомендуется для переведенного текста, общего для нескольких приложений. Пары перевода, которые добавляются в файл ABUTIL.TRN, автоматически распознаются любым приложением, основанным на ABC библиотеке и ABC шаблонах.

Перевод текущего текста

По умолчанию пары перевода из файла ABUTIL.TRN не включают тот текущий текст, который Вам, возможно, хотелось бы видеть на ваших окнах и меню. Чтобы перевести текущий текст, необходимо просто добавить пары перевода на глобальном или на локальном уровне в зависимости от ваших требований. Чтобы помочь идентифицировать текущий текст, Translator-класс автоматически опознает любой текст, оставшийся непереведенным; Вам остается только определить перевод. См. *ExtractText* для получения дополнительной информации.

Макро-замена

Translator-класс определяет и переводит макро-строки. Макро-строка Translator-класса представляет собой просто текст, разграниченный символами процента (%), что выглядит как %mymacro %. Вы можете использовать макро-строки внутри текста, относящегося к элементам управления или заголовкам в приложении, окне или отчете, а также внутри текста, содержащего пары перевода Translator-класса.

Макро-строки определяются путем ограничения их символами процента (%), а заменяющее значение можно определить с помощью задания пар перевода Translator-класса (без знаков процента).

Такая возможность макро-замены позволяет Вам

- переводить малую часть большей строки текста
- производить многоуровневые переводы (значение макро-замены также может содержать макро-строки)

См. *Пояснительный Пример* для получения дополнительной информации.

Связь с другими Классами разработки приложений

WindowManager, Popup-класс и PrintPreview-класс опционально используют Translator-класс для перевода текста во время выполнения программы. Эти классы не требуют для своей работы Translator-класса; однако, если Вы хотите, чтобы они во время их работы выполнялся также и перевод текста приложения, необходимо включить в вашу программу Translator-класс. См. *Пояснительный Пример*.

Реализация ABC шаблона

ABC шаблоны порождают глобальный объект Translator-класса для каждого приложения, у которого в диалоговом окне **Глобальных Свойств** взведен флажок **Разрешить Перевод На Этапе Выполнения**. См. *Краткий обзор Шаблона — Свойства Приложения* для получения дополнительной информации.

Объект Translator-класса называется Translator, и каждая сгенерированная шаблоном процедура обращается к объекту Translator для перевода всего текста из принадлежащего ей окна приложения, окна или отчета. Вдобавок ко всему, сгенерированные шаблоном объекты Popup-класса (ASCIIViewer и BrowseBox шаблоны) и объекты PrintPreview-класса (шаблон Report) используют Translator для перевода текста меню.

Обратите внимание: Шаблоны ABC используют Translator-класс для работы с текстом пользовательского интерфейса, определенного во время компиляции. Шаблоны не обеспечивают переключение между языками общения в интерфейсе пользователя на этапе выполнения.

Файлы-Источники Translator-класса

Файлы с исходным кодом Translator-класса установлены по умолчанию в директорию \CLARION4\LIBSRC. Ниже представлены файлы Translator-класса и их соответствующие компоненты:

ABUTIL.INC	Объявления Translator-класса
ABUTIL.CLW	Определения методов Translator-класса
ABUTIL.TRN	Определенные по умолчанию пары перевода Translator-класса

Пояснительный Пример

Следующий пример показывает типовую последовательность операторов, предназначенных для объявления, порождения, инициализации, использования и завершения объекта Translator-класса.

Этот пример применяет как перевод по умолчанию, так и текущий перевод в окне "выбора". Здесь также реализован сбор и хранение непереведенного текста в файле, так что не приходится вручную собирать текст, подлежащий переводу.

```
PROGRAM
INCLUDE('ABUTIL.INC')                    !объявить TranslatorClass
MAP
END
MyTranslations  GROUP                    ! объявить локальные переводы
Items       USHORT(4)                    !4 пары перевода
PSTRING('Company')                       ! раздел 1 замещаемый текст (макро)
PSTRING('Widget %CoType%')               ! раздел 1 замещающий текст
PSTRING('&Sound')                        ! раздел 2 замещаемый текст
PSTRING('&xSoundx')                      ! раздел 2 замещающий текст
PSTRING('&Volume')                       ! раздел 3 замещаемый текст
PSTRING('&xVolumex')                     ! раздел 3 замещающий текст
PSTRING('OK')                            ! раздел 4 замещаемый текст
PSTRING('xOKx')                          ! раздел 4 замещающий текст
END
INIMgr      INIClass                     !объявить объект INIMgr
Translator  TranslatorClass              ! объявить объект Translator
CoType      STRING('Inc.')               !тип компании по умолчанию
Sound       STRING('ON ')                !предпочитаемое значение по умолчанию
Volume      BYTE(3)                      !предпочитаемое значение по умолчанию
PWindow   WINDOW('%Company% Preferences'),AT(,,100,35),IMM,SYSTEM,GRAY
CHECK('&Sound'),AT(8,6),USE(Sound),VALUE('ON','OFF')
PROMPT('&Volume'),AT(31,19),USE(?VolumePrompt)
SPIN(@s20),AT(8,20,21,7),USE(Volume),HVSCROLL,RANGE(0,9),STEP(1)
BUTTON('OK'),AT(57,3,30,10),USE(?OK)
END
CODE
INIMgr.Init('.\MyApp.INI')               !инициализировать объект INIMgr
INIMgr.Fetch('Preferences','CoType',CoType)
                                         !получить тип компании, по умолчанию Inc.
Translator.Init                          ! инициализировать объект Translator:
                                         ! добавить пары перевода по умолчанию
Translator.AddTranslation(MyTranslations)
                                         !добавить локальные пары перевода
Translator.AddTranslation('CoType',CoType)
                                         !добавить пары перевода из INI-файла
Translator.ExtractText='.\MyApp.trn'
                                         !собрать текст пользовательского интерфейса
    OPEN(PWindow)
```

```
Translator.TranslateWindow
            !перевести текст на элементах управления и на управляющей панели
ACCEPT
IF EVENT() = EVENT:Accepted
IF FIELD() = ?OK
INIMgr.Update('Preferences','Sound',Sound)
INIMgr.Update('Preferences','Volume',Volume)
POST(EVENT:CloseWindow)
…
Translator.Kill      !написать текст пользовательского интерфейса
```

Свойства Translator-класса

Translator-класс содержит следующие свойства:

ExtractText (определить подлежащий переводу текст)

ExtractText CSTRING(File:MaxFilePath)

ExtractText свойство содержит путь и имя файла, получающего текст пользовательского интерфейса, подлежащий переводу во время выполнения. Если ExtractText содержит путь и имя файла, то Translator-класс идентифицирует, извлекает и записывает тот текст интерфейса пользователя, с которым ему пришлось столкнуться, в указанный файл.

Чтобы сгенерировать полный список текста, подлежащего переводу, следует присвоить имя файла свойству ExtractText, откомпилировать и запустить приложение, после чего надо поочередно пооткрывать каждую процедуру, меню и опцию в Вашем приложении. Когда Вы закроете приложение, Translator-класс сгенерирует отсортированный список всех непереведенных участков текста. Вот тогда то можно использовать полученную информацию для того, чтобы снабдить соответствующими переводами все непереведенные участки текста. См. *AddTranslation* для получения дополнительной информации.

Для приложений, которые осуществляют динамические присвоения текста, основываясь на каких-либо данных, Вы, возможно, захотите установить свойство ExtractText в момент развертывания приложения, так что можно будет определять текст, появляющийся на экранах конечного пользователя, в зависимости от особенностей работы пользователей и используемых ими данных.

Реализация: Свойство ExtractText по умолчанию не заполнено, что приводит к тому, что непереведенный текст не извлекается. Ненулевое же значение означает,

что непереведенный текст извлекается и, при наличии правильно указанного пути и имени файла, записывается в указанный файл.

Смотри также: AddTranslation

Методы Translator-класса

Translator-класс содержит следующие методы:

AddTranslation (добавить пары для перевода)

AddTranslation(|*Группа* | **)**
|*Текст, перевод*|

AddTranslation	Добавить пары для перевода.
Группа	Метка структуры, которая содержит одну или более пар *текст / перевод*.
Текст	Константа типа string, переменная, метка соответствия или выражение, содержащее текст пользовательского интерфейса, который следует найти. Translator-класс заменяет каждый найденный фрагмент *текста* его соответствующим *переводом*.
Перевод	Константа типа string, переменная, метка соответствия или выражение, содержащее текст, которым будет заменен соответствующий найденный *текст*.

AddTranslation метод добавляет пары перевода к уже используемым в процессе перевода на этапе исполнения.

Параметр *Текст* не ограничивается единственным словом; это может быть фраза или любую другая строка текста, включая макросы Translator-класса (см. *Концепции Translator-класса - Перевод На Этапе Исполнения*).

Реализация: Параметр *группа* должен обозначать группу, которая *начинается* так же, как и структура TranslatorGroup, объявленная в файле ABUTIL.INC:

```
TranslatorGroup GROUP,TYPE
Number     USHORT
END
```

Когда Вы объявляете *группу* перевода, убедитесь, что установлен правильный номер пар перевода в группе. Например:

```
MyAppTranslator   GROUP
Pairs         USHORT(2)
                                        !2 пары перевода
PSTRING('&Insert')                      !начать первую пару
PSTRING('&Aggregar')                    ! закончить первую пару
PSTRING('Insert a new Record')          ! начать вторую пару
PSTRING('Aggregar el humidor')          ! закончить вторую пару
              END
```

Translator-класс ищет слово целиком, а также различает строчные и прописные буквы при поиске *текста*. Например, 'Вставить', не соответствует '&Вставить', 'ВСТАВИТЬ' или 'Вставить новую запись'.

Init метод применяет метод AddTranslation для использования пар перевода, объявленных в файле ABUTIL.TRN, в процессе перевода.

Пары перевода используются различными" Translate " методами.

Пример:

```
MyTranslations GROUP                    !объявить локальные переводы
Pairs         USHORT(4)                 !4 пары перевода
 PSTRING('&Sound')                      ! текст 1-го пункта
PSTRING('&xSoundx')                     ! заменяющий текст 1-го пункта
PSTRING('&Volume')                      ! текст 2-го пункта
 PSTRING('&xVolumex')                   ! заменяющий текст 2-го пункта
 PSTRING('Preferences')                 ! текст 3-го пункта
PSTRING('xPreferencesx')                ! заменяющий текст 3-го пункта
 PSTRING('OK')                          ! текст 4-го пункта
PSTRING('xOKx')                         ! заменяющий текст 4-го пункта
 END
Translator          TranslatorClass     !объявить объект Translator
 CODE
 Translator.Init                        !инициализировать объект Translator
                                        !объявить пары для перевода по умолчанию
 Translator.AddTranslation(MyTranslations)
                                        !добавить локальные пары для перевода
 OPEN(MyWindow)
 Translator.TranslateWindow
                                        !перевести текст на всех оконных элементах
                                        !управления и заголовок окна
```

Смотри также: Init, TranslateControl, TranslatedControls, TranslateString, TranslateWindow

Init (Инициализировать объект Translator-класса)

Init

Init метод инициализирует объект Translator-класса.

Реализация: Init метод использует метод TranslatorClass.AddTranslation для использования пар перевода, объявленных в файле ABUTIL.TRN, в процессе перевода.

Пример:

```
Translator  TranslatorClass         ! объявить объект Translator
            CODE
            Translator.Init         ! инициализировать объект Translator:
                                    ! с парами перевода, заданными по
                                    !умолчанию
                                    !program code
            Translator.Kill         ! закрыть объект Translator
```

Kill (Закрыть объект Translator-класса)

Kill

Метод **Kill** освобождает любую память, размещенную в течение жизни объекта, и исполняет любой другой требуемый код завершения.

Реализация: В случае, когда свойство ExtractText содержит правильный путь и имя INI-файла, метод **Kill** создает список непереведенных строк текста

Пример:

```
Translator  TranslatorClass         ! объявить объект Translator
            CODE
            Translator.Init         ! инициализировать объект Translator:
                                    ! с парами перевода, заданными по
                                    !умолчанию
                                    !program code
Translator.Kill                     ! закрыть объект Translator
```

TranslateControl (перевести текст для элемента управления)

TranslateControl(*Элемент управления* [, *окно*] **), VIRTUAL**

TranslateControl	Переводит текст с элемента управления.
Элемент управления	Целочисленная константа, переменная, метка соответствия

	или выражение, содержащее номер того элемента управления, текст на котором следует перевести.
Окно	Метка окна приложения, окна или отчета, текст на котором подлежит переводу. Если параметр опущен, TranslateControl работает на окне, активном в данный момент.
TranslateControl	метод переводит текст с указанного *элемента управления*. AddTranslation метод устанавливает значения перевода для текста элемента управления.

Метод TranslateControl является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TranslateControl в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: TranslateControl метод применяет к указанному элементу управления метод TranslateString. Там, где это возможно, метод TranslateControl переводит текст, содержащийся в атрибутах MSG, TIP и FORMAT.

TranslateControl метод не переводит содержимое переменной USE; следовательно он не переводит ни элемент управления STRING, отображающий реальное значение переменной, ни содержимое поля ввода прокручивающегося окна списка, окна текста или комбинированного окна списка. Если есть необходимость, то можно использовать метод TranslateString для перевода этих элементов.

Пример:

```
PWindow WINDOW('Preferences'),AT(,,89,34),IMM,SYSTEM,GRAY
    CHECK('&Sound'),AT(8,6),USE(Sound),VALUE('ON','OFF')
    PROMPT('&Volume'),AT(31,19),USE(?VolumePrompt)
    SPIN(@s20),AT(8,20,21,7),USE(Volume),HVSCROLL,RANGE(0,9),STEP(1)
    BUTTON('OK'),AT(57,3,30,10),USE(?OK)
  END
 CODE
 OPEN(PWindow)
 Translator.TranslateControl(?Sound)        !перевести надпись у поля флажка Sound
 Translator.TranslateControl(?VolumePrompt)! перевести надпись  Volume
           ACCEPT                           !оставить только кнопку OK
           END                              ! и поле заголовка окна
```

Смотри также: AddTranslation, TranslateString

TranslateControls (перевести текст с ряда элементов управления)

TranslateControls(*Первый элемент управления, последний элемент управления* [*, окно*] **), VIRTUAL**

TranslateControls	Переводит текст с ряда элементов управления.
Первый элемент управления	Целочисленная константа, переменная, метка соответствия или выражение, содержащее номер первого элемента управления, который следует перевести.
Последний элемент управления	Целочисленная константа, переменная, метка соответствия или выражение, содержащее номер последнего элемента управления, который следует перевести.
Окно	Метка окна приложения, окна или отчета, текст на котором следует перевести. Если параметр опущен, метод TranslateControl работает на объекте, владеющем фокусом.

TranslateControls метод переводит текст с каждого элемента управления, находящегося между *первым элементом управления* и *последним элементом управления* включительно. Значения перевода для текста с элемента управления устанавливает метод AddTranslation.

Метод TranslateControls является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TranslateControls в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: TranslateControls метод вызывает метод TranslateControl для каждого элемента управления в указанном диапазоне, имеющего атрибут USE. Элементы управления, не имеющие атрибут USE, метод TranslateControls игнорирует.

Пример:

```
PWindow WINDOW('Preferences'),AT(,,89,34),IMM,SYSTEM,GRAY
     CHECK('&Sound'),AT(8,6),USE(Sound),VALUE('ON','OFF')
     PROMPT('&Volume'),AT(31,19),USE(?VolumePrompt)
     SPIN(@s20),AT(8,20,21,7),USE(Volume),HVSCROLL,RANGE(0,9),STEP(1)
     BUTTON('OK'),AT(57,3,30,10),USE(?OK)
   END
 CODE
 OPEN(PWindow)
```

```
Translator.TranslateControls(?Sound,?VolumePrompt)
                                    !перевести текст с элементов управления
                                    !?Sound и ?Volume
    ACCEPT
                                !оставить непереведенным текст на кнопке OK
    END
```

Смотри также: AddTranslation, TranslateControl

TranslateString (переводит текст)

TranslateString(*текст*), VIRTUAL

TranslateString	Переводит строку текста.
Текст	Константа типа string, переменная, метка соответствия, или выражение, содержащее текст, который необходимо найти.

TranslateString метод возвращает значение перевода указанного *текста*. Значения перевода и макро-замен устанавливаются методом AddTranslation.

Метод TranslateString является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TranslateString в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод TranslateString ищет слово целиком, а также различает строчные и прописные буквы при поиске *текста*. Например, 'Вставить', не соответствует '&Вставить', 'ВСТАВИТЬ' или 'Вставить новую запись'. Если для указанного значения параметра *текст* не найдено никакого соответствующего значения перевода, TranslateString возвращает собственно значение параметра *текст*.

Метод TranslateString осуществляет макро-замену Translator-класса, переводя любой ограниченный символами процента (%) текст, который он обнаруживает внутри его собственного возвращаемого значения.

Возвращаемый тип данных: STRING

Пример:

```
MyVar    STRING('Sound')
PWindow  WINDOW('Preferences'),AT(,,89,34),IMM,SYSTEM,GRAY
STRING(@s12),AT(8,30),USE(MyVar)
```

```
BUTTON('OK'),AT(57,3,30,10),USE(?OK)
   END
 CODE
 OPEN(PWindow)
 MyVar=Translator.TranslateString(MyVar)
                                        !перевести значение USE-переменной
          ACCEPT
          END
```

Смотри также: AddTranslation

TranslateWindow (Перевести весь текст на окне)

TranslateWindow([*Окно*]), VIRTUAL

TranslateControls Переводит текст с любого элемента управления на окне.

Окно Метка окна приложения, окна, отчета, подлежащего переводу. Если параметр опущен, метод TranslateControl работает на объекте, в данный момент владеющем фокусом.

TranslateWindow метод переводит текст на объекте, в данный момент владеющем фокусом (окно приложения, окно, отчет). Метод AddTranslation устанавливает значения перевода для текста с элемента управления.

Метод TranslateWindow является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TranslateWindow в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: TranslateWindow метод вызывает TranslateControls метод, определяющий весь список элементов управления на окне, за исключением меню и панели инструментов.

Пример:

```
PWindow WINDOW('Preferences'),AT(,,89,34),IMM,SYSTEM,GRAY
    CHECK('&Sound'),AT(8,6),USE(Sound),VALUE('ON','OFF')
    PROMPT('&Volume'),AT(31,19),USE(?VolumePrompt)
    SPIN(@s20),AT(8,20,21,7),USE(Volume),HVSCROLL,RANGE(0,9),STEP(1)
    BUTTON('OK'),AT(57,3,30,10),USE(?OK)
  END
 CODE
 OPEN(PWindow)
 Translator.TranslateWindow          !перевести все элементы управления
          ACCEPT                     ! включая заголовок окна
          END
```

Смотри также: AddTranslation, TranslateControls

Класс WindowManager

Краткий обзор

Концепции класса WindowManager

Структурированный Менеджер Оконных Процедур

WindowManager класс объявляет Менеджера Окна, который обеспечивает высокоструктурированную, последовательную, гибкую и удобную обработку для оконных процедур языка Clarion. WindowManager класс - настоящий менеджер оконной *процедуры*. Это относится практически к каждой процедуре, сгенерированной с помощью шаблонов, включая процедуры обработки и процедуры отчетов.

Объект WindowManager'а инициализирует процедуру, управляет работой процедуры, обрабатывая все события, происходящие в окне, внутри ACCEPT-цикла, а затем закрывает процедуру. WindowManager обрабатывает события, в основном, путем отправления события другим объектам ABC библиотеки для последующей обработки.

WindowManager - истинный основной класс, и поэтому обрабатывает события и процессы, которые являются общими для большинства Windows-приложений. Для примера реализации WindowManager, специфического для конкретного процесса, см. *Класс Предварительного просмотра Печати*, и *Класс Менеджера Отчетов*.

Осуществляет Политику Формы Модернизации

В дополнение к его функциям как менеджера оконной процедуры общего назначения, WindowManager может быть сконфигурирован для осуществления ряда действий в окнах модернизации — окнах, которые осуществляют вставку записи, ее изменение и удаление. Впоследствии WindowManager будет выполнять выбранные для этих окон действия.

Объединенный с другими Объектами ABC библиотеки

WindowManager тесно связан с некоторыми другими объектами ABC библиотеки; в частности, с объектами Browse-класса, Toolbar-класса, DropList и DropCombo. Эти объекты уведомляют друг друга о своем присутствии, определяют свойства и вызывают методы друг друга для достижения своих целей.

Эти объекты могут переопределять методы WindowManager'а (такие, как TakeAccepted) с целью выполнения своих собственных задач; однако, поскольку предусмотрено двустороннее взаимодействие WindowManager'а и этих ABC объектов, то, как только эти объекты зарегистрированы (AddItem), WindowManager управляет ими непосредственно в соответствии с их задокументированным интерфейсом.

Сгруппированная Обработка События

WindowManager позволяет отдельным виртуальным методам группировать обработку всех событий ACCEPT-цикла в логические, удобные пакеты (виртуальные методы), так что, как только Вам потребуется осуществить обработку события, не заданную по умолчанию, придется произвести изменения только в пределах относительно маленькой области определенного виртуального метода, который осуществляет ту обработку события по умолчанию, которую надо изменить. Такая логическая группировка обработки события в окне происходит следующим образом:

TakeEvent	(обрабатывает **все** события)
TakeWindowEvent	(обрабатывает **все** события, не связанные с полем —производит обработку по умолчанию для большинства не связанных с полем событий)
TakeAccepted	(производит обработку по умолчанию события EVENT:Accepted)
TakeRejected	(производит обработку по умолчанию события EVENT: Rejected)
TakeSelected	(производит обработку по умолчанию события EVENT: Selected)
TakeNewSelection	(производит обработку по умолчанию события EVENT: NewSelection)
TakeCompleted	(производит обработку по умолчанию события EVENT: Completed)
TakeCloseEvent	(производит обработку по умолчанию события EVENT: Close)
TakeFieldEvent	(обрабатывает **все** события, связанные с полем —производит текущую обработку связанных с полем событий)

Реализация ABC шаблона

ABC шаблоны *получают* класс для *каждой* процедуры, управляющей диалоговым окном, включая процедуры отчета и обработки, из класса

WindowManager. Полученный класс называется ThisWindow, и его методы и поведение могут быть изменены с помощью панели WindowBehavior-класса.

ABC шаблоны генерируют виртуальные методы как необходимые для обеспечения процедуры своей собственной инициализацией, обработкой событий и закрытием.

Связь с другими Классами разработки приложений

WindowManager тесно связан с некоторыми другими объектами ABC библиотеки; в частности, с объектами Browse-класса, Toolbar-класса, DropList и DropCombo. Эти объекты уведомляют друг друга о своем присутствии, определяют свойства друг друга и вызывают методы друг друга для достижения своих целей.

Browse-класс использует WindowManager для того, чтобы освежить окно при необходимости. Поэтому, если ваша программа порождает Browse-класс, должен быть также порожден и WindowManager. При этом большинство необходимых действий выполняются автоматически в том случае, когда Вы включите заголовок Browse-класса (ABBROWSE.INC) в секцию данных вашей программы. См. *Пояснительный Пример* и *Browse-класс* для получения дополнительной информации.

WindowManager является базовым для PrintPreview-класса и ReportManager'a. То есть и PrintPreview-класс, и ReportManager образуются из WindowManager, поскольку оба полученных класса управляют оконной процедурой.

PrintPreview-класс — Менеджер Окна Предварительного просмотра Печати

Результатом деятельности PrintPreview-класса является показ полностью сформированного окна предварительного просмотра печати. См. *PrintPreview-класс* для получения дополнительной информации.

ReportManager — Менеджер Окна построения отчета

Результатом деятельности ReportManager является отображение окна, управляющего построением отчета и отображающего его текущее состояние по мере построения. См. *ReportManager-класс* для получения дополнительной информации.

Файлы с исходным кодом WindowManager'a

Исходный код WindowManager установлен по умолчанию в директорию \CLARION4\LIBSRC. Соответствующие WindowManager файлы и их соответствующие компоненты представлены ниже:

ABWINDOW.INC	объявление WindowManager'a
ABWINDOW.CLW	описания каждого метода WindowManager'a

Пояснительный Пример

Следующий пример показывает типовую последовательность операторов, предназначенных для того, чтобы объявлять, порождать, инициализировать, использовать и завершать WindowManager и связанные с ним объекты. В этом примере демонстрируются повторяющиеся вставки в файл Customer, а также добавление телефонных номеров каждого клиента в связанный файл Phones. При этом используется WindowManager для вызова процедуры проверки кода страны клиента, выполняющей сверку с файлом States.

Обратите внимание, что WindowManager признает другие ABC объекты, типа объектов Browse-класса, Toolbar объектов, FileDrop объектов и т.д. Этот пример демонстрирует взаимодействие между объектом WindowManager, объектом FileManager и объектом Browse-класса.

```
AddCustomer   PROGRAM
  INCLUDE('ABWINDOW.INC')          !объявить WindowManager
  INCLUDE('ABFILE.INC')            ! объявить File,View&Relation Mgrs
  INCLUDE('ABBROWSE.INC')          ! объявить BrowseClass
  MAP
SelectState   PROCEDURE            !процедура проверки кода государства
  END
GlobalErrors   ErrorClass          ! объявить объект GlobalErrors
GlobalRequest  BYTE(0),THREAD      !межпроцедурное взаимодействие
GlobalResponse BYTE(0),THREAD      ! межпроцедурное взаимодействие
VCRRequest LONG(0),THREAD          ! межпроцедурное взаимодействие
Customer
FILE,DRIVER('TOPSPEED'),PRE(CUS),CREATE,THREAD
BYNUMBER
KEY(CUS:CUSTNO),NOCASE,OPT,PRIMARY
Record     RECORD,PRE()
CUSTNO     LONG
Name       STRING(30)
State      STRING(2)
 END
END
Phones
FILE,DRIVER('TOPSPEED'),PRE(PH),CREATE,THREAD
IDKEY
KEY(PH:ID),DUP,NOCASE
```

```
Record      RECORD,PRE()
ID          LONG
NUMBER      STRING(20)
END
END
State
FILE,DRIVER('TOPSPEED'),PRE(ST),CREATE,THREAD
StateCodeKey KEY(ST:STATECODE),NOCASE,OPT
Record      RECORD,PRE()
STATECODE STRING(2)
STATENAME  STRING(20)
END
END
Access:State    CLASS(FileManager)          !объявить объект Access:State
Init            PROCEDURE
END
Relate:State CLASS(RelationManager)         ! объявить объект Relate:State
Init            PROCEDURE
END
Access:Customer  CLASS(FileManager)         ! объявить объект Access:Customer
Init            PROCEDURE
END
Relate:Customer CLASS(RelationManager)! объявить объект Relate:Customer
Init PROCEDURE
END
Access:Phones  CLASS(FileManager)           ! объявить объект Access:Phones
Init            PROCEDURE
END
Relate:Phones  CLASS(RelationManager)  ! объявить объект Relate:Phones
Init            PROCEDURE
END
PhoneViewVIEW(Phones)                       !объявить VIEW-структуру для файла
Phones
END
PhoneQ      QUEUE
            !объявить очередь PhoneQ для окна просмотра
PH:ID       LIKE(PH:ID)
PH:NUMBER   LIKE(PH:NUMBER)
ViewPos     STRING(512)
END
CUS:Save  LIKE(CUS:RECORD),STATIC
!объявить структуру, в которой будет хранится копия значения ключа на файл Cus-
tomers
```

```
CUSWindow WINDOW('Add Customer'),AT(,,146,128),IMM,SYSTEM,GRAY
SHEET,AT(4,4,136,102),USE(?CurrentTab)
TAB('General'),USE(?GeneralTab)          !общая закладка
PROMPT('ID:'),AT(8,35),USE(?CUSTNO:Prompt)
ENTRY(@n-14),AT(42,35,41,10),USE(CUS:CUSTNO),RIGHT(1)          ! ID клиента
PROMPT('Name:'),AT(8,49),USE(?NAME:Prompt)
ENTRY(@s30),AT(42,49,90,10),USE(CUS:NAME)                      ! Имя клиента
PROMPT('State:'),AT(8,63),USE(?State:Prompt)
ENTRY(@s2),AT(42,63,40,10),USE(CUS:State)
                                         ! страна, к которой принадлежит клиент
END
TAB('Phones'),USE(?PhoneTab)             ! закладка телефонов
LIST,AT(8,20,128,63),USE(?PhoneList),IMM,HVSCROLL,FROM(PhoneQ),|
FORMAT('38R(2)|M~ID~C(0)@n-14@80L(2)|M~NUMBER~@s20@')
BUTTON('&Insert'),AT(8,87),USE(?Insert)
BUTTON('&Change'),AT(53,87),USE(?Change)
BUTTON('&Delete'),AT(103,87),USE(?Delete)
END
END
BUTTON('OK'),AT(68,110),USE(?OK),DEFAULT
BUTTON('Cancel'),AT(105,110),USE(?Cancel)
END
ThisWindow  CLASS(WindowManager)
                                         !объявить полученный объект ThisWindow
Init        PROCEDURE(),BYTE,PROC,VIRTUAL
                                         !специфическая инициализация процедуры
Kill        PROCEDURE(),BYTE,PROC,VIRTUAL
                                         ! специфическое завершение процедуры
Run         PROCEDURE(USHORT Number,BYTE Request),BYTE,PROC,VIRTUAL
                                         !выполнить процедуру
TakeAccepted   PROCEDURE(),BYTE,PROC,VIRTUAL
            !заданная не по умолчанию обработка события EVENT:Accepted
END
PhBrowse  CLASS(BrowseClass)             !объявить объект PhBrowse
Q           &PhoneQ                      !который работает собъектом ThisWindow
END
 CODE
 ThisWindow.Run()                        !выполнить программу / процедуру
                                         !(Init, Ask, Kill)
ThisWindow.Init  PROCEDURE()
            !установить и «запрограммировать» ThisWindow
ReturnValue  BYTE,AUTO
 CODE
```

```
 GlobalErrors.Init                    !инициализировать объект GlobalErrors
 Relate:Customer.Init                 ! инициализировать объект Relate:Customer
 Relate:State.Init                    ! инициализировать объект Relate:State
 Relate:Phones.Init                   ! инициализировать объект Relate:Phones
 ReturnValue = PARENT.Init()          !вызвать основной класс WindowManager.Init
 Relate:Customer.Open                 !открыть файл Customer и связанные с ним
                                          файлы
 Relate:State.Open                    ! открыть файл State и связанные с ним файлы
                                      !Запрограммировать объект ThisWindow:
 SELF.Request = InsertRecord          ! только на добавление записей
 SELF.FirstField = ?CUSTNO:Prompt     ! CustNo – первое поле для ThisWindow
 SELF.VCRRequest &= VCRRequest        ! установить VCRRequest для ThisWindow
 SELF.Errors &= GlobalErrors          ! установить обработчик событий для
ThisWindow
 SELF.HistoryKey = 734                ! установить клавишу копирования (CTRL')
 SELF.AddHistoryFile(CUS:Record,CUS:Save)     ! установить файл копирования
 SELF.AddHistoryField(?CUS:CUSTNO,1)          ! установить поле копирования
                                              (восстанавливаемое)
 SELF.AddHistoryField(?CUS:NAME,2)            ! установить поле копирования
                                              (восстанавливаемое)
          SELF.AddHistoryField(?CUS:State,3)
          ! установить поле копирования (восстанавливаемое)
 SELF.AddUpdateFile(Access:Customer)
          ! взаимная регистрация FileManager и ThisWindow
 SELF.Primary &= Relate:Customer
          ! взаимная регистрация RelationMgr и ThisWindow
 SELF.AddItem(?Cancel,RequestCancelled)
          ! определить действие для кнопки Cancel
 SELF.InsertAction = Insert:Batch
          ! определить действия при вставке (повторяющиеся)
 SELF.OkControl = ?OK                                  ! установить кнопку OK
 IF SELF.PrimeUpdate() THEN RETURN Level:Notify.       !подготовить запись к
                                                       добавлению
 OPEN(CUSWindow)                                       !открыть окно
 SELF.Opened=True                                      ! пометить его как открытое
                                                       !Запрограммировать объект
PhBrowse,
                                      ! включая регистрацию ThisWindow (SELF)
  PhBrowse.Init(?PhoneList,PhoneQ.ViewPos,PhoneView,PhoneQ,Relate:Phones,SELF)
 PhBrowse.Q &= PhoneQ
 PhBrowse.AddSortOrder(,PH:IDKEY)
 PhBrowse.AddRange(PH:ID,Relate:Phones,Relate:Customer)
 PhBrowse.AddField(PH:ID,PhBrowse.Q.PH:ID)
```

```
  PhBrowse.AddField(PH:NUMBER,PhBrowse.Q.PH:NUMBER)
  PhBrowse.InsertControl=?Insert
  PhBrowse.ChangeControl=?Change
  PhBrowse.DeleteControl=?Delete
  SELF.SetAlerts()                          ! «Горячая клавиша» для ThisWindow
  RETURN ReturnValue
ThisWindow.Kill   PROCEDURE()               !закрыть ThisWindow
ReturnValue  BYTE,AUTO
  CODE
  ReturnValue = PARENT.Kill()    !вызвать основной класс WindowManager.Kill
  Relate:Customer.Close          ! закрыть файл Customer и связанные с ним файлы
  Relate:State.Close             ! закрыть файл State и связанные с ним файлы
  Relate:Customer.Kill                  !закрыть объект Relate:Customer
  Relate:State.Kill                     !закрыть объект Relate:State
  Relate:Phones.Kill                    ! закрыть объект Relate:Phones
  GlobalErrors.Kill                     ! закрыть объект GlobalErrors
  RETURN ReturnValue
ThisWindow.Run   PROCEDURE(USHORT Number,BYTE Request)
                                        !вызвать другие процедуры
ReturnValue BYTE,AUTO
  CODE
  GlobalRequest = Request
          !определить передаваемый между процедурами запрос
  EXECUTE Number                        !запустить определенную процедуру
    SelectState
  END
  ReturnValue = GlobalResponse
          ! определить передаваемый между процедурами ответ на запрос
  RETURN ReturnValue
ThisWindow.TakeAccepted  PROCEDURE()
                                        ! обработка события EVENT:Accepted
ReturnValue
BYTE,AUTO
Looped BYTE
          CODE
          LOOP
IF Looped THEN RETURN Level:Notify ELSE Looped = 1.  !организовать работу цикла
ReturnValue = PARENT.TakeAccepted()
          !произвести стандартную обработку события EVENT:Accepted
CASE ACCEPTED()
          ! произвести особенную обработку события EVENT:Accepted
OF ?CUS:State                           ! при попадании на поле State
ST:STATECODE = CUS:State                ! найти код страны
IF Access:State.Fetch(ST:StateCodeKey)                  ! если не найден
```

```
IF SELF.Run(1,SelectRecord) = RequestCompleted
                                        !предоставить пользователю возможность
                                        !выбора
CUS:State = ST:STATECODE                ! установить выбранную страну
ELSE                                    ! если пользователь не выбрал никакой страны
SELECT(?CUS:State)                      ! установить фокус ввода на поле State
CYCLE                                   ! начать сначала
        END
        END
ThisWindow.Reset()                      !при необходимости переустановить
ThisWindow
END
RETURN ReturnValue
END
```

Свойства WindowManager

Ниже описаны свойства WindowManager.

AutoRefresh (установить флаг автоперерисовки)

AutoRefresh BYTE

AutoRefresh свойство определяет, перерисовывает ли WindowManager окно и его объекты автоматически всякий раз, как обнаруживает изменения в их состоянии. WindowManager проверяет окно и объекты на нем на предмет наличия изменений всякий раз, как только он обработает какое-нибудь событие. Значение единица (1 или Истина) автоматически перерисовывает окно; значение же ноль (0 или Ложь) - нет.

AutoRefresh особенно полезен при переопределении такого объекта Browse-класса, изменение которого приводит к изменениям в поле, являющемся ограничивающим диапазон значением для другого объекта Browse-класса.

Реализация: Init метод устанавливает значение свойства AutoRefresh равным единице. TakeEvent метод осуществляет действие, указанное свойством AutoRefresh, путем вызова метода Reset и только в том случае, когда изменился какой-нибудь зарегистрированный объект Browse-класса.

AddItem метод регистрирует объекты Browse-класса для WindowManager.

Смотри также: AddItem, Init, Reset

AutoToolbar (установить объект панели инструментов, владеющий фокусом после выбора новой закладки)

AutoToolbar BYTE

AutoToolbar свойство определяет, каким образом WindowManager устанавливает ToolbarTarget. Значение единица (1 или Истина) использует объект Toolbar-класса для установки фокуса на соответствующий элемент панели управления всякий раз при выборе новой закладки; значение же ноль (0 или Ложь) использует текущий элемент.

Реализация: Init метод устанавливает значение свойства AutoToolbar равным Истине. TakeNewSelection метод осуществляет действие, указанное свойством AutoToolbar, путем вызова ToolbarClass.SetTarget в том случае, если выбранный элемент управления - ЛИСТ.

Смотри также: Init, ToolbarClass.SetTarget, ToolbarTargetClass

CancelAction (реакция на запрос отмены)

CancelAction BYTE

CancelAction свойство указывает WindowManager действие, которое следует произвести в случае, когда конечный пользователь осуществляет отмену всех произведенных в окне изменений. Возможные действия:

Cancel:Cancel	немедленный отказ (без подтверждения)
Cancel:Save	немедленное сохранение (без подтверждения)
Cancel:Save+Cancel:Query	предложение сохранить или отказаться
Cancel:Cancel+Cancel:Query	предложение закончить редактирование или отказаться

Реализация: Init метод устанавливает свойству CancelAction значение Cancel:Save + Cancel:Query. Метод TakeCloseEvent производит действие, указанное свойством CancelAction.

Метки соответствия различных значений свойства CancelAction объявлены в ABWINDOW.INC следующим образом:

```
ITEMIZE,PRE(Cancel)
Cancel      EQUATE(0)
Save        EQUATE(1)
Query       EQUATE(2)
END
```

Смотри также: Init, TakeCloseEvent, Request, Response

ChangeAction (реакция на запрос изменения)

ChangeAction BYTE

ChangeAction свойство указывает, доступны ли на этой карточке (окне модернизации) действия по изменению. Значение единица (1 или Истина) указывает, что эта карточка может изменять (записывать) запись; значение ноль (0 или Ложь) указывает, что действия по изменению записи в карточке недоступны.

Реализация: Init метод устанавливает значение свойства ChangeAction равным единице (1).

Смотри также: Init

Dead (сигнализатор закрытия)

Dead BYTE, PROTECTED

Свойство **Dead** указывает, следует ли закрыть WindowManager. WindowManager использует это свойство для того, чтобы произвести нормальное закрытие при первом удобном случае. Значение единица (1 или Истина) указывает, что WindowManager должен закрыться; значение же ноль (0 или Ложь) - что WindowManager должен продолжить работу.

Это свойство является защищенным, следовательно, оно может быть использовано лишь методом или WindowManager-класса, или класса, полученного из WindowManager.

Реализация: Метод Kill устанавливает значение свойства Dead равным Истине.

Смотри также: Kill

DeleteAction (реакция на запрос удаления)

DeleteAction BYTE

DeleteAction свойство указывает WindowManager действие, которое следует произвести в случае, когда конечный пользователь запрашивает удаление записи. Возможные действия:

Delete:None	удаление не разрешено
Delete:Warn	с помощью окна сообщения запросить подтверждение удаление
Delete:Form	с помощью карточки (окна обновления) запросить подтверждение на удаление

Delete:Auto непосредственное удаление (без подтверждения)

Реализация: Метод Init устанавливает значение свойства DeleteAction равным Delete:Warn. Метод PrimeUpdate выполняет действие, указанное свойством DeleteAction.

Метки соответствия различных значений свойства DeleteAction объявлены в ABWINDOW.INC следующим образом:

```
                ITEMIZE,PRE(Delete)
None            EQUATE
Warn            EQUATE
Form            EQUATE
Auto            EQUATE
                END
```

Смотри также: Init, TakeCloseEvent, Request, Response

Errors (Объект Error-класса)

Errors &ErrorClass

Свойство **Errors** - ссылка на объект Error-класса, который обрабатывает неожиданные условия для WindowManager. В программе, исходный код которой сгенерирован на основе ABC шаблонов, объект Error-класса называется GlobalErrors.

Реализация: WindowManager-класс не инициализирует свойство Errors. Свойство Errors должно быть инициализировано Init методом. См. *Пояснительный Пример*.

FirstField (Первый оконный элемент управления)

FirstField SIGNED

FirstField свойство содержит номер того оконного элемента управления (метку соответствия поля на окне), который получает фокус ввода при первоначальном отображении окна.

Реализация: WindowManager-класс не инициализирует свойство FirstField. Свойство FirstField должно быть инициализировано Init методом. См. *Пояснительный Пример.*

ForcedReset (взвести флаг переустановки)

ForcedReset BYTE

ForcedReset свойство указывает, следует ли WindowManager безоговорочно переустановиться. Значение ноль (0 или Ложь) допускает условную переустановку (т.е. переустанавливается только в том случае, когда это требуется по обстоятельствам, например, когда конечный пользователь выбирает новый порядок сортировки записей, просматриваемых в BrowseBox, или же использует локатор BrowseBox); значение единица (1 или Истина) приводит к безоговорочной переустановке.

Реализация: Метод Reset выполняет действие, указанное свойством ForcedReset.

Смотри также: Reset

HistoryKey (восстановить значение поля ключа)

HistoryKey SIGNED

HistoryKey свойство позволяет сохранять / восстанавливать "историю поля", и определяет код клавиши, которая восстанавливает прежнее сохраненное значение поля. Когда конечный пользователь нажимает указанную клавишу, WindowManager восстанавливает для поля, владеющего в данный момент фокусом ввода, прежнее сохраненное значение.

Реализация: WindowManager-класс не инициализирует свойство HistoryKey. Свойство HistoryKey должно быть инициализировано Init методом, коль скоро Вы используете т.н. «клавишу истории». См. Пояснительный Пример.

Метод AddHistoryFile определяет файл и буфер записи, из которых впоследствии будут браться значения полей для восстановления. AddHistoryField связывает определенные поля файла истории с соответствующими им элементами управления на окне. Метод SaveHistory сохраняет копию полей истории. Метод RestoreField восстанавливает содержимое определенного элемента управления.

Метки соответствия различных клавиш объявлены в файле \LIBSRC\KEYCODES.CLW.

Смотри также: AddHistoryField, AddHistoryFile, RestoreField, SaveHistory

InsertAction (реакция на запрос вставки)

InsertAction BYTE

InsertAction свойство указывает WindowManager действие, которое следует произвести в случае, когда конечный пользователь запрашивает действие на вставку записи. Возможные действия:

Insert:None	Используют, по умолчанию вставляют действие (Insert:Caller)
Insert:Caller	Возврат в вызвавшую процедуру
Insert:Batch	Непосредственно разрешить следующую вставку
Insert:Query	Предложить либо вернуться, либо произвести другую вставку

Реализация: Метод Init устанавливает значение свойства InsertAction равным Insert:Caller. Метод TakeCompleted выполняет действие, определенное свойством InsertAction.

Метод AddUpdateFile регистрирует файла, используемые в пакетном добавлении.

Метки соответствия различных значений свойства InsertAction объявлены в ABWINDOW.INC следующим образом:

```
ITEMIZE,PRE(Insert)
None    EQUATE
Caller  EQUATE
Batch   EQUATE
Query   EQUATE
        END
```

Смотри также: AddUpdateFile, Init, TakeCompleted, Request, Response

OKControl (элемент управления, определяющий принятие окна—кнопка OK)

OKControl SIGNED

OKControl свойство содержит номер элемента управления (метка соответствия поля) на окне, который указывает на принятие окна конечным пользователем — обычно это кнопка OK.

Реализация: WindowManager-класс не инициализирует OKControl свойство. Свойство OKControl должно быть инициализировано Init методом. См. *Пояснительный Пример.*

Opened (сигнализатор открытия окна)

Opened	BYTE

Свойство **Opened** указывает, было ли открыто окно WindowManager'а. Значение единица (1 или Истина) говорит о том, что оно было открыто; значение же ноль (0 или Ложь) - что нет. Это свойство можно использовать для управления такими задачами, как изменение размеров или сохранение и последующее восстановление координат окна, для выполнения которых требуется, чтобы окно было открыто или закрыто.

Реализация: WindowManager-класс не устанавливает свойство Opened. Свойство Opened должно быть инициализировано Init методом. См. *Пояснительный Пример*.

Смотри также: Init

OriginalRequest (исходный запрос к базе данных)

OriginalRequest	BYTE

Свойство **OriginalRequest** указывает, для выполнения каких действий с базой данных была вызвана процедура WindowManager. Сам же WindowManager использует это свойство для принятия соответствующих решений по обработке в отношении первичных записей, будет ли это сохранение или отказ от изменений и т.д. Возможные варианты запросов:

InsertRecord
ChangeRecord
DeleteRecord
SelectRecord

Реализация: Метод Init приравнивает свойство OriginalRequest свойству Request. Метки соответствия для различных вариантов свойств OriginalRequest и Request объявлены в файле \LIBSRC\TPLEQU.CLW следующим образом:

```
InsertRecord    EQUATE (1)    ! Добавить запись в таблицу
ChangeRecord    EQUATE (2)    ! Изменить текущую запись
DeleteRecord    EQUATE (3)    ! Удалить текущую запись
SelectRecord    EQUATE (4)    ! Выбрать текущую запись
```

Смотри также: Init, Request

Primary (объект RelationManager'а)

Primary	&RelationManager

Для первичного файла WindowManager свойство **Primary** представляет собой ссылку на объект RelationManager. WindowManager использует это свойство для выполнения вставок, изменения и удаления записей.

Реализация: WindowManager-класс не инициализирует свойство Primary. В случае, если процедура выполняет действия по изменению базы данных, свойство Primary должно быть инициализировано Init методом. См. *Пояснительный Пример*.

Request (запрос к базе данных)

Request	BYTE

Свойство **Request** указывает, для выполнения каких действий с базой данных была вызвана процедура WindowManager. Сам же WindowManager использует это свойство для принятия соответствующих решений по обработке в отношении первичных записей, будет ли это сохранение или отказ от изменений и т.д. Возможные варианты запросов:

```
InsertRecord
ChangeRecord
DeleteRecord
SelectRecord
```

Реализация: WindowManager-класс не определяет свойство Request. Свойство Request должно быть инициализировано Init методом. Метод WindowManagerClass.Init устанавливает свойство OriginalRequest равным свойству Request с целью сохранения его начального значения. См. *Пояснительный Пример*.

Метки соответствия различных значений свойств OriginalRequest и Request объявлены в файле \LIBSRC\TPLEQU.CLW следующим образом:

```
InsertRecord    EQUATE (1)    ! Добавить запись в таблицу
ChangeRecord    EQUATE (2)    ! Изменить текущую запись
DeleteRecord    EQUATE (3)    ! Удалить текущую запись
SelectRecord    EQUATE (4)    ! Выбрать текущую запись
```

Смотри также: Init, OriginalRequest

ResetOnGainFocus (сигнализатор переустановки при получении окном фокуса)

ResetOnGainFocus BYTE

ResetOnGainFocus свойство указывает на то, следует ли при получении окном фокуса безусловно переопределить WindowManager. Значение ноль (0 или Ложь) разрешает условную переустановку (переустанавливать только в случае, если изменения необходимы, например, когда конечный пользователь определяет новый порядок сортировки записей в BrowseBox'е или определяет новое значение в поле локатора BrowseBox'а); значение единица (1 или Истина) приводит к безоговорочной переустановке (переустановка происходит независимо от обстоятельств).

Реализация: Свойство ResetOnGainFocus по умолчанию считается равным нулю (0). TakeWindowEvent метод выполняет действие, указанное ResetOnGainFocus свойством, при потере же окном фокуса опционально устанавливает значение свойства ForcedReset равным Истине.

Смотри также: ForcedReset

Response (ответ на запрос к базе данных)

Response BYTE

Свойство **Response** указывает ответ WindowManager'а на исходный запрос к базе данных (обозначенный свойством OriginalRequest). Сам же WindowManager использует это свойство для принятия соответствующих решений по обработке в отношении первичных записей, будет ли это сохранение или отказ от изменений и т.д.

SetResponse метод определяет значение свойства Response и покидает процедуру.

Реализация: Метки соответствия для свойства Response объявлены в файле \LIBSRC\TPLEQU.CLW следующим образом:

```
RequestCompleted    EQUATE (1)      ! Обновление завершено
RequestCancelled    EQUATE (2)      ! Обновление прервано
```

Смотри также: OriginalRequest, SetResponse

Saved (создание копии буфера записи первичного файла)

```
Saved USHORT, PROTECTED
```

Свойство **Saved** размещает копию буфера записи первичного файла WindowManager'a. WindowManager же использует это свойство для того, чтобы обнаружить возможные изменения в записи, и, при необходимости, восстановить запись в ее прежнем значении.

Метод SetSaved устанавливает значение свойства Saved.

Это свойство является защищенным, поэтому может использоваться только методом либо класса WindowManager, либо класса, полученного из WindowManager.

Реализация: WindowManager использует методы FileManager. SaveBuffer, FileManager. RestoreBuffer и FileManager. EqualBuffer (через их свойство Primary) для управления свойством Saved.

Смотри также: FileManager.SaveBuffer, FileManager.RestoreBuffer, FileManager.EqualBuffer

Translator (объект Translator-класса)

```
Translator &TranslatorClass
```

Свойство **Translator** представляет собой ссылку на объект Translator-класса для WindowManager. WindowManager же использует это свойство для перевода текста, находящегося на окне, на соответствующий язык.

AddItem метод устанавливает значение свойства Translator.

Реализация: WindowManager-класс не инициализирует свойство Translator. Он только вызывает объект Translator-класса в случае ненулевого значения свойства Translator. Если перевод необходим, то свойство Translator должно быть инициализировано Init методом. См. *Пояснительный Пример*.

Смотри также: AddItem

VCRRequest (отложенный запрос скроллирования)

```
VCRRequest &LONG
```

VCRRequest свойство представляет собой ссылку на переменную,

идентифицирующую запрос на скроллирование, сделанный одновременно с запросом на произведение действий в базе данных. WindowManager использует это свойство для того, чтобы выполнить запрос скроллирования после окончания выполнения действий с базой данных.

Например, когда конечный пользователь изменяет какое-нибудь поле на форме, а затем нажимает кнопку Insert, он одновременно запрашивает два действия: сохранение изменения и переход к следующей записи. WindowManager заканчивает выполнение запроса изменения, и только затем обрабатывает запрос на скроллирование.

Реализация: Метки соответствия различных значений свойства VCRRequest объявлены в файле \LIBSRC\ABTOOLBA.INC следующим образом:

```
ITEMIZE,PRE(VCR)
Forward                 EQUATE(Toolbar:Down)
Backward                EQUATE(Toolbar:Up)
PageForward             EQUATE(Toolbar:PageDown)
PageBackward            EQUATE(Toolbar:PageUp)
First                   EQUATE(Toolbar:Top)
Last                    EQUATE(Toolbar:Bottom)
Insert                  EQUATE(Toolbar:Insert)
None                    EQUATE(0)
END
```

Методы WindowManager

Функциональная Организация — Предполагаемое Использование

Для лучшего понимания WindowManager будет полезно организовать его различные методы в две больших категории согласно их предполагаемому использованию — первичный интерфейс и виртуальные методы. Такая организация, по мнению разработчиков, наиболее полно отражает типовое использование методов WindowManager.

Первичные Интерфейсные Методы

Первичные интерфейсные методы, вызов которых, скорее всего, будет иметь место в Вашей программе, могут быть далее разделены на три категории:

Одноразовое применение:

Run[V]	выполнить эту процедуру
Init[V]	инициализировать объект WindowManager'a
AddHistoryField	добавить восстанавливаемый элемент управления и поле
AddHistoryFile	добавить восстанавливаемый фал истории изменений
AddItem	запрограммировать объект WindowManager'a
AddUpdateFile	регистрация файлов для пакетного добавления
Kill [V]	завершить объект WindowManager'a

Используются в основном:

Используются изредка:

Run[V]	запустить другую процедуру
SaveHistory	сохранить поля истории для последующего восстановления
PostCompleted	произвести заключительную обработку окна

[V] Эти методы также являются виртуальными.

Виртуальные Методы

В основном Вам не придется прямо вызывать эти методы, поскольку они вызываются Первичными Интерфейсными методами. Однако у Вас наверняка часто будет возникать потребность в переопределении этих методов, а поскольку они являются виртуальными, то сделать это достаточно просто. Эти методы уже по умолчанию обеспечивают «разумное» поведение, в случае, если Вы не хотите переопределить их.

Init	инициализировать объект WindowManager'a
Ask	отображение окна и обработка связанных с ним событий
Kill	закрыть объект WindowManager'a
Open	действия по обработке события EVENT:OpenWindow
PrimeFields	действия по подготовке полей формы модернизации
PrimeUpdate	модернизация или подготовка к модернизации
Reset	переопределить состояние окна и его элементов
RestoreField	восстанавливает последнее сохраненное значение элемента управления
Run	запустить ту или иную процедуру
SetAlerts	подготовка «горячих клавиш» для элементов управления на окне
SetResponse	определить для окна обработку по типу OK или Cancel
TakeAccepted	обработка события EVENT:Accepted

TakeCompleted	действия по принятию формы модернизации
TakeCloseEvent	действия при непринятии окна
TakeEvent	обработка всех событий
TakeFieldEvent	обработка связанных с полем событий
TakeNewSelection	обработка события EVENT:NewSelection
TakeRejected	обработка события EVENT:Rejected
TakeSelected	обработка события EVENT:Selected
TakeWindowEvent	обработка не связанных с полем событий
Update	подготовка строк данных к записи на диск

AddHistoryField (добавить восстанавливаемый элемент управления и поле)

AddHistoryField(*Элемент управления, поле* **)**

AddHistoryField	Добавляет поле истории к объекту WindowManager'a.
Элемент управления	Целочисленная константа, переменная, метка соответствия или выражение, содержащее номер элемента управления, чье содержание подлежит восстановлению из *поля*. Представляет собой номер элемента управления.
поле	Целочисленная константа, переменная, метка соответствия или выражение, содержащее положение поля в структуре записей файла истории. Поле идентифицируется его положением в описании файла. Значение единица (1) указывает на первое поле, два (2) - на второе и т.д. См. *ЧТО* и *ГДЕ* в *Описании Языка* для получения дополнительной информации.

AddHistoryField метод добавляет поле истории к объекту WindowManager'a. AddHistoryField связывает элемент управления окна с соответствующим полем или столбцом базы данных так, что WindowManager может восстановить содержание элемента управления при нажатии конечным пользователем клавиши истории (или элемента FrameBrowseControl аналогичного кнопке).

Реализация: AddHistoryFile метод определяет файл и буфера записи, в которых сохраняются и откуда восстанавливаются поля. Метод AddHistoryField связывает определенные поля файла истории с соответствующими им оконными элементами управления. SaveHistory метод сохраняет копию полей истории. RestoreField метод восстанавливает содержание определенного элемента управления.

Пример:

```
ThisWindow.Init PROCEDURE()
 CODE                                !procedure code
```

```
SELF.HistoryKey = CtrlR
SELF.AddHistoryFile(CLI:Record,History::CLI:Record)
SELF.AddHistoryField(?CLI:Name,2)
SELF.AddHistoryField(?CLI:StateCode,3)
```

Смотри также: AddHistoryFile, HistoryKey, RestoreField, SaveHistory

AddHistoryFile (добавить файл истории)

AddHistoryFile(*буфер записи, буфер сохранения* **)**

AddHistoryFile	Добавляет файл истории к объекту WindowManager.
буфер записи	Метка записи файла истории.
буфер сохранения	Метка переменной с атрибутом STATIC, объявленная подобной *буферу записи* (LIKE(*буфер записи*)). Именно в этой переменной WindowManager сохраняет, и именно оттуда он восстанавливает сохраняемое значение.

AddHistoryFile метод добавляет файл истории к объекту WindowManager. AddHistoryFile устанавливает буфер записи файла и соответствующий буфер хранения так, что когда конечный пользователь нажимает т.н. клавишу истории (или элемента FrameBrowseControl аналогичного кнопке), WindowManager может восстановить необходимое значение из буфера хранения.

Реализация: AddHistoryFile метод определяет файл и буфера записи, в которых сохраняются и откуда восстанавливаются поля. Метод AddHistoryField связывает определенные поля файла истории с соответствующими им оконными элементами управления. SaveHistory метод сохраняет копию полей истории. RestoreField метод восстанавливает содержание определенного элемента управления.

Пример:

```
ThisWindow.Init PROCEDURE()
 CODE                                    !procedure code
 SELF.HistoryKey = CtrlR
 SELF.AddHistoryFile(CLI:Record,History::CLI:Record)
 SELF.AddHistoryField(?CLI:Name,2)
 SELF.AddHistoryField(?CLI:StateCode,3)
```

Смотри также: AddHistoryField, HistoryKey, RestoreField, SaveHistory

AddItem (установить связь объекта WindowManager'а с объектами других классов)

```
AddItem(  | BrowseClass          | )
        | FileDropClass          |
        | ToolbarClass           |
        | ToolbarUpdateClass     |
        | TranslatorClass        |
        | WindowResizeClass      |
        | элемент управленияl, response |
```

AddItem	Добавляет WindowManager'у определенные функциональные возможности.
BrowseClass	Метка объекта Browse-класса.
FileDropClass	Метка объекта DropList-класса.
ToolbarClass	Метка объекта Toolbar-класса.
ToolbarUpdateClass	Метка объекта ToolbarUpdate-класса.
TranslatorClass	Метка объекта Translator-класса.
WindowResizeClass	Метка объекта WindowResize-класса.
Элемент управления	Целочисленная константа, переменная, метка соответствия или выражение, содержащее номер элемента управления, принятие которого вызывает *действие* — обычно это кнопки OK и Cancel.
Ответ	Целочисленная константа, переменная, метка соответствия, или выражение, отображающее результат действия при принятии *элемента управления.*

AddItem метод добавляет объекту WindowManager определенные функциональные возможности.

Реализация: Когда *элемент управления* принят, метод TakeAccepted присваивает свойству Response значение *ответ*. Метки соответствия для различных значений параметра *ответ* объявлены в файле \LIBSRC\TPLEQU.CLW следующим образом:

```
RequestCompleted EQUATE (1)          ! Изменение завершено
RequestCancelled  EQUATE (2)         !Изменение прервано

Пример:
ThisWindow.Init PROCEDURE()          !настройка WindowManager
 CODE                                !текст процедуры
      SELF.AddItem(Toolbar)
```

```
        ! Добавить функциональные возможности панели инструментов
 SELF.AddItem(ToolbarForm)                !должен следовать за AddItem(Toolbar)
 SELF.AddItem(?Cancel,RequestCancelled)
        ! Добавить функциональные возможности кнопки cancel
 SELF.AddItem(Resizer)
         Добавить функциональные возможности по изменению размеров окна
 SELF.AddItem(Translator)
        ! Добавить функциональные возможности по переводу языка
MyBrowse.Init(?CusList,Cus:Q.Position,Cus:View,Cus:Q,Relate:Cus,ThisWindow)
        !текст процедуры
MyBrowse.Init  PROCEDURE                 |
(SIGNED ListBox,*STRING Posit,VIEW V,QUEUE Q,RelationManager F,WindowManager
WM)
 CODE      !procedurecode
WM.AddItem(SELF)
        ! Добавить функциональные возможности окна просмотра
```

Смотри также: Response, TakeAccepted

AddUpdateFile (регистрация файлов для пакетного добавления)

AddUpdateFile(*Менеджер файла* **)**

AddUpdateFile	Регистрирует объект FileManager для объекта WindowManager'а.
Менеджер файла	Метка объекта FileManager для файла.

AddUpdateFile метод выполняет взаимную регистрацию объектов FileManager и объекта WindowManager для тех файлов, чьи буферы записи должны быть сохранены и потом восстановлены (для поддержки функции пакетного добавления).

Реализация: WindowManager использует FileManager файлов модернизации для сохранения и восстановления буфера файла.

InsertAction свойство определяет добавляемый пакет.

Пример:

```
ThisWindow.Init PROCEDURE()
 CODE                                !procedure code
 SELF.AddUpdateFile(Access:Client)   !procedure code
```

Смотри также: InsertAction

Ask (отображение окна и обработка событий, связанных с ним)

Ask, VIRTUAL

Метод **Ask** отображает окно и обрабатывает его события.

Метод Ask является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод Ask в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод Run вызывает Ask метод только в том случае, если Init метод возвращает значение Level:Benign. Если значение свойства Dead – Истина, то метод Ask возвратиться немедленно. Метод Kill устанавливает свойство Dead равным Истине, так что вызов метода Kill до вызова метода Ask имеет эффект закрытия оконной процедуры еще до того, как окно будет отображено методом Ask.

Метод Ask организует ACCEPT-цикл для окна и вызывает метод TakeEvent для обработки всех событий. ACCEPT- цикл продолжается до тех пор, пока TakeEvent не возвратит значение Level:Fatal.

Совет: Для того, чтобы закрыть оконную процедуру во время работы метода Ask, необходимо просто возвратить значение Level:Fatal из любого метода типа «Take».

Когда метод TakeEvent возвращает значение Level:Notify, ACCEPT-цикл возвращается в начало.

Совет: Для немедленного прекращения обработки события (включая прекращение изменения размеров и обработку «горячих клавиш») необходимо просто возвратить значение Level: Notify из любого метода типа «Take».

Пример:

```
WindowManager.Run PROCEDURE
 CODE
 IF ~SELF.Init()
  SELF.Ask
 END
 SELF.Kill

WindowManager.Ask PROCEDURE
 CODE
 IF SELF.Dead THEN RETURN .
```

```
 CLEAR(SELF.LastInsertedPosition)
 ACCEPT
  CASE SELF.TakeEvent()
  OF Level:Fatal
   BREAK
  OF Level:Notify
   CYCLE    ! Для быстрого прекращения обработки события Not as dopey at it looks,
it is for 'short-stopping' certain events
  END
 END
```

Смотри также: Dead, Init, Kill, Run, TakeEvent

Init (инициализация объекта WindowManager)

Init, VIRTUAL, PROC

Init метод инициализирует объект WindowManager. Метод Init для сигнализации о нормальном прохождении инициализации возвращает значение Level:Benign.

Init метод настраивает объект WindowManager'a и инициализирует общую процедуру.

WindowManager может быть сконфигурирован для осуществления ряда действий в окнах модернизации (карточках). Init метод может быть использован для определения поведения карточки (окна модернизации) путем установки таких свойств, как Request, InsertAction, ChangeAction, и DeleteAction.

WindowManager тесно связан с некоторыми другими объектами ABC библиотеки. Вызывая метод AddItem, можно использовать Init метод для регистрации этих объектов в WindowManager. Тогда эти объекты смогут устанавливать свойства друг друга, а также, при необходимости, вызывать методы друг друга для выполнения своих соответствующих целей.

Метод Init является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод Init в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Как правило, Init метод связан с методом Kill, выполняя задачи, обратные задачам метода Kill.

Метод Run вызывает метод Init.

Метки соответствия возвращаемых значений объявлены в ABERROR.INC.

Совет: Чтобы предотвращать выполнение метода Ask, необходимо возвратить из метода Init значение Level:Notify.

Возвращаемый тип данных: BYTE

Пример:

```
MyWindowManager.Run PROCEDURE
 CODE
 IF SELF.Init() = Level:Benign
  SELF.Ask
 END
 SELF.Kill
ThisWindow.Init PROCEDURE()
 CODE
 SELF.Request = GlobalRequest
 PARENT.Init()
 SELF.FirstField = ?Browse:1
 SELF.VCRRequest &= VCRRequest
 SELF.Errors &= GlobalErrors
 SELF.AddItem(Toolbar)
 CLEAR(GlobalRequest)
 CLEAR(GlobalResponse)
 SELF.AddItem(?Close,RequestCancelled)
 Relate:Client.Open
 FilesOpened = True
 OPEN(QuickWindow)
 SELF.Opened=True
 Resizer.Init(AppStrategy:Surface,Resize:SetMinSize)
 SELF.AddItem(Resizer)
 Resizer.AutoTransparent=True
BRW1.Init(?Browse:1,Queue:Browse:1.Position,BRW1::View:Browse,Queue:Browse:1,Relate:Client,SELF)
 BRW1.Q &= Queue:Browse:1
 BRW1::Sort1:StepClass.Init(+ScrollSort:AllowAlpha,ScrollBy:Runtime)
 BRW1.AddSortOrder(BRW1::Sort1:StepClass,CLI:NameKey)
 BRW1.AddLocator(BRW1::Sort1:Locator)
 BRW1::Sort1:Locator.Init(,CLI:Name,1,BRW1)
 BRW1.AddField(CLI:Name,BRW1.Q.CLI:Name)
 BRW1.AddField(CLI:StateCode,BRW1.Q.CLI:StateCode)
```

```
BRW1.AddField(CLI:ID,BRW1.Q.CLI:ID)
BRW1.InsertControl=?Insert:2
BRW1.ChangeControl=?Change:2
BRW1.DeleteControl=?Delete:2
BRW1.AddToolbarTarget(Toolbar)
BRW1.AskProcedure = 1
SELF.SetAlerts()
RETURN Level:Benign
```

Смотри также: AddItem, Ask, Kill, Run

Kill (закрыть объект WindowManager)

Kill, VIRTUAL, PROC

Метод **Kill** освобождает любую память, размещенную в течение жизни объекта, а также выполняет любой другой требуемый код завершения. Kill возвращает значение, несущее в себе статус закрытия.

Метод Kill является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод Kill в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Kill устанавливает свойство Dead равным Истине и, для сигнализации о нормальном закрытии, возвращает значение Level:Benign. Если свойство Dead уже установлено равным Истине, метод Kill возвращает значение Level:Notify для того, чтобы указать, что никакого дополнительного действия не требуется.

Как правило, Init метод действует совместно с методом Kill, выполняя задачи, обратные задачам метода Kill.

Метод Run вызывает метод Init.

Метки соответствия возвращаемых значений объявлены в ABERROR.INC.

Возвращаемый тип данных: BYTE

Пример:

```
ThisWindow.Kill PROCEDURE()
CODE
IF PARENT.Kill() THEN RETURN Level:Notify.
IF FilesOpened
Relate:Defaults.Close
END
IF SELF.Opened
INIMgr.Update('Main',AppFrame)
            END
            GlobalResponse =
CHOOSE(LocalResponse=0,RequestCancelled,LocalResponse)
```

Смотри также: Dead, Init, Run

Open (действия по обработке события EVENT:OpenWindow)

Open, VIRTUAL

Метод **Open** готовит окно к отображению. Он предназначен для того, чтобы обрабатывать на открывающемся окне события типа EVENT:OpenWindow и EVENT:GainFocus.

Метод Open является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод Open в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод Open вызывает Translator, если он присутствует, и вызывает метод Reset для переустановки окна.

Метод TakeWindowEvent вызывает метод Open.

Пример:

```
ThisWindow.TakeWindowEvent          PROCEDURE
 CODE
 CASE EVENT()
 OF EVENT:OpenWindow
  IF ~BAND(SELF.Inited,1)
    SELF.Open
  END
 OF EVENT:GainFocus
```

```
  IF BAND(SELF.Inited,1)
    SELF.Reset
  ELSE
    SELF.Open
  END
 END
 RETURN Level:Benign

ThisWindow.Open                    PROCEDURE
 CODE
  IF ~SELF.Translator&=NULL
    SELF.Translator.TranslateWindow
  END
  SELF.Reset
  SELF.Inited = BOR(SELF.Inited,1)
```

Смотри также: Reset, TakeWindowEvent

PostCompleted (произвести заключительную обработку окна)

PostCompleted

PostCompleted метод производит заключительную обработку окна или обработку при закрытии окна. Обычно этот процесс начинается с нажатием кнопки «OK». Реальная же обработка зависит от типа определенного окна.

Реализация: Метод TakeAccepted вызывает метод PostCompleted. ToolbarUpdateClass.TakeEvent также вызывает метод PostCompleted. Метод PostCompleted задает для Форм модернизации режим AcceptAll (см. *SELECT* в *Описании Языка* для получения дополнительной информации), а также посылает всем остальным окнам событие EVENT:Completed.

Пример:

```
WindowManager.TakeAccepted PROCEDURE
I LONG,AUTO
A SIGNED,AUTO
 CODE
 A = ACCEPTED()
 IF ~SELF.Toolbar &= NULL
   SELF.Toolbar.TakeEvent(SELF.VCRRequest,SELF)
   IF A = Toolbar:History
     SELF.RestoreField(FOCUS())
   END
```

```
 END
 LOOP I = 1 TO RECORDS(SELF.Buttons)
  GET(SELF.Buttons,I)
  IF SELF.Buttons.Control = A
   SELF.SetResponse(SELF.Buttons.Action)
   RETURN Level:Notify
  END
 END
 IF SELF.OkControl AND SELF.OkControl = A
  SELF.PostCompleted
 END
 RETURN Level:Benign
```

Смотри также: OKControl, TakeAccepted

PrimeFields (действия по подготовке полей формы модернизации)

PrimeFields, VIRTUAL, PROC

Метод **PrimeFields** - виртуальный метод, предназначенный для подготовки полей к добавлению в файл. PrimeFields вызывается *после* метода FileManager.PrimeRecord для того, чтобы предоставить форме модернизации возможность соответствующей подготовки полей.

Метод PrimeFields является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод PrimeFields в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Пример:

```
ThisWindow.PrimeFields PROCEDURE
        CODE
        CLI:StateCode = 'FL'
        PARENT.PrimeFields
```

PrimeUpdate (модернизация или подготовка к модернизации)

PrimeUpdate, VIRTUAL, PROC

PrimeUpdate метод готовит буфер записи ко входу в ACCEPT-цикл формы модернизации. Для действий, которые могут быть закончены и без ACCEPT-цикла, метод PrimeUpdate предотвращает выполнение ACCEPT-цикла путем возврата

соответствующего значения.

Для сигнализации о том, что буфер записи готов и следует начать выполнение ACCEPT-цикла соответствующей формы модернизации, PrimeUpdate возвращает значение Level:Benign.

Для сигнализации о том, что не следует начинать выполнение ACCEPT-цикла, как по причине того, что буфер записи не может быть подготовлен, так и потому, что требуемое действие PrimeUpdate закончил и нет необходимости в каких-нибудь дальнейших действиях, PrimeUpdate возвращает значение Level: Fatal.

Метод PrimeUpdate является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод PrimeUpdate в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Метод PrimeUpdate готовит буфер записи при добавлении записи, удаляет запись при автоматическом удалении, но во всех случаях сохраняет копию буфера записи.

Метки соответствия возвращаемого значения объявлены в файле ABERROR.INC.

Возвращаемый тип данных: BYTE

Пример:

```
ThisWindow.Init PROCEDURE()
 CODE
          !procedure code
 IF SELF.PrimeUpdate() THEN RETURN Level:Fatal .
 OPEN(ClientFormWindow)
 SELF.SetAlerts()
 RETURN Level:Benign
```

Reset (вновь переопределить состояние окна для отображения)

Reset([*Произвести переустановку*] **), VIRTUAL**

Reset Переустанавливает объект WindowManager.

Произвести переустановку Числовая константа, переменная, метка соответствия, или выражение, которое определяет условную или безусловную переустановку окна. Значение единица (1 или Истина) безусловно переустанавливает окно; значение же ноль (0 или Ложь) переустанавливает окно только в том случае, когда этого требуют обстоятельства, такие, например, как выбор нового порядка сортировки на объекте, предназначенном для просмотра записей, или же произошедшее изменение в поле, определяющем то, какие записи будут отображены (т.н. «reset field»). Если параметр опущен, *Произвести переустановку* по умолчанию считается равным нулю (0).

Метод **Reset** переустанавливает объект WindowManager и любые другие зарегистрированные объекты (AddItem). Если значение параметра *Произвести переустановку* равно единице (1 или Истина), то происходит безоговорочная переустановка всех объектов, поэтому оно должно использоваться осторожно с целью недопущения значительного замедления работы.

Метод Reset является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод Reset в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод Reset для каждого объекта Browse-класса, зарегистрированного AddItem методом, вызывает методы ResetSort и UpdateWindow. Для каждого же объекта FileDropClass, зарегистрированного AddItem методом, он вызывает метод ResetQueue.

Методы Open, TakeWindowEvent и TakeNewSelection – все они вызывают метод Reset.

Пример:

```
ThisWindow.TakeWindowEvent              PROCEDURE
 CODE
 CASE EVENT()
 OF EVENT:GainFocus
  IF BAND(SELF.Inited,1)
   SELF.Reset
  ELSE
   SELF.Open
  END
 OF EVENT:Sized
```

```
  IF BAND(SELF.Inited,2)
    SELF.Reset
  ELSE
    SELF.Inited = BOR(SELF.Inited,2)
  END
 END
 RETURN Level:Benign
```

Смотри также: AutoRefresh, Open, ResetOnGainFocus, TakeNewSelection, TakeWindowEvent, BrowseClass.AddResetField, BrowseClass.ResetSort, BrowseClass.UpdateWindow

RestoreField (восстанавливает последнее сохраненное значение элемента управления)

RestoreField(*элемент управления* **), VIRTUAL**

RestoreField Восстанавливает содержимое указанного элемента управления.

Элемент управления Целочисленная константа, переменная, метка соответствия, или выражение, содержащее номер элемента управления, содержание которого следует восстановить. Представляет собой номер оконного элемента управления.

RestoreField метод восстанавливает содержимое указанного элемента управления путем приведения его к тому значению, которое он содержал в себе в момент, когда запись последний раз была сохранена. RestoreField работает только в том случае, если установлено свойство HistoryKey.

Метод RestoreField является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод RestoreField в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод AddHistoryFile определяет файл и те буфера записи, которые используются для сохранения и восстановления значений полей. Метод AddHistoryField связывает определенные поля файла истории с соответствующими им оконными элементами управления. Метод SaveHistory сохраняет копию полей истории. Метод RestoreField восстанавливает содержание определенного элемента управления.

Пример:

```
WindowManager.TakeAccepted PROCEDURE
A SIGNED,AUTO
 CODE
 A = ACCEPTED()
 IF ~SELF.Toolbar &= NULL
  SELF.Toolbar.TakeEvent(SELF.VCRRequest,SELF)
  IF A = Toolbar:History
   SELF.RestoreField(FOCUS())
  END
 END                                    !procedure code
```

Смотри также: AddHistoryField, AddHistoryFile, HistoryKey, SaveHistory

Run (Запустить эту процедуру или указанную зависимую процедуру)

Run([*номер, запрос*] **), VIRTUAL, PROC**

Run Запускает эту процедуру или указанную зависимую процедуру.

Номер Целочисленная константа, переменная, метка соответствия, или выражение, идентифицирующее зависимую процедуру, подлежащую запуску. Значение единица (1) запускает первую процедуру, значение два (2) - вторую и т.д. Как правило, обозначает номер строки, содержащей вызов процедуры, в структуре EXECUTE. Если параметр опущен, Run осуществляет обычную для WindowManager последовательность Init-Ask-Kill.

запрос целочисленная константа, переменная, метка соответствия или выражение, идентифицирующее действие (добавление, изменение, удаление, выбор) производимое зависимой процедурой. Если параметр опущен, Run осуществляет обычную для WindowManager последовательность Init-Ask-Kill.

Метод **Run** либо осуществляет обычную для WindowManager последовательность Init-Ask-Kill, либо запускает на выполнение указанную зависимую процедуру в том же самом процессе (здесь под процессом понимается ряд выполняющихся процедур, имеющих общую выделенную область памяти, то есть одни и те же глобальные и локальные переменные, буферы записи файлов для разных процессов будут иметь разные значения). Run возвращает значение, указывающее, выполнено или нет требуемое действие.

Run **Run** осуществляет обычную для WindowManager последовательность Init-Ask-Kill.

Run (*номер, запрос*) Виртуальный метод, выполняющий процедуру, идентифицированную *номером*. Это позволяет другим объектам и шаблонно сгенерированному коду вызывать подчиненные WindowManager'у процедуры как по номеру, так и по имени. Вызванная процедура при этом выполняется в том же процессе, что и вызывающая процедура.

Метод Run является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод Run в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Возвращаемый тип данных: BYTE

Реализация: Метки соответствия возвращаемых значений объявлены в файле \LIBSRC\TPLEQU.CLW следующим образом:

```
RequestCompleted EQUATE (1)          !модернизация завершена
RequestCancelled  EQUATE (2)         ! модернизация отменена
```

Пример:

```
                                     !данные процедуры
 CODE
 ThisWindow.Run                      !обычная последовательность Init-Ask-Kill
ThisWindow.TakeAccepted PROCEDURE()
 CODE                                !текст процедуры
IF SELF.Run(1,SelectRecord) = RequestCompleted
                                    !выполнить процедуру в этом процессе
CLI:StateCode = ST:StateCode
ELSE
SELECT(?CLI:StateCode)
CYCLE
END
BrowseClass.Ask PROCEDURE(BYTE Request)
 CODE                                ! текст процедуры
Response=SELF.Window.Run(SELF.AskProcedure,Request)
                                    ! выполнить процедуру в этом процессе

ThisWindow.Run PROCEDURE            ! выполнить последовательность Init-Ask-Kill
 CODE
 IF SELF.Init() = Level:Benign
```

```
  SELF.Ask
 END
 SELF.Kill
 RETURN GlobalResponse
ThisWindow.Run PROCEDURE(USHORT Number,BYTE Request)
                                        !выполнить зависимую процедуру
 CODE
 GlobalRequest = Request
 EXECUTE Number
  SelectStates
  UpdatePhones
 END
 RETURN GlobalResponse
```

Смотри также: Init, Ask, Kill

SaveHistory (сохранить поле истории для последующего восстановления)

SaveHistory, PROTECTED

SaveHistory метод сохраняет копию определенных методом AddHistoryField полей, предназначенных для восстановления в дальнейшем методом RestoreField.

Этот метод является защищенным, поэтому может вызываться методами класса WindowManager или класса, полученного из WindowManager.

Реализация: Метод AddHistoryFile определяет файл и буфера записи, предназначенные для сохранения и восстановления значений полей. Метод AddHistoryField связывает определенные поля файла истории с соответствующими им элементами управления. SaveHistory метод сохраняет копию полей истории. RestoreField метод восстанавливает содержимое определенного элемента управления.

Пример:

```
WindowManager.TakeCompleted PROCEDURE
 CODE
 SELF.SaveHistory
 CASE SELF.Request
 OF InsertRecord
  DO InsertAction
 OF ChangeRecord
```

```
  DO ChangeAction
 OF DeleteRecord
  DO DeleteAction
 END
```

Смотри также: AddHistoryFile, AddHistoryField, HistoryKey, RestoreField

SetAlerts (подготовка «горячих клавиш» для элементов управления на окне)

SetAlerts, VIRTUAL

Метод **SetAlerts** готовит любые необходимые сочетания клавиш, соответствующие оконному элементу управления, включая те, которые необходимы для клавиши истории окна, окон просмотра и локаторов (устройств ввода символов).

Метод SetAlerts является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод SetAlerts в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: SetAlerts метод вызывает метод BrowseClass.SetAlerts для каждого объекта Browse-класса, добавленного методом AddItem. SetAlerts также подготавливает сочетания клавиш HistoryKey для каждого элемента управления из AddHistoryField.

Обратите внимание, что подготовленные сочетания клавиш связаны только с определенными, задействованными элементами управления, типа окна списка или поля ввода. Эти сочетания клавиш не предназначены для управления окном. См. *ALRT* в *Описании Языка* для получения дополнительной информации.

Пример:

```
ThisWindow.Init PROCEDURE()
 CODE                                !текст процедуры
 SELF.SetAlerts()
 RETURN Level:Benign
```

Смотри также: AddHistoryField, HistoryKey, BrowseClass.SetAlerts

SetResponse (определить для окна обработку по типу OK или Cancel)

SetResponse(*Ответ* **), VIRTUAL**

SetResponse Устанавливает стандартную обработку типа «OK» или «Cancel».

Ответ Целочисленная константа, переменная, метка соответствия, или выражение, определяющее ответ WindowManager (принять или пропустить) на запрашиваемое действие.

SetResponse метод начинает для процедуры стандартную обработку типа «OK» or «Cancel». Он фиксирует результат работы процедуры (закончена нормально или отменена) и переходит к нормальному закрытию процедуры.

Метод SetResponse является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод SetResponse в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод TakeAccepted вызывает метод SetResponse. Метод же SetResponse устанавливает свойство Response и посылает событие EVENT: CloseWindow. Если получен *ответ* RequestCancelled, то метод SetResponse также устанавливает свойство VCRRequest равным VCR: None.

Метки соответствия для возможных значений параметра *ответ* объявлены в файле \LIBSRC\TPLEQU.CLW следующим образом:

```
RequestCompleted  EQUATE (1)          !модернизация завершена нормально
RequestCancelled  EQUATE (2)          ! модернизация прервана
```

Пример:

```
WindowManager.TakeAccepted PROCEDURE
I LONG,AUTO
A SIGNED,AUTO
 CODE
 A = ACCEPTED()                       !procedure code
 LOOP I = 1 TO RECORDS(SELF.Buttons)
  GET(SELF.Buttons,I)
  IF SELF.Buttons.Control = A
   SELF.SetResponse(SELF.Buttons.Action)
   RETURN Level:Notify
  END
 END                                  !procedure code
 RETURN Level:Benign
```

Смотри также: Request, Response

TakeAccepted (действия по обработке события EVENT:Accepted)

TakeAccepted, VIRTUAL, PROC

TakeAccepted метод обрабатывает события EVENT:Accepted для оконных элементов управления и возвращает значение, показывающее, когда обработка окна завершена и ACCEPT-цикл должен остановиться. TakeAccepted возвращает Level:Benign чтобы указать, что обработка этого события должна продолжиться в обычном режиме; Level:Notify - чтобы указать, что обработка этого события закончена и ACCEPT-цикл должен перейти в начало; Level:Fatal - чтобы указать, что событие не может быть обработано и ACCEPT-цикл следует прервать.

Метод TakeAccepted является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeAccepted в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: TakeAccepted carries out HistoryKey and 2 parameter AddItem actions.

Возвращаемые значения объявлены в файле ABERROR.INC.

TakeEvent метод вызывает метод TakeAccepted.

Возвращаемый тип данных: BYTE

Пример:

```
MyWindowManager.TakeEvent PROCEDURE
RVal BYTE(Level:Benign)
I   USHORT,AUTO
 CODE
IF ~FIELD()
RVal = SELF.TakeWindowEvent()
IF RVal THEN RETURN RVal.
END
CASE EVENT()
OF EVENT:Accepted;                    RVal = SELF.TakeAccepted()
OF EVENT:Rejected;                    RVal = SELF.TakeRejected()
```

```
OF EVENT:Selected;                  RVal = SELF.TakeSelected()
OF EVENT:NewSelection;              RVal = SELF.TakeNewSelection()
OF EVENT:Completed;                 RVal = SELF.TakeCompleted()
OF EVENT:CloseWindow OROF EVENT:CloseDown
        RVal = SELF.TakeCloseEvent()
END
IF RVal THEN RETURN RVal.
IF FIELD()
RVal = SELF.TakeFieldEvent()
END
RETURN RVal
```

Смотри также: AddItem, HistoryKey, TakeEvent

TakeCloseEvent (действия при непринятии окна)

TakeCloseEvent, VIRTUAL, PROC

Метод **TakeCloseEvent** обрабатывает события EVENT: CloseWindow и EVENT:CloseDown для окна и возвращает значение, показывающее, когда обработка окна завершена и ACCEPT-цикл должен остановиться.

Когда конечный пользователь не принимает форму модернизации (нажимает кнопку Cancel), TakeCloseEvent производит действия, предписанные по умолчанию. Реальные же действия зависят от значения различных свойств WindowManager, таких как Request, Response, CancelAction, OriginalRequest и т.д.

TakeCloseEvent возвращает Level:Benign чтобы указать, что обработка этого события должна продолжиться в обычном режиме; Level:Notify - чтобы указать, что обработка этого события закончена и ACCEPT-цикл должен перейти в начало; Level:Fatal - чтобы указать, что событие не может быть обработано и ACCEPT-цикл следует прервать.

Метод TakeCloseEvent является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeCloseEvent в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: TakeCloseEvent метод отменяет любые действия, которые надо отменить при непринятии формы (например, отмена автонумерации записей, которая больше не нужна).

Возвращаемые значения объявлены в файле ABERROR.INC.
TakeEvent метод вызывает метод TakeCloseEvent.
Возвращаемый тип данных: BYTE

Пример:

```
MyWindowManager.TakeEvent PROCEDURE
RVal BYTE(Level:Benign)
I   USHORT,AUTO
 CODE
            IF ~FIELD()
                                    RVal = SELF.TakeWindowEvent()
                                    IF RVal THEN RETURN RVal.
            END
            CASE EVENT()
            OF EVENT:Accepted;      RVal = SELF.TakeAccepted()
            OF EVENT:Rejected;      RVal = SELF.TakeRejected()
            OF EVENT:Selected;      RVal = SELF.TakeSelected()
            OF EVENT:NewSelection;  RVal = SELF.TakeNewSelection()
            OF EVENT:Completed;     RVal = SELF.TakeCompleted()
            OF EVENT:CloseWindow OROF EVENT:CloseDown
                                    RVal = SELF.TakeCloseEvent()
            END
            IF RVal THEN RETURN RVal.
            IF FIELD()
                                    RVal = SELF.TakeFieldEvent()
            END
            RETURN RVal
```

Смотри также: CancelAction, Request, Response, OriginalRequest, TakeEvent

TakeCompleted (действия по принятию формы модернизации)

TakeCompleted, VIRTUAL, PROC

Метод **TakeCompleted** обрабатывает событие EVENT:Completed для окна и возвращает значение, показывающее, когда обработка окна завершена и ACCEPT-цикл должен остановиться.

Когда конечный пользователь принимает форму модернизации (нажимает кнопку OK), TakeCompleted производит действия, предписанные по умолчанию. Фактические же действия зависят от значения различных свойств WindowManager, таких как Request, InsertAction, VCRRequest и т.д.

TakeCompleted возвращает Level:Benign чтобы указать, что обработка этого события должна продолжиться в обычном режиме; Level:Notify - чтобы указать, что обработка этого события закончена и ACCEPT-цикл должен перейти в начало; Level:Fatal - чтобы указать, что событие не может быть обработано и ACCEPT-цикл следует прервать.

Метод TakeCompleted является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeCompleted в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Метод TakeCompleted вызывает метод SaveHistory, затем завершает требуемое действие (вставка, изменение или удаление) с учетом различных ограничений по пригодности. Объект FileManager производит проверку полей формы модернизации и проверку параллелизма, а объект RelationManager реализует любые ссылочные ограничения.

TakeCompleted устанавливает свойство Response и, когда это возможно, посылает событие EVENT: CloseWindow.

Возвращаемые значения объявлены в ABERROR.INC.

Метод TakeEvent вызывает метод TakeCompleted.

Возвращаемый тип данных: BYTE

Пример:

```
MyWindowManager.TakeEvent PROCEDURE
RVal BYTE(Level:Benign)
I   USHORT,AUTO
  CODE
IF ~FIELD()
RVal = SELF.TakeWindowEvent()
IF RVal THEN RETURN RVal.
              END
CASE EVENT()
OF EVENT:Accepted;                    RVal = SELF.TakeAccepted()
OF EVENT:Rejected;                    RVal = SELF.TakeRejected()
```

```
OF EVENT:Selected;                    RVal = SELF.TakeSelected()
OF EVENT:NewSelection;                RVal = SELF.TakeNewSelection()
OF EVENT:Completed;                   RVal = SELF.TakeCompleted()
OF EVENT:CloseWindow OROF EVENT:CloseDown
                                      RVal = SELF.TakeCloseEvent()
END
IF RVal THEN RETURN RVal.
IF FIELD()                            RVal = SELF.TakeFieldEvent()
          END
RETURN RVal
```

Смотри также: InsertAction, Request, Response, TakeEvent, VCRRequest

TakeEvent (обработка всех событий)

TakeEvent, VIRTUAL, PROC

Метод **TakeEvent** обрабатывает все события окна и возвращает значение, показывающее, когда обработка окна завершена и ACCEPT-цикл должен остановиться. TakeEvent возвращает Level:Benign чтобы указать, что обработка этого события должна продолжиться в обычном режиме; Level:Notify - чтобы указать, что обработка этого события закончена и ACCEPT-цикл должен перейти в начало; Level:Fatal - чтобы указать, что событие не может быть обработано и ACCEPT-цикл следует прервать.

Метод TakeEvent является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeEvent в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Возвращаемые значения объявлены в ABERROR.INC.

Метод Ask вызывает метод TakeEvent.

Возвращаемый тип данных: BYTE

Пример:

```
WindowManager.Ask PROCEDURE
 CODE
 IF SELF.Dead THEN RETURN .
```

```
 CLEAR(SELF.LastInsertedPosition)
 ACCEPT
  CASE SELF.TakeEvent()
  OF Level:Fatal
   BREAK
  OF Level:Notify
   CYCLE    ! Для быстрого прекращения обработки события Not as dopey at it looks,
it is for 'short-stopping' certain events
  END
 END
```

Смотри также: Ask

TakeFieldEvent (действие по обработке событий, связанных с полем)

TakeFieldEvent, VIRTUAL, PROC

TakeFieldEvent метод – виртуальный метод, предназначенный для обработки всех связанных с полем или элементом управления событий в окне. Он возвращает значение, показывающее, когда обработка окна завершена и ACCEPT-цикл должен остановиться. TakeFieldEvent возвращает Level:Benign чтобы указать, что обработка этого события должна продолжиться в обычном режиме; Level:Notify - чтобы указать, что обработка этого события закончена и ACCEPT-цикл должен перейти в начало; Level:Fatal - чтобы указать, что событие не может быть обработано и ACCEPT-цикл следует прервать.

Метод TakeFieldEvent является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeFieldEvent в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Возвращаемые значения объявлены в ABERROR.INC.
Метод TakeEvent вызывает метод TakeFieldEvent.
Возвращаемый тип данных: BYTE

Пример:

```
MyWindowManager.TakeEvent PROCEDURE
RVal BYTE(Level:Benign)
I   USHORT,AUTO
 CODE
```

```
IF ~FIELD()                    RVal = SELF.TakeWindowEvent()
IF RVal THEN RETURN RVal.
END
CASE EVENT()
OF EVENT:Accepted;             RVal = SELF.TakeAccepted()
OF EVENT:Rejected;             RVal = SELF.TakeRejected()
OF EVENT:Selected;             RVal = SELF.TakeSelected()
OF EVENT:NewSelection;         RVal = SELF.TakeNewSelection()
OF EVENT:Completed;            RVal = SELF.TakeCompleted()
OF EVENT:CloseWindow OROF EVENT:CloseDown
                               RVal = SELF.TakeCloseEvent()
END
IF RVal THEN RETURN RVal.
IF FIELD()
                               RVal = SELF.TakeFieldEvent()
END
RETURN RVal
```

Смотри также: Ask

TakeNewSelection (обработка события EVENT:NewSelection)

TakeNewSelection, VIRTUAL, PROC

Метод **TakeNewSelection** обрабатывает событие EVENT:NewSelection на всех оконных элементах управления и возвращает значение, показывающее, когда обработка окна завершена и ACCEPT-цикл должен остановиться. TakeNewSelection возвращает Level:Benign чтобы указать, что обработка этого события должна продолжиться в обычном режиме; Level:Notify - чтобы указать, что обработка этого события закончена и ACCEPT-цикл должен перейти в начало; Level:Fatal - чтобы указать, что событие не может быть обработано и ACCEPT-цикл следует прервать.

Метод TakeFieldEvent является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeFieldEvent в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: TakeNewSelection переустанавливает WindowManager при выборе конечным пользователем новой закладки.

Возвращаемые значения объявлены в файле ABERROR.INC.

Метод TakeEvent вызывает метод TakeNewSelection.

Возвращаемый тип данных: BYTE

Пример:

```
MyWindowManager.TakeEvent PROCEDURE
RVal BYTE(Level:Benign)
I   USHORT,AUTO
 CODE
            IF ~FIELD()                 RVal = SELF.TakeWindowEvent()
            IF RVal THEN RETURN RVal.
            END
            CASE EVENT()
            OF EVENT:Accepted;          RVal = SELF.TakeAccepted()
            OF EVENT:Rejected;          RVal = SELF.TakeRejected()
            OF EVENT:Selected;          RVal = SELF.TakeSelected()
            OF EVENT:NewSelection;      RVal = SELF.TakeNewSelection()
            OF EVENT:Completed;         RVal = SELF.TakeCompleted()
            OF EVENT:CloseWindow OROF EVENT:CloseDown
                                        RVal = SELF.TakeCloseEvent()
            END
            IF RVal THEN RETURN RVal.
            IF FIELD()
                                        RVal = SELF.TakeFieldEvent()
            END
            RETURN RVal
```

Смотри также: TakeEvent

TakeRejected (обработка события EVENT:Rejected)

TakeRejected, VIRTUAL, PROC

Метод **TakeRejected** обрабатывает событие EVENT:Rejected на оконных элементах управления и возвращает значение, показывающее, когда обработка окна завершена и ACCEPT-цикл должен остановиться. TakeRejected возвращает Level:Benign чтобы указать, что обработка этого события должна продолжиться в обычном режиме; Level:Notify - чтобы указать, что обработка этого события закончена и ACCEPT-цикл должен перейти в начало; Level:Fatal - чтобы указать, что событие не может быть обработано и ACCEPT-цикл следует прервать.

Метод TakeRejected является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeRejected в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: TakeRejected выдает звуковое предупреждение и возвращает фокус на тот элемент управления, с которого пытались уйти.

Возвращаемые значения объявлены в файле ABERROR.INC.

Метод TakeEvent вызывает метод TakeRejected.
Возвращаемый тип данных: BYTE

Пример:

```
MyWindowManager.TakeEvent PROCEDURE
RVal BYTE(Level:Benign)
I   USHORT,AUTO
 CODE
            IF ~FIELD()                     RVal = SELF.TakeWindowEvent()
            IF RVal THEN RETURN RVal.
            END
            CASE EVENT()
            OF EVENT:Accepted;              RVal = SELF.TakeAccepted()
            OF EVENT:Rejected;              RVal = SELF.TakeRejected()
            OF EVENT:Selected;              RVal = SELF.TakeSelected()
            OF EVENT:NewSelection;          RVal = SELF.TakeNewSelection()
            OF EVENT:Completed;             RVal = SELF.TakeCompleted()
            OF EVENT:CloseWindow OROF EVENT:CloseDown
                                            RVal = SELF.TakeCloseEvent()
            END
            IF RVal THEN RETURN RVal.
            IF FIELD()
                                            RVal = SELF.TakeFieldEvent()
            END
            RETURN RVal
```

Смотри также: TakeEvent

TakeSelected (обработка события EVENT:Selected)

TakeSelected, VIRTUAL, PROC

Метод **TakeSelected** предназначен для обработки события EVENT:Selected на оконных элементах управления и возвращает значение, показывающее, когда обработка окна завершена и ACCEPT-цикл должен остановиться. TakeSelected возвращает Level:Benign чтобы указать, что обработка этого события должна

продолжиться в обычном режиме; Level:Notify - чтобы указать, что обработка этого события закончена и ACCEPT-цикл должен перейти в начало; Level:Fatal - чтобы указать, что событие не может быть обработано и ACCEPT-цикл следует прервать.

Метод TakeSelected является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeSelected в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Возвращаемые значения объявлены в файле ABERROR.INC.
Метод TakeEvent вызывает метод TakeSelected.
Возвращаемый тип данных: BYTE

Пример:

```
MyWindowManager.TakeEvent PROCEDURE
RVal BYTE(Level:Benign)
I   USHORT,AUTO
 CODE
            IF ~FIELD()                 RVal = SELF.TakeWindowEvent()
            IF RVal THEN RETURN RVal.
            END
            CASE EVENT()
            OF EVENT:Accepted;          RVal = SELF.TakeAccepted()
            OF EVENT:Rejected;          RVal = SELF.TakeRejected()
            OF EVENT:Selected;          RVal = SELF.TakeSelected()
            OF EVENT:NewSelection;      RVal = SELF.TakeNewSelection()
            OF EVENT:Completed;         RVal = SELF.TakeCompleted()
            OF EVENT:CloseWindow OROF EVENT:CloseDown
                                        RVal = SELF.TakeCloseEvent()
            END
            IF RVal THEN RETURN RVal.
            IF FIELD()
                                        RVal = SELF.TakeFieldEvent()
            END
            RETURN RVal
```

Смотри также: TakeEvent

TakeWindowEvent (обработка не связанных с полем событий)

TakeWindowEvent, VIRTUAL, PROC

Метод **TakeWindowEvent** обрабатывает все не связанные с полем или элементом управления события в окне. Он возвращает значение, показывающее, когда обработка окна завершена и ACCEPT-цикл должен остановиться. TakeWindowEvent возвращает Level:Benign чтобы указать, что обработка этого события должна продолжиться в обычном режиме; Level:Notify - чтобы указать, что обработка этого события закончена и ACCEPT-цикл должен перейти в начало; Level:Fatal - чтобы указать, что событие не может быть обработано и ACCEPT-цикл следует прервать.

Метод TakeWindowEvent является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeWindowEvent в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: TakeWindowEvent осуществляет стандартную обработку таких событий, как EVENT: OpenWindow (Open метод), EVENT:LoseFocus, EVENT:GainFocus (Reset метод), и EVENT:Sized (WindowResizeClass.Resize метод).

Возвращаемые значения объявлены в ABERROR.INC.

Метод TakeEvent вызывает метод TakeWindowEvent.

Возвращаемый тип данных: BYTE

Пример:

```
MyWindowManager.TakeEvent PROCEDURE
RVal BYTE(Level:Benign)
I   USHORT,AUTO
 CODE
            IF ~FIELD()                 RVal = SELF.TakeWindowEvent()
            IF RVal THEN RETURN RVal.
            END
            CASE EVENT()
            OF EVENT:Accepted;          RVal = SELF.TakeAccepted()
            OF EVENT:Rejected;          RVal = SELF.TakeRejected()
            OF EVENT:Selected;          RVal = SELF.TakeSelected()
         OF EVENT:NewSelection;         RVal = SELF.TakeNewSelection()
```

```
    OF EVENT:Completed;       RVal = SELF.TakeCompleted()
    OF EVENT:CloseWindow OROF EVENT:CloseDown
                              RVal = SELF.TakeCloseEvent()
    END
    IF RVal THEN RETURN RVal.
    IF FIELD()
                              RVal = SELF.TakeFieldEvent()
    END
    RETURN RVal
```

Смотри также: Open, Reset, TakeEvent, WindowResizeClass.Resize

Update (подготовка строк данных к записи на диск)

Update, VIRTUAL

Метод **Update** подготавливает строки данных в структурах FILE и VIEW WindowManager'а для записи на диск, синхронизируя содержимое буфера со значениями соответствующих полей на экране. Метод Update также производит автоматическую проверку параллелизма, так что попытка записи на диск возвращает ошибку в случае, если другой пользователь изменил данные с тех пор, как они были получены.

Метод Update является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод Update в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод Update вызывает BrowseClass.UpdateViewRecord для каждого объекта Browse-класса, добавленного методом AddItem.

Пример:

```
ThisWindow.TakeAccepted PROCEDURE()
Looped BYTE
 CODE
 LOOP
  IF Looped
   RETURN Level:Notify
  ELSE
   Looped = 1
  END
  PARENT.TakeAccepted()
  CASE ACCEPTED()
  OF ?Expand
   ThisWindow.Update
```

```
  ?CusTree{PropList:MouseDownRow} = CHOICE(?CusTree)
  DO REL1::ExpandAll
OF ?Contract
  ThisWindow.Update
  ?CusTree{PropList:MouseDownRow} = CHOICE(?CusTree)
  DO REL1::ContractAll
OF ?Insert
  ThisWindow.Update
  ?CusTree{PropList:MouseDownRow} = CHOICE(?CusTree)
  DO REL1::AddEntry
OF ?Change
  ThisWindow.Update
  ?CusTree{PropList:MouseDownRow} = CHOICE(?CusTree)
  DO REL1::EditEntry
OF ?Delete
  ThisWindow.Update
  ?CusTree{PropList:MouseDownRow} = CHOICE(?CusTree)
  DO REL1::RemoveEntry
END
RETURN Level:Benign
```

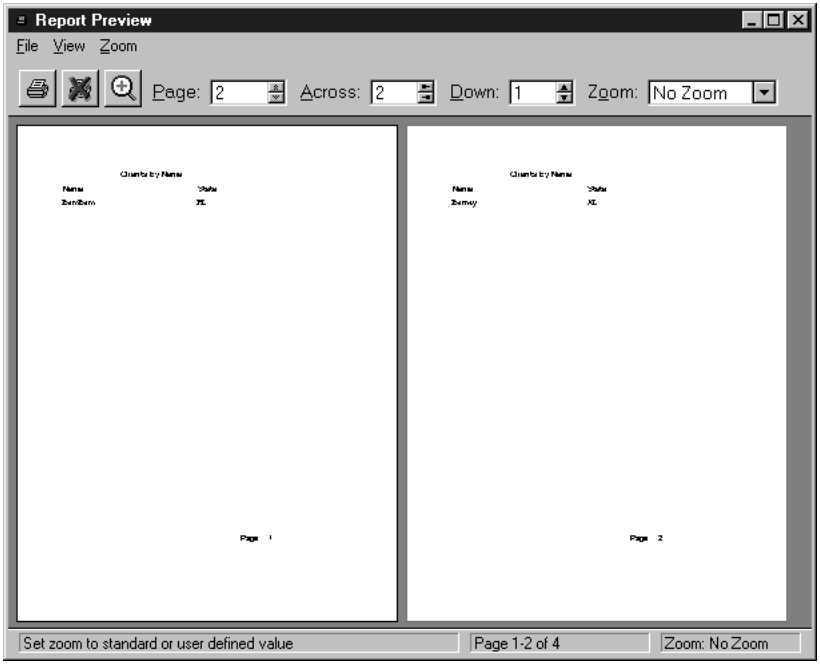
Класс Предварительного просмотра Печати

Краткий обзор

Концепции PrintPreview-класса

PrintPreview-класс представляет собой WindowManager, который реализует полностью оснащенное окно предварительного просмотра печати. Оно предоставляет возможность уменьшения и увеличения масштаба изображения с различными уровнями изменения масштаба, возможность выборочной и последовательной постраничной навигации, а также просмотр каждой страницы отчета с использованием указателя. Вы даже можете сами задать количество строк и столбцов в указателе, которое должно быть отображено.

Когда Вы закончите просмотр отчета, его можно тут же послать принтеру для немедленной печати по принципу Что-Вы-Видите-То-Вы-И-Получаете (WYSIWYG).



PrintPreview-класс представляет отчет, подготовленный для предварительного просмотра, в виде метафайлов Windows (.WMF), по одному для каждой страницы отчета. Для генерации этих метафайлов достаточно определить атрибут PREVIEW, и тогда все остальные действия будут произведены шаблонами Отчета Clarion'a. См. PREVIEW в *Описании Языка* для получения дополнительной информации, а для дополнительной информации о шаблонах Отчета см. *Процедурные Шаблоны— Отчет.*

Связь с другими Классами разработки приложений

PrintPreview-класс порожден из класса WindowManager (см. *Класс Менеджера Окна* для получения дополнительной информации).

PrintPreview-класс при выполнении некоторых задач использует Popup-класс и, опционально, Translator-класс. Поэтому, если в Вашей программе порождается PrintPreview-класс, то должен быть также порожден и Popup-класс, а также, возможно, потребуется порождение и Translator-класса. Большая часть этих действий выполняется автоматически, стоит Вам только включить заголовок PrintPreview-класса (файл ABREPORT.INC) в секцию данных вашей программы. См. *Пояснительный Пример*.

Реализация ABC шаблона

Процедурный Шаблон Отчета и шаблон Утилиты Мастера Отчета автоматически генерируют весь код и включают все классы, необходимые для обеспечения отчетов, содержащихся в Вашем приложении, средством предварительного просмотра печати.

Эти шаблоны Отчета порождают объект PrintPreview-класса, именуемый Previewer, для *каждой* процедуры отчета из приложения. Этот объект поддерживает все функциональные возможности, указанные в секции **Опции Предварительного просмотра (Preview Options)** окна диалога **Свойства Отчета** шаблона отчета. См. *Процедурные Шаблоны— Отчет* для получения дополнительной информации.

Объектом Previewer управляет произведенный с помощью шаблонов объект ReportManager (ThisWindow), так что, в общем случае, для начального конфигурирования свойств Previewer'a необходимы лишь ссылки на объект Previewer внутри произведенного с помощью шаблонов кода.

Файлы-источники PrintPreview-класса

Файлы с исходным кодом PrintPreview-класса по умолчанию установлены в директорию \CLARION4\LIBSRC. Собственно файлы PrintPreview-класса и их

соответствующие компоненты представлены ниже:

ABREPORT.INC	Объявления PrintPreview-класса
ABREPORT.CLW	Определения методов PrintPreview-класса
ABREPORT.TRN	Текст пользовательского интерфейса PrintPreview-класса

Конфигурирование изменения масштаба изображения

Текст интерфейса пользователя и стандартный выбор вариантов изменения масштаба изображения, которые показываются PrintPreview-классом на этапе выполнения, определены в файле ABREPORT.TRN. Для изменения или настройки этого текста или стандартных вариантов изменения масштаба изображения, необходимо просто отредактировать файл ABREPORT.TRN в соответствии с Вашими потребностями, сделав предварительно его резервную копию. См. *ZoomIndex* для получения дополнительной информации.

Пояснительный Пример

Приведенный ниже пример показывает типовую последовательность операторов, предназначенных для объявления, порождения, инициализации, использования и завершения объекта PrintPreview-класса, а также некоторых связанных с ним объектов.

В этом примере объект PrintPreview-класса используется для предварительного просмотра очень простого отчета до того, как его напечатать. Программа определяет начальное положение и размер окна предварительного просмотра печати и позволяет настраивать масштаб изображения.

```
 PROGRAM
  INCLUDE('ABREPORT.INC')                !объявить ReportManager
                                         ! и PrintPreviewClass
  MAP
  END
GlobalErrors  ErrorClass
VCRRequest LONG(0),THREAD
Customer
FILE,DRIVER('TOPSPEED'),PRE(CUS),THREAD
BYNUMBER
KEY(CUS:CUSTNO),NOCASE,OPT,PRIMARY
Record      RECORD,PRE()
CUSTNO    LONG
```

```
Name        STRING(30)
State       STRING(2)
END
END
Access:Customer CLASS(FileManager)      !объявить объект Access:Customer
Init        PROCEDURE
END
Relate:Customer CLASS(RelationManager)! объявить объект Relate:Customer
Init PROCEDURE
END
CusView     VIEW(Customer)                  ! объявить VIEW-структуру CusView
END
PctDone     BYTE                            ! Переменная отображения хода процесса
                                            !report
REPORT,AT(1000,1542,6000,7458),PRE(RPT),FONT('Arial',10,,),THOUS
HEADER,AT(1000,1000,6000,542),FONT(,,,FONT:bold)
STRING('Customers'),AT(2000,20),FONT(,14,,)
STRING('Id'),AT(52,313),TRN
STRING('Name'),AT(2052,313),TRN
STRING('State'),AT(4052,313),TRN
END
detail      DETAIL,AT(,,6000,281),USE(?detail)
STRING(@n-14),AT(52,52),USE(CUS:CUSTNO)
STRING(@s30),AT(2052,52),USE(CUS:NAME)
STRING(@s2),AT(4052,52),USE(CUS:State)
END
FOOTER,AT(1000,9000,6000,219)
STRING(@pPage <<<#p),AT(5250,31),PAGENO,USE(?PageCount)
END
END
ProgressWindow
WINDOW('Progress...'),AT(,,142,59),CENTER,TIMER(1),GRAY,DOUBLE
PROGRESS,USE(PctDone),AT(15,15,111,12),RANGE(0,100)
STRING(''),AT(0,3,141,10),USE(?UserString),CENTER
STRING(''),AT(0,30,141,10),USE(?TxtDone),CENTER
BUTTON('Cancel'),AT(45,42),USE(?Cancel)
END
ThisProcedure CLASS(ReportManager)      ! объявить объект ThisProcedure
Init
PROCEDURE(),BYTE,PROC,VIRTUAL
Kill
PROCEDURE(),BYTE,PROC,VIRTUAL
END
```

```
CusReport CLASS(ProcessClass)          ! объявить объект CusReport
TakeRecord PROCEDURE(),BYTE,PROC,VIRTUAL
END
Previewer   PrintPreviewClass          ! объявить объект Previewer
                                       ! для использования совместно с
ThisProcedure
 CODE
 ThisProcedure.Run()                   !выполнить процедуру
ThisProcedure.Init PROCEDURE()         !инициализировать ThisProcedure
ReturnValue BYTE,AUTO
 CODE
 GlobalErrors.Init
 Relate:Customer.Init
 ReturnValue = PARENT.Init()
 SELF.FirstField = ?PctDone
 SELF.VCRRequest &= VCRRequest
 SELF.Errors &= GlobalErrors
 Relate:Customer.Open
 OPEN(ProgressWindow)
 SELF.Opened=True
  CusReport.Init(CusView,Relate:Customer,?TxtDone,PctDone,RECORDS(Customer))
 CusReport.AddSortOrder(CUS:BYNUMBER)
 SELF.AddItem(?Cancel,RequestCancelled)
 SELF.Init(CusReport,report,Previewer)          !зарегистрировать Previewer с
                                                !ThisProcedure
 SELF.Zoom = PageWidth
 Previewer.AllowUserZoom=True                   !разрешить изменение масштаба
                                                !изображения
 Previewer.Maximize=True                        !начальная максимизация окна
                                                !предварительного просмотра
 SELF.SetAlerts()
 RETURN ReturnValue
ThisProcedure.Kill    ьPROCEDURE()
ReturnValue      BYTE,AUTO
 CODE
 ReturnValue = PARENT.Kill()
 Relate:Customer.Close
 Relate:Customer.Kill
 GlobalErrors.Kill
 RETURN ReturnValue
CusReport.TakeRecord    PROCEDURE()
ReturnValue             BYTE,AUTO
```

```
SkipDetails BYTE
  CODE
  ReturnValue = PARENT.TakeRecord()
  PRINT(RPT:detail)
  RETURN ReturnValue
Access:Customer.Init  PROCEDURE
  CODE
  PARENT.Init(Customer,GlobalErrors)
  SELF.FileNameValue = 'Customer'
  SELF.Buffer &= CUS:Record
  SELF.Create = 0
  SELF.LazyOpen = False
  SELF.AddKey(CUS:BYNUMBER,'CUS:BYNUMBER',0)
Relate:Customer.Init  PROCEDURE
  CODE
  Access:Customer.Init
  PARENT.Init(Access:Customer,1)
```

Свойства PrintPreview-класса

PrintPreview-класс содержит свойства, которые главным образом делают возможным конфигурирование окна предварительного просмотра печати и определение его характеристик. Свойства PrintPreview-класса описаны ниже.

AllowUserZoom (разрешить какой-нибудь вариант изменения масштаба изображения)

AllowUserZoom	BYTE

AllowUserZoom свойство указывает, допускает ли объект PrintPreview-класса возможность изменения масштаба изображения конечным пользователем. Изменение масштаба изображения позволяет ему использовать любой вариант изменения масштаба изображения. Если же такая возможность не предоставлена, то пользователь может применять только предложенные стандартные варианты изменения масштаба изображения.

Свойство ZoomIndex указывает, изменяет ли пользователь варианты изменения масштаба изображения, или использует стандартные.

Реализация: Значение единица (1) допускает возможность изменения масштаба пользователем; значение же ноль (0) - запрещает. Конкретный вариант изменения масштаба изображения пользователем содержится в свойстве UserPercentile.

Смотри также: UserPercentile, ZoomIndex

CurrentPage (выбранная страница отчета)

CurrentPage LONG

Свойство **CurrentPage** содержит номер выбранной страницы отчета. Объект PrintPreview-класса использует это свойство для того, чтобы выдвинуть на первый план выбранную страницу отчета в случае отображения на экране более чем одной страницы, для постраничной навигации, а также для показа номера текущей страницы конечному пользователю.

Maximize (количество отображаемых страниц по горизонтали)

Maximize BYTE

Свойство **Maximize** указывает, следует ли максимизировать окно предварительного просмотра при открытии. Значение единица (1 или Истина) максимизирует окно; значение ноль (0 или Ложь) открывает окно в соответствии со свойством WindowSizeSet.

Смотри также: WindowSizeSet

PagesAcross (количество страниц, отображаемых по горизонтали)

PagesAcross USHORT

PagesAcross свойство содержит количество страниц, отображаемых объектом PrintPreview-класса *по горизонтали* внутри окна предварительного просмотра. Объект PrintPreview-класса использует это свойство для подсчета соответствующего положения и размера каждой страницы при отображении нескольких страниц одновременно.

Объект PrintPreview-класса показывает значение PagesAcross во время выполнения, а также позволяет устанавливать это значение конечному пользователю.

PagesDown (количество отображаемых страниц по вертикали)

PagesDown USHORT

Свойство **PagesDown** содержит количество страниц указателя, отображаемых объектом PrintPreview-класса *по вертикали* внутри окна предварительного просмотра. Объект PrintPreview-класса использует это свойство для подсчета соответствующего положения и размера каждой страницы при отображении

нескольких страниц одновременно.

Объект PrintPreview-класса показывает значение PagesDown во время выполнения, а также позволяет устанавливать это значение конечному пользователю.

UserPercentile (текущий вариант изменения масштаба изображения)

UserPercentile USHORT

UserPercentile свойство содержит определенный пользователем вариант изменения масштаба изображения. Объект PrintPreview-класса запрашивает этот вариант у конечного пользователя и применяет его к выбранной странице отчета в том случае, когда значение свойства AllowUserZoom - Истина. Метод SetZoomPercentile устанавливает свойство UserPercentile.

Смотри также: AllowUserZoom, SetZoomPercentile

WindowPosSet (использовать не заданное по умолчанию начальное положение окна предварительного просмотра)

WindowPosSet BYTE

Свойство **WindowPosSet** содержит значение, показывающее, определено ли для окна предварительного просмотра печати какое-нибудь конкретное, а не по умолчанию, начальное положение. Объект PrintPreview-класса использует это свойство для определения начального положения окна предварительного просмотра печати.

Реализация: Метод SetPosition определяет значение этого свойства. Значение единица (1) указывает, что используется заданное начальное положение, а не определенное по умолчанию; значение же ноль (0) говорит о том, что не определено никакого начального положения, а используется то, которое задано по умолчанию.

Смотри также: SetPosition

WindowSizeSet (использовать не заданный по умолчанию начальный размер окна предварительного просмотра)

WindowSizeSet BYTE

Свойство **WindowSizeSet** содержит значение, показывающее, определен ли для окна предварительного просмотра печати какой- нибудь конкретный, а не по умолчанию, начальный размер. Объект PrintPreview-класса использует это свойство

для определения начального размера окна предварительного просмотра печати.

Реализация: Метод SetPosition определяет значение этого свойства. Значение единица (1) указывает, что используется заданный начальный размер, а не определенный по умолчанию; значение же ноль (0) говорит о том, что не определено никакого начального размера, а, следовательно, используется тот, который задан по умолчанию.

Смотри также: SetPosition

ZoomIndex (индекс варианта изменения масштаба изображения)

ZoomIndex BYTE

Свойство **ZoomIndex** содержит значение, указывающее, какой вариант изменения масштаба изображения применен. Объект PrintPreview-класса использует это свойство для идентификации и применения выбранного варианта изменения масштаба изображения. Метод SetZoomPercentile определяет свойство ZoomIndex.

Реализация: Значение ZoomIndex указывает или на один из 7 стандартных вариантов изменения масштаба, или же на вариант, содержащий определенные пользователем настройки изменения масштаба изображения. Объект PrintPreview-класса устанавливает значение ZoomIndex при выборе конечным пользователем желаемого варианта либо из меню, либо из комбинированного окна списка вариантов изменения масштаба. Стандартные варианты изменения масштаба изображения определены в файле ABREPORT.TRN следующим образом:

No Zoom	Отображает в окне предварительного просмотра выровненные по краям страницы, количество которых определено свойствами PagesAcross и PagesDown.
Page Width	Отображает одну-единственную страницу, ширина которой равна ширине окна просмотра.
50%	Отображает одну страницу в 50% ее натурального размера при печати.
75%	Отображает одну страницу в 75% ее натурального размера при печати.
100%	Отображает одну страницу в 100% ее натурального размера при печати.
200%	Отображает одну страницу в 200% ее натурального размера при печати.
300%	Отображает одну страницу в 300% ее натурального размера при печати.

Значение ZoomIndex, равное нулю (0), указывает на то, что определен нестандартный вариант изменения масштаба изображения. Нестандартный вариант изменения масштаба изображения, носителем которого является свойство UserPercentile, может быть определен в том случае, когда свойство AllowUserZoom - Истина.

Смотри также: AllowUserZoom, PagesAcross, PagesDown, UserPercentile, SetZoomPercentile

Методы PrintPreview-класса

PrintPreview-класс содержит методы, описанные ниже.

Функциональная Организация — Предполагаемое Использование

Для лучшего понимания PrintPreview-класса будет полезно организовать его различные методы в две больших категории согласно их предполагаемому использованию — первичный интерфейс и виртуальные методы. Такая организация, по мнению разработчиков, наиболее полно отражает типовое использование методов PrintPreview-класса.

Первичные Интерфейсные Методы

Первичные интерфейсные методы, вызов которых, скорее всего, будет иметь место в Вашей программе, могут быть далее разделены на три категории:

Одноразовое применение:

Init[V]	инициализировать объект PrintPreview-класса
SetPosition	установить начальные координаты окна предварительного просмотра
Display[V]	просмотр отчета
Kill[V]	завершить объект PrintPreview -класса

Используются изредка:

SetINIManager	сохранить и восстановить координаты окна
SetPosition	установить начальные координаты окна предварительного просмотра
SetZoomPercentile	установить определенный пользователем или стандартный вариант изменения масштаба

[V] Эти методы также являются виртуальными.

Виртуальные Методы

В основном Вам не придется прямо вызывать эти методы, поскольку они

вызываются методом Display. Однако у Вас наверняка часто будет возникать потребность в переопределении этих методов, а поскольку они являются виртуальными, то сделать это достаточно просто. Эти методы уже по умолчанию обеспечивают «разумное» поведение, в случае, если Вы не переопределите их.

Init	инициализировать объект PrintPreview-класса
AskPage	предложение новой страницы отчета
AskThumbnails	предложение новой конфигурации указателя
Display	просмотр отчета
Open	подготовить окно предварительного просмотра к отображению
TakeAccepted	обработать события EVENT:Accepted
TakeEvent	обработать все события
TakeFieldEvent	действие по обработке событий, связанных с полем
TakeWindowEvent	обработать не связанные с полем события
Kill	закрыть объект PrintPreview -класса

AskPage (предложение новой страницы отчета)

AskPage, PROC, VIRTUAL, PROTECTED

AskPage метод предлагает конечному пользователю показать новую страницу отчета и возвращает значение, указывающее, отобрана ли новая страница. Возвращаемое значение единица (1) указывает на то, что новая страница выбрана и необходима перерисовка экрана; значение же ноль (0) говорит о том, что новой страницы выбрано не было, и перерисовывать экран не надо.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Метод AskPage является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод AskPage в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод защищен, поэтому он может быть вызван только либо методом PrintPreview-класса, либо методом в классе, полученном из PrintPreview-класса.

Реализация: Метод PrintPreviewClass.Display вызывает метод AskPage. AskPage метод, в свою очередь, показывает диалог, который приглашает на определенную страницу отчета.

Возвращаемый тип данных: BYTE

Пример:

```
!Фактическая реализация метода AskPage: упрощенная версия без переводчика…
PrintPreviewClass.AskPage FUNCTION
JumpPage LONG,AUTO
RVal    BOOL(False)
JumpWin   WINDOW('Jump to Page'),AT(,,181,26),CENTER,GRAY,DOUBLE
PROMPT('&Page:'),AT(5,8),USE(?JumpPrompt)
SPIN(@n5),AT(30,7),USE(JumpPage),RANGE(1,10),STEP(1)
BUTTON('OK'),AT(89,7),USE(?OKButton),DEFAULT
BUTTON('Cancel'),AT(134,7),USE(?CancelButton)
            END
            CODE
            JumpPage=SELF.CurrentPage
            OPEN(JumpWin)
            ACCEPT
            CASE EVENT()
            OF EVENT:OpenWindow
?JumpPage{PROP:RangeHigh}=RECORDS(SELF.ImageQueue)
            OF EVENT:Accepted
            CASE ACCEPTED()
            OF ?OKButton
            IF JumpPage NOT=SELF.CurrentPage
            RVal=True                    !SELF.CurrentPage изменен
            SELF.CurrentPage=JumpPage
            END
            POST(EVENT:CloseWindow)
            OF ?CancelButton
            POST(EVENT:CloseWindow)
            .                            .   .
  CLOSE(JumpWin)
  RETURN RVal
```

AskThumbnails (предложение новой конфигурации указателя)

AskThumbnails, VIRTUAL, PROTECTED

AskThumbnails метод предлагает конечному пользователю выбрать количество страниц для отображения по горизонтали и по вертикали в окне предварительного просмотра.

Метод AskThumbnails является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод AskThumbnails в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод защищен, поэтому он может быть вызван только либо методом PrintPreview-класса, либо методом в классе, полученном из PrintPreview-класса.

Реализация: Метод PrintPreviewClass.Display вызывает метод AskThumbnails. AskThumbnails метод , в свою очередь, показывает диалог, который предлагает выбрать количество страниц для отображения по горизонтали и по вертикали.

Пример:

```
! Фактическая реализация метода AskThumbnails
! немного упрощенная версия без всякого переводчика…
PrintPreviewClass.AskThumbnails PROCEDURE
SelectWindow WINDOW('Pages Displayed'),AT(,,141,64),GRAY,DOUBLE
  GROUP('Across'),AT(7,10,62,32),BOXED
       SPIN(@N2),AT(13,22,15),USE(SELF.PagesAcross,,?PagesAcross),RANGE(1,10)
      END
      GROUP('Down'),AT(72,10,62,32),BOXED
       SPIN(@N2),AT(79,22,15),USE(SELF.PagesDown,,?PagesDown),RANGE(1,10)
      END
     BUTTON('OK'),AT(98,47,40,14),KEY(EnterKey),USE(?OK)
   END
  CODE
  OPEN(SelectWindow)
  ACCEPT
   CASE EVENT()
   OF EVENT:Accepted
    CASE FIELD()
    OF ?OK
     IF SELF.PagesAcross*SELF.PagesDown>RECORDS(SELF.ImageQueue)
      SELECT(?PagesAcross)
     ELSE
      POST(EVENT:CloseWindow)
     END
    END
   END
  END
  CLOSE(SelectWindow)
```

Display (предварительный просмотр отчета)

Display([*zoom*] [,*страница*] [, *поперек*] [, *вниз*] **), VIRTUAL, PROC**

Display	Отображает метафайл, содержащий изображение отчета.
zoom	Целочисленная константа, переменная, метка соответствия, или выражение,

содержащее начальный вариант изменения масштаба изображения для предварительного просмотра печати. Если параметр опущен, метод Display использует определенный по умолчанию в файле ABREPORT.TRN вариант изменения масштаба изображения.

Страница Целочисленная константа, переменная, метка соответствия, или выражение, содержащее начальный номер страницы, подлежащей отображению. Если параметр опущен, значение параметра *страница* по умолчанию считается равным единице (1).

Поперек Целочисленная константа, переменная, метка соответствия, или выражение, содержащее количество страниц по горизонтали при начальном отображении окна предварительного просмотра печати. Если параметр опущен, значение параметра *поперек* по умолчанию считается равным единице (1).

Вниз Целочисленная константа, переменная, метка соответствия, или выражение, содержащее количество страниц по вертикали при начальном отображении окна предварительного просмотра печати. Если параметр опущен, значение параметра *вниз* по умолчанию считается равным единице (1).

Метод **Display** отображает метафайлы, содержащие изображения страниц отчета, и возвращает значение, указывающее, следует ли их напечатать. Возвращаемое значение единица (1 или Истина) говорит о том, что конечный пользователь запросил печать отчета; значение же ноль (0 или Ложь) - что запроса на печать отчета от конечного пользователя не поступало.

Метод Display руководит всем механизмом предварительного просмотра печати, обеспечивая навигацию по отчету, изменение масштаба изображения, конфигурацию окна просмотра. Он также предоставляет возможность непосредственной печати отчета (без предварительного просмотра).

Метод Display является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод Display в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Метод Display объявляет окно предварительного просмотра, а затем, для его отображения и обработки событий на нем, вызывает метод WindowManager.Ask.

Метки соответствия для различных значений параметра *zoom* объявлены в файле ABREPORT.INC следующим образом:

```
NoZoom      EQUATE(-2)
PageWidth    EQUATE(-1)
```

В дополнение к значениям, определенным метками соответствия, можно также определить любой другой целочисленный вариант изменения масштаба изображения, типа 50 (50 % реального размера) или 200 (200 % реального размера).

Возвращаемый тип данных: BYTE

Пример:

```
IF ReportCompleted                    !если не было отмены отчета
ENDPAGE(report)                       ! закончить подготовку отчета
IF PrtPrev.Display()                  !просмотреть отчет «в реальном масштабе
                                      !времени»
report{PROP:FlushPreview} = True      !и, при соответствующем запросе
                                      !пользователя, напечатать его
        END
        END
```

Смотри также: WindowManager.Ask

Init (Инициализировать объект PrintPreview-класса)

Init(*Очередь_изображений)***), VIRUAL**

Init	Инициализирует объект PrintPreview-класса.
Очередь_изображений	Метка очереди, содержащей имена метафайлов с изображениями страниц отчета. Для получения дополнительной информации о них обратитесь к главе *PRE VIEW* в *Описании Языка*.

Метод **Init** инициализирует объект PrintPreview-класса.

Метод Init является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод Init в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод PrintPreviewClass.Init порождает объект Popup-класса для объекта PrintPreview-класса, используя определенный в файле ABREPORT.TRN текст меню.

Параметр *Очередь_изображений* определяет очередь такой же структуры, что и очередь PreviewQueue, объявленная в файле \ABREPORT.INC следующим образом:

```
PreviewQueue    QUEUE,TYPE
Filename        STRING(128)
          END
Пример:
PrintPreviewQueue PreviewQueue          !объявить очередь изображений отчета
PrtPrev
PrintPreviewClass                       !объявить объект PrtPrev
CODE
PrtPrev.Init(PrintPreviewQueue)         !инициализировать объект PrtPrev
             ! текст программы
             PrtPrev.Kill               !закрыть объект PrtPrev
```

Kill (закрыть объект PrintPreview-класса)

Kill, VIRTUAL, PROC

Метод **Kill** освобождает любую память, размещенную в течение жизни объекта, а также исполняет любой другой требуемый код завершения. Он возвращает значение, показывающее статус завершения.

Метод Kill является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод Kill в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Метод Kill вызывает метод WindowManager.Kill и возвращает значение Level:Benign для сигнализации о нормальном завершении. Метки соответствия для различных вариантов возвращаемого значения объявлены в файле ABERROR.INC.

Возвращаемый тип данных: BYTE

Пример:

```
PrintPreviewQueue PreviewQueue      !объявить очередь изображений отчета
                                    !PrtPrev
PrintPreviewClass                   !объявить объект PrtPrev
```

```
CODE
PrtPrev.Init(PrintPreviewQueue)          !инициализировать объект PrtPrev
          ! текст программы
          PrtPrev.Kill                   !закрыть объект PrtPrev
```

Смотри также: WindowManager.Kill

Open (подготовить окно предварительного просмотра к отображению)

Open, VIRTUAL

Метод **Open** готовит окно PrintPreview-класса к начальному показу. Он предназначен для обработки при открытии окна таких событий, как EVENT:OpenWindow и EVENT:GainFocus.

Метод Open является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод Open в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод Open устанавливает начальный размер и положение окна, делает доступными или нет для выбора те элементы управления, для которых это необходимо, а также определяет конфигурацию вариантов изменения масштаба.

Метод WindowManager.TakeWindowEvent вызывает метод Open.

Пример:

```
ThisWindow.TakeWindowEvent          PROCEDURE
 CODE
 CASE EVENT()
 OF EVENT:OpenWindow
  IF ~BAND(SELF.Inited,1)
   SELF.Open
  END
 OF EVENT:GainFocus
  IF BAND(SELF.Inited,1)
   SELF.Reset
  ELSE
   SELF.Open
  END
 END
 RETURN Level:Benign
```

Смотри также: WindowManager.TakeWindowEvent

SetINIManager (сохранить и восстановить координаты окна)

SetINIManager(*INI_менеджер* **)**

SetINIManager	Делает возможным сохранение и восстановление положения и размеров окна предварительного просмотра в промежутке между его использованием.
INI_менеджер	Метка объекта INI- класса, который сохраняет и восстанавливает оконные координаты. См. *INI Класс* для получения дополнительной информации.

SetINIManager метод объявляет объект INI-класса, предназначенный для сохранения и восстановления координат окна в промежутке между его использованием.

Реализация: Метод Open для восстановления начальных размеров и положения окна использует *INI_менеджер*. Для сохранения же этих параметров *INI_менеджер* используется методом TakeEvent.

Пример:

```
ThisWindow.Init PROCEDURE()
 CODE      !текст программы
  ThisWindow.Init(Process,report,Previewer)
  Previewer.SetINIManager(INIMgr)
```

Смотри также: Open, TakeEvent

SetPosition (установить начальные координаты окна предварительного просмотра)

SetPosition([*x*] [,*y*] [, *ширина*] [, *высота*] **)**

SetPosition	Установить начальное положение и размер окна предварительного просмотра печати.
x	Целочисленная константа, переменная, метка соответствия, или выражение, содержащее начальное положение окна предварительного просмотра печати по горизонтали. Если параметр опущен, окно открывается в месте, определенном Windows по умолчанию.
y	Целочисленная константа, переменная, метка соответствия, или выражение, содержащее начальное положение окна предварительного просмотра печати по вертикали. Если параметр опущен, окно открывается в месте, определенном Windows по умолчанию.
Ширина	Целочисленная константа, переменная, метка соответствия, или выражение, содержащее начальную ширину окна предварительного просмотра печати.

Если параметр опущен, его ширина определяется по умолчанию.

Высота Целочисленная константа, переменная, метка соответствия, или выражение, содержащее начальную высоту окна предварительного просмотра печати. Если параметр опущен, его высота определяется по умолчанию.

Метод **SetPosition** определяет начальное положение и размер окна предварительного просмотра печати.

Реализация: Метод SetPosition устанавливает свойства WindowPosSet и WindowSizeSet.

Ширина и высота окна предварительного просмотра печати, принимаемая по умолчанию, закладывается при определении метода Display.

Пример:

```
PrtPrev.SetPosition(1,1,300,250)        !установить начальное положение и размер
        PrtPrev.SetPosition(1,1)        ! установить только начальное положение
        PrtPrev.SetPosition(,,300,250)
                                        ! установить только начальный  размер
```

Смотри также: WindowPosSet, WindowSizeSet

SetZoomPercentile (установить определенный пользователем или стандартный вариант изменения масштаба)

SetZoomPercentile(*вариант* **)**

SetZoomPercentile Устанавливает свойства ZoomIndex и UserPercentile.

вариант Целочисленная константа, переменная, метка соответствия, или выражение, указывающее применяемый вариант изменения масштаба.

Метод **SetZoomPercentile** устанавливает свойства ZoomIndex и UserPercentile.

Реализация: SetZoomPercentile метод предполагает, что значение свойства AllowUserZoom есть Истина. Если значение параметра *вариант* равняется определенному в ZoomIndex, SetZoomPercentile устанавливает для свойства ZoomIndex значение параметра *вариант*, а для свойства UserPercentile - ноль. Если же значение параметра *вариант* не равняется определенному в ZoomIndex, SetZoomPercentile устанавливает для свойства UserPercentile значение параметра *вариант,* а для свойства ZoomIndex - ноль.

Пример:

```
ThisWindow.Init PROCEDURE()
```

```
CODE                                    !procedure code
 ThisWindow.Init(Process,report,Previewer)
 Previewer.SetZoomPercentile(120)
```

Смотри также: AllowUserZoom, UserPercentile, ZoomIndex

TakeAccepted (обработать события EVENT:Accepted)

TakeAccepted, VIRTUAL, PROC

Метод **TakeAccepted** обрабатывает события EVENT:Accepted на всех элементах управления в окне предварительного просмотра, а затем возвращает значение, указывающее, когда обработка окна закончена и пора остановиться. TakeAccepted возвращает значение Level:Benign для указания на то, что обработку этого события следует продолжить в обычном режиме; Level:Notify - что обработка этого события закончена и ACCEPT-цикл должен вернуться в начало; Level:Fatal - что событие не может быть обработано и ACCEPT-цикл следует прервать.

Метод TakeAccepted является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeAccepted в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Метод TakeEvent вызывает метод TakeAccepted. В свою очередь, метод TakeAccepted вызывает метод WindowManager.TakeAccepted, а затем обрабатывает события EVENT:Accepted для всех элементов управления на окне предварительного просмотра, таких как элементы управления масштабом изображения, кнопка печати, навигационные элементы управления, элементы управления конфигурацией указателя страниц и т.д.

Возвращаемый тип данных: BYTE

Пример:

```
MyWindowManager.TakeEvent PROCEDURE
RVal BYTE(Level:Benign)
I   USHORT,AUTO
  CODE
IF ~FIELD()
RVal = SELF.TakeWindowEvent()
```

```
                                IF RVal THEN RETURN RVal.
    END
    CASE EVENT()
    OF EVENT:Accepted;                  RVal = SELF.TakeAccepted()
    OF EVENT:Rejected;                  RVal = SELF.TakeRejected()
    OF EVENT:Selected;                  RVal = SELF.TakeSelected()
    OF EVENT:NewSelection;    RVal = SELF.TakeNewSelection()
    OF EVENT:Completed;                 RVal = SELF.TakeCompleted()
    OF EVENT:CloseWindow OROF EVENT:CloseDown
                                RVal = SELF.TakeCloseEvent()
    END
    IF RVal THEN RETURN RVal.
    IF FIELD()
                                RVal = SELF.TakeFieldEvent()
    END
    RETURN RVal
```

Смотри также: TakeEvent, WindowManager.TakeEvent

TakeEvent (обработать все события)

TakeEvent, VIRTUAL, PROC

TakeEvent метод обрабатывает все события окна предварительного просмотра и возвращает значение, указывающее, когда обработка окна закончена и пора остановиться. TakeEvent возвращает значение Level:Benign для указания на то, что обработку этого события следует продолжить в обычном режиме; Level:Notify - что обработка этого события закончена и ACCEPT-цикл должен вернуться в начало; Level:Fatal - что событие не может быть обработано и ACCEPT-цикл следует прервать.

Метод TakeEvent является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeEvent в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Метод Ask вызывает метод TakeEvent. Метод же TakeEvent вызывает метод WindowManager.TakeEvent, а затем обрабатывает на окне предварительного просмотра такие события, как EVENT:CloseWindow, EVENT:Sized и EVENT:AlertKey.

Возвращаемый тип данных: BYTE

Пример:

```
WindowManager.Ask PROCEDURE
 CODE
 IF SELF.Dead THEN RETURN .
 CLEAR(SELF.LastInsertedPosition)
 ACCEPT
  CASE SELF.TakeEvent()
  OF Level:Fatal
   BREAK
  OF Level:Notify
   CYCLE    ! Не просто так, а для быстрой остановки обработки некоторого события
  END
 END
```

Смотри также: WindowManager.Ask

TakeFieldEvent (действия по обработке связанных с полем событий)

TakeFieldEvent, VIRTUAL, PROC

Метод **TakeFieldEvent** представляет собой виртуальный метод, предназначенный для обработки всех связанных с конкретным полем или элементом управления событий в окне. Он возвращает значение, указывающее, когда обработка окна закончена и пора остановиться. TakeFieldEvent возвращает значение Level:Benign для указания на то, что обработку этого события следует продолжить в обычном режиме; Level:Notify - что обработка этого события закончена и ACCEPT-цикл должен вернуться в начало; Level:Fatal - что событие не может быть обработано и ACCEPT-цикл следует прервать.

Метод TakeFieldEvent является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeFieldEvent в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Метод TakeEvent вызывает метод TakeFieldEvent. Метод TakeFieldEvent обрабатывает события EVENT:NewSelection для элементов управления типа прокручивающегося списка в окне предварительного просмотра.

Возвращаемый тип данных: BYTE

Пример:

```
MyWindowManager.TakeEvent PROCEDURE
RVal BYTE(Level:Benign)
I   USHORT,AUTO
 CODE
        IF ~FIELD()
                                    RVal = SELF.TakeWindowEvent()
                                    IF RVal THEN RETURN RVal.
        END
        CASE EVENT()
        OF EVENT:Accepted;                  RVal = SELF.TakeAccepted()
        OF EVENT:Rejected;                  RVal = SELF.TakeRejected()
        OF EVENT:Selected;                  RVal = SELF.TakeSelected()
        OF EVENT:NewSelection;    RVal = SELF.TakeNewSelection()
        OF EVENT:Completed;                 RVal = SELF.TakeCompleted()
        OF EVENT:CloseWindow OROF EVENT:CloseDown
                                    RVal = SELF.TakeCloseEvent()
        END
        IF RVal THEN RETURN RVal.
        IF FIELD()
                                    RVal = SELF.TakeFieldEvent()
        END
        RETURN RVal
```

Смотри также: Ask

TakeWindowEvent (обработать не связанные с полем события)

TakeWindowEvent, VIRTUAL, PROC

TakeWindowEvent метод обрабатывает все несвязанные с полем события, происходящие в окне предварительного просмотра, и возвращает значение, указывающее, когда обработка окна ACCEPT-циклом закончена и пора остановиться. TakeWindowEvent возвращает значение Level:Benign для указания на то, что обработку этого события следует продолжить в обычном режиме; Level:Notify - что обработка этого события закончена и ACCEPT-цикл должен вернуться в начало; Level:Fatal - что событие не может быть обработано и ACCEPT-цикл следует прервать.

Метод TakeWindowEvent является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeWindowEvent в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Метод TakeEvent вызывает метод TakeWindowEvent. Метод TakeWindowEvent вызывает для всех событий, кроме EVENT:GainFocus, метод WindowManager.TakeWindowEvent.

Возвращаемый тип данных: BYTE

Пример:

```
MyWindowManager.TakeEvent PROCEDURE
RVal BYTE(Level:Benign)
I   USHORT,AUTO
 CODE
        IF ~FIELD()
                                RVal = SELF.TakeWindowEvent()
                                IF RVal THEN RETURN RVal.
        END
        CASE EVENT()
        OF EVENT:Accepted;              RVal = SELF.TakeAccepted()
        OF EVENT:Rejected;              RVal = SELF.TakeRejected()
        OF EVENT:Selected;              RVal = SELF.TakeSelected()
        OF EVENT:NewSelection;  RVal = SELF.TakeNewSelection()
        OF EVENT:Completed;             RVal = SELF.TakeCompleted()
        OF EVENT:CloseWindow OROF EVENT:CloseDown
                                RVal = SELF.TakeCloseEvent()
        END
        IF RVal THEN RETURN RVal.
        IF FIELD()
                                RVal = SELF.TakeFieldEvent()
        END
        RETURN RVal
```

Смотри также: TakeEvent

Класс Менеджера Отчета

Краткий обзор

Концепции ReportManager'а

ReportManager представляет собой WindowManager, использующий объект Process-класса как *подготовительный* процесс, а также опционально использует объект PrintPreview-класса для подготовки полностью оснащенного окна предварительного просмотра печати. Следовательно, ReportManager поддерживает пакетное выполнение процедуры построения отчета в окне отображения хода процесса, предварительный просмотр печати, необходимую фильтрацию записей, а также оптимизированное распределение машинных ресурсов.

Связь с другими Классами разработки приложений

ReportManager получен из WindowManager (см. *Класс Менеджера Окна* для получения дополнительной информации) поскольку он поддерживает окно отображения хода процесса (окно прогресса), предназначенное для обеспечения визуальной обратной связи с конечным пользователем.

ReportManager использует Process-класс для управления пакетной обработкой лежащей в основе отчета VIEW-структуры. Также он опционально использует PrintPreview-класс для обеспечения отчета полностью подготовленным средством предварительного просмотра печати.

Если в Вашей программе порождается ReportManager, то должен быть также порожден и Process-класс, а также, возможно, потребуется порождение и PrintPreview-класса. Большая часть этих действий выполняется автоматически, стоит Вам только включить заголовок ReportManager'а (файл ABREPORT.INC) в секцию данных вашей программы. См. *Пояснительный Пример*.

Реализация ABC шаблона

Процедурный Шаблон Отчета и шаблон Утилиты Мастера Отчета автоматически генерируют весь код и включают все классы, необходимые для поддержки сгенерированных с помощью шаблонов отчетов, содержащихся в Вашем приложении.

Эти шаблоны Отчета генерируют текст для порождения объекта ReportManager'a, именуемого ThisWindow, для каждой сгенерированной процедуры отчета. Для использования объектом ThisWindow шаблоны Отчета также порождают такие объекты, как Process, и, опционально, Previewer.

Объект ThisWindow поддерживает все функциональные возможности, указанные в принадлежащем шаблону Отчета окне диалога **Свойства Отчета**. См. *Процедурные Шаблоны— Отчет* для получения дополнительной информации.

Файлы-источники ReportManager'a

Файлы с исходным кодом ReportManager'a по умолчанию установлены в директорию \CLARION4\LIBSRC. Собственно файлы ReportManager'a и их соответствующие компоненты представлены ниже:

ABREPORT.INC	Объявления ReportManager'a
ABREPORT.CLW	Определения методов ReportManager'a

Пояснительный Пример

Приведенный ниже пример показывает типовую последовательность операторов, предназначенных для объявления, порождения, инициализации, использования и завершения объекта ReportManager'a и связанных с ним объектов.

В этом примере объект ReportManager'a используется для предварительного просмотра очень простого отчета до того, как его напечатать. Программа определяет окна предварительного просмотра печати с максимизированными размерами.

```
  PROGRAM
INCLUDE('ABREPORT.INC')                    !объявить ReportManager
                                           ! и PrintPreviewClass
  MAP
  END
GlobalErrors  ErrorClass
VCRRequest LONG(0),THREAD
Customer
FILE,DRIVER('TOPSPEED'),PRE(CUS),THREAD
BYNUMBER
KEY(CUS:CUSTNO),NOCASE,OPT,PRIMARY
Record     RECORD,PRE()
CUSTNO     LONG
```

```
Name        STRING(30)
State       STRING(2)
            END
            END
Access:Customer CLASS(FileManager)     !объявить объект Access:Customer
Init        PROCEDURE
END
Relate:Customer CLASS(RelationManager)! объявить объект Relate:Customer
Init        PROCEDURE
END
CusView     VIEW(Customer)             ! объявить VIEW-структуру CusView
END
PctDone     BYTE                       ! Переменная отображения хода процесса
                                       !report
REPORT,AT(1000,1542,6000,7458),PRE(RPT),FONT('Arial',10,,),THOUS
HEADER,AT(1000,1000,6000,542),FONT(,,,FONT:bold)
STRING('Customers'),AT(2000,20),FONT(,14,,)
STRING('Id'),AT(52,313),TRN
STRING('Name'),AT(2052,313),TRN
STRING('State'),AT(4052,313),TRN
END
detail      DETAIL,AT(,,6000,281),USE(?detail)
STRING(@n-14),AT(52,52),USE(CUS:CUSTNO)
STRING(@s30),AT(2052,52),USE(CUS:NAME)
STRING(@s2),AT(4052,52),USE(CUS:State)
            END
            FOOTER,AT(1000,9000,6000,219)
STRING(@pPage <<<#p),AT(5250,31),PAGENO,USE(?PageCount)
            END
            END
ProgressWindow
WINDOW('Progress...'),AT(,,142,59),CENTER,TIMER(1),GRAY,DOUBLE
            PROGRESS,USE(PctDone),AT(15,15,111,12),RANGE(0,100)
STRING(''),AT(0,3,141,10),USE(?UserString),CENTER
STRING(''),AT(0,30,141,10),USE(?TxtDone),CENTER
BUTTON('Cancel'),AT(45,42),USE(?Cancel)
END
ThisProcedure  CLASS(ReportManager)    ! объявить объект ThisProcedure
Init
PROCEDURE(),BYTE,PROC,VIRTUAL
Kill
PROCEDURE(),BYTE,PROC,VIRTUAL
END
```

```
CusReport CLASS(ProcessClass)          ! объявить объект CusReport
TakeRecord  PROCEDURE(),BYTE,PROC,VIRTUAL
END
Previewer   PrintPreviewClass          ! объявить объект Previewer
                                       ! для использования совместно с
ThisProcedure
 CODE
 ThisProcedure.Run()                   !выполнить процедуру
ThisProcedure.Init  PROCEDURE()        !инициализировать ThisProcedure
ReturnValue  BYTE,AUTO
 CODE
 GlobalErrors.Init
 Relate:Customer.Init
 ReturnValue = PARENT.Init()
 SELF.FirstField = ?PctDone
 SELF.VCRRequest &= VCRRequest
 SELF.Errors &= GlobalErrors           !установить обработчик ошибок для
ThisProcedure
 Relate:Customer.Open                  !открыть файл Customer и файлы, связанные с
                                       !ним
 OPEN(ProgressWindow)
 SELF.Opened=True                      !произвести специфическую инициализацию
                                       !отчета
 CusReport.Init(CusView,Relate:Customer,?TxtDone,PctDone,RECORDS(Customer))
 CusReport.AddSortOrder(CUS:BYNUMBER)
                                       !установить порядок сортировки отчета
 SELF.AddItem(?Cancel,RequestCancelled)
                                       !установить действие при отмене
 SELF.Init(CusReport,report,Previewer)! зарегистрировать Previewer и CusReport с
                                       !ThisProcedure
 SELF.Zoom = PageWidth
 Previewer.AllowUserZoom=True
                                       !разрешить изменение масштаба
                                       !изображения
 Previewer.Maximize=True
                                       !начальная максимизация окна
                                       !предварительного просмотра
 SELF.SetAlerts()                      ! «горячие клавиши» для ThisProcedure
 RETURN ReturnValue
ThisProcedure.Kill  PROCEDURE()        !закрыть ThisProcedure
ReturnValue  BYTE,AUTO
 CODE
```

```
  ReturnValue = PARENT.Kill()          !вызвать закрытие основного класса
  Relate:Customer.Close                ! закрыть файл Customer и файлы, связанные с
                                       !ним
  Relate:Customer.Kill                 !закрыть объект Relate:Customer
  GlobalErrors.Kill                    ! закрыть объект GlobalErrors
  RETURN ReturnValue
CusReport.TakeRecord  PROCEDURE()
                                       !произвести некую обработку конкретной
                                       !записи
ReturnValue   BYTE,AUTO
SkipDetails  BYTE
  CODE
  ReturnValue = PARENT.TakeRecord()    !стандартная обработка каждой записи
  PRINT(RPT:detail)                    !напечатать секцию detail для каждой записи
  RETURN ReturnValue
Access:Customer.Init  PROCEDURE
  CODE
  PARENT.Init(Customer,GlobalErrors)
  SELF.FileNameValue = 'Customer'
  SELF.Buffer &= CUS:Record
  SELF.Create = 0
  SELF.LazyOpen = False
  SELF.AddKey(CUS:BYNUMBER,'CUS:BYNUMBER',0)
Relate:Customer.Init  PROCEDURE
  CODE
  Access:Customer.Init
  PARENT.Init(Access:Customer,1)
```

Свойства ReportManager'а

ReportManager содержит свойства, которые делают возможным предварительный просмотр отчета в диалоговом режиме и конфигурируют его. Свойства ReportManager'а описаны ниже.

DeferOpenReport (отложить открытие отчета)

```
DeferOpenReport   BYTE, PROTECTED
```

DeferOpenReport свойство определяет, открывает ли ReportManager отчет в момент работы метода Open, или откладывает его открытие до первого прохождения цикла таймера. Значение единица (1 или Истина) откладывает открытие; значение ноль (0 или Ложь) открывает отчет немедленно.

DeferOpenReport свойство предоставляет возможность до того, как начнется печать отчета, уточнить у конечного пользователя значения различных фильтров и порядок сортировки.

Это свойство защищено, поэтому оно может быть использовано либо методом ReportManager, либо методом в классе, полученном из ReportManager.

Реализация: И метод Open, и метод TakeWindowEvent осуществляют поведение, определенное свойством DeferOpenReport.

Смотри также: Open, TakeWindowEvent

Preview (Объект PrintPreview-класса)

Preview &PrintPreviewClass, PROTECTED

Свойство **Preview** представляет собой ссылку на тот объект PrintPreview-класса, который используется ReportManager для обеспечения предварительного просмотра отчета в диалоговом режиме.

Это свойство защищено, поэтому оно может быть использовано либо методом ReportManager, либо методом в классе, полученном из ReportManager.

Реализация: Метод Init устанавливает свойство Preview.

Смотри также: Init

PreviewQueue (полные пути к метафайлам отчета)

PreviewQueue &PreviewQueue, PROTECTED

Свойство **PreviewQueue** представляет собой ссылку на структуру, содержащую полные пути и имена относящихся к отчету метафайлов Windows (*.WMF) — по одному метафайлу для каждой страницы отчета. Объект ReportManager использует это свойство и для обеспечения предварительного просмотра отчета в диалоговом режиме, и для печати отчета после просмотра. См. *PREVIEW* в *Описании Языка* для получения дополнительной информации о метафайлах отчета.

Это свойство защищено, поэтому оно может быть использовано либо методом ReportManager, либо методом в классе, полученном из ReportManager.

Реализация: ReportManager только тогда будет использовать свойство PreviewQueue, когда определено свойство Preview.

Структура PreviewQueue объявлена в ABREPORT.INC следующим образом:

```
PreviewQueue  QUEUE,TYPE
Filename    STRING(128)
            END
```

Смотри также: Preview

Process (Объект Process-класса)

Process	&ProcessClass, PROTECTED

Свойство **Process** представляет собой ссылку на объект Process-класса, который используется ReportManager для управления пакетной обработкой относящихся к отчету данных. Свойство Process применяет к данным все необходимые порядки сортировки, ограничения диапазонов изменения, фильтры, а также обеспечивает визуальную обратную связь с конечным пользователем путем отображения окна прогресса со сведениями о ходе обработки.

Это свойство защищено, поэтому оно может быть использовано либо методом ReportManager, либо методом в классе, полученном из ReportManager.

Реализация: Метод Init определяет свойство Process.
Смотри также: Init

Report (руководимый Отчет)

Report	&WINDOW

Свойство **Report** представляет собой ссылку на управляемую структуру Отчет. Это свойство используется ReportManager для открытия, печати и закрытия Отчета.

Реализация: Метод Init определяет свойство Report.
Смотри также: Init

SkipPreview (напечатать отчет, пропустив предварительный просмотр)

SkipPreview	BYTE

SkipPreview свойство определяет, происходит ли при вызове ReportManager открытие окна предварительного просмотра, или вместо этого печатается отчет. Значение единица (1 или Истина) приводит к тому, что отчет печатается сразу; значение же ноль (0 или Ложь) – что вначале происходит его предварительный просмотр. SkipPreview действует только тогда, когда установлено свойство Preview.

Свойство SkipPreview позволяет подавить вызов диалогового предварительного просмотра печати в любое время, прежде чем сработает метод AskPreview.

Реализация: Метод AskPreview реализует поведение, указанное SkipPreview свойством.

Смотри также: AskPreview, Preview

TimeSlice (использование ресурсов отчетом)

TimeSlice USHORT

Свойство **TimeSlice** содержит количество времени в сотых долях секунды, в течение которого работает ReportManager при каждом прохождении цикла. Цикл начинается с EVENT:Timer (см. *TIMER* в *Описании Языка*), и заканчивается на TimeSlice позже. Например, если TimeSlice равен 100, то ReportManager обрабатывает столько записей, насколько это возможно в пределах 100/100 (одной) секунды до тех пор, пока управление не будет передано назад операционной системе. Для обеспечения эффективного распределения машинных ресурсов рекомендуется устанавливать таймер несколько меньше, чем TimeSlice или равным ему.

Реализация: Метод Init устанавливает значение TimeSlice соответствующим одной секунде (100). Метод TakeWindowEvent непрерывно регулирует количество обрабатываемых за цикл записей с тем, чтобы заполнить весь указанный промежуток TimeSlice — то есть обрабатывать столько записей, насколько это возможно в пределах TimeSlice диапазона. Это обеспечивает как эффективную обработку отчета, так и разумное распределение машинных ресурсов, если величина таймера меньше или равна значению TimeSlice. Таким образом пользователю предоставляется возможность управления в многозадачной среде, особенно при обработке большого набора данных.

Смотри также: Init, TakeWindowEvent

Zoom (Начальное увеличение при предварительном просмотре отчета)

Zoom SHORT

Свойство **Zoom** определяет начальный вариант изменения масштаба изображения для окна предварительного просмотра отчета. Значение ноль (0) использует установки изменения масштаба объекта PrintPreview-класса по умолчанию. Любое другое значение определяет конкретный начальный вариант изменения масштаба.

Свойство Zoom позволяет переопределить присущие по умолчанию объекту PrintPreview-класса варианты изменения масштаба изображения. Объект PrintPreview-класса определяет фактически примененный вариант.

Свойство Zoom действует только тогда, когда установлено свойство Preview.

Реализация: Метод AskPreview осуществляет поведение, определенное свойством Zoom путем передачи значения Zoom методу PrintPreviewClass.Display.

Если объект PrintPreview-класса допускает использование измененных вариантов изменения масштаба, то начальное увеличение равняется значению Zoom (81 соответствует 81 %, 104 соответствует 104 % и т.д.). Если же объект PrintPreview-класса поддерживает только ограниченный набор дискретных вариантов увеличения, то начальное увеличение будет соответствовать самому близкому по величине значению Zoom (81 соответствует 75 %, 104 соответствует 100 % и т.д.).

Смотри также: AskPreview, Preview, PrintPreviewClass.ZoomIndex

Методы ReportManager

ReportManager содержит методы описанные ниже.

Функциональная Организация — Предполагаемое Использование

Для лучшего понимания ReportManager будет полезно организовать его различные методы в две больших категории согласно их предполагаемому использованию — первичный интерфейс и виртуальные методы. Такая организация, по мнению разработчиков, наиболее полно отражает типовое использование методов ReportManager.

Первичные Интерфейсные Методы

Первичные интерфейсные методы, вызов которых, скорее всего, будет иметь место в Вашей программе, могут быть далее разделены на три категории:

Одноразовое применение:

Init[V]	инициализировать объект ReportManager
Kill[V]	завершить объект ReportManager

[V] Эти методы также являются виртуальными.

Виртуальные Методы

В основном Вам не придется прямо вызывать эти методы, поскольку они вызываются первичными интерфейсными методами. Однако у Вас наверняка часто будет возникать потребность в переопределении этих методов, а поскольку они являются виртуальными, то сделать это достаточно просто. Эти методы уже по умолчанию обеспечивают «разумное» поведение, в случае, если Вы не переопределите их.

AskPreview	предварительный просмотр или печать отчета
Next	получить следующую запись отчета
Open	подготовить окно прогресса
OpenReport	подготовить отчет к выполнению
TakeCloseEvent	обработка событий EVENT:CloseWindow
TakeWindowEvent	обработка несвязанных с полем событий
Kill	закрыть объект ReportManager

AskPreview (предварительный просмотр или печать отчета)

AskPreview, VIRTUAL

Метод **AskPreview** просматривает или печатает отчет, только если свойство Preview ссылается на действующий объект PrintPreview-класса.

Если значение свойства SkipPreview есть Истина, AskPreview не производит предварительного просмотра отчета, а сразу печатает его.

Метод AskPreview является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод AskPreview в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод Ask вызывает метод AskPreview для печати или предварительного просмотра отчета. Для предварительного просмотра отчета метод AskPreview вызывает метод PrintPreviewClass.Display.

Как правило, Init метод устанавливает ссылки свойства Preview.

Пример:

```
ReportManager.Ask PROCEDURE
 CODE
  PARENT.Ask
  SELF.AskPreview()
  IF ~SELF.Report&=NULL
   CLOSE(SELF.Report)
  END
```

Смотри также: Ask, PrintPreviewClass.Display, Init, Preview, SkipPreview

Init (Инициализировать объект ReportManager)

Init(*Объект обработки* [,*Отчет*] [,*Объект предварительного просмотра*] **)**

Init	Инициализирует объект ReportManager.
Объект обработки	Метка объекта Process-класса, используемого ReportManager для пакетной обработки VIEW-структуры *отчета* и обеспечения соответствующей визуальной обратной связи с конечным пользователем посредством окна прогресса *отчета.*
*Отчет*Метка	управляемой структуры Отчет. Если параметр опущен, ReportManager превращается в пакетный обработчик VIEW-структуры с автоматическим управлением ресурсами.
Объект предварительного просмотра	Метка объекта PrintPreview-класса, используемого ReportManager для предварительного просмотра или печати *отчета.* Если параметр опущен, ReportManager печатает сообщение без генерации файлов для предварительного просмотра.

Метод **Init** производит специфическую для данного отчета инициализацию объекта ReportManager. Этот Init метод является дополнением к тому Init методу, унаследованному от класса WindowManager, который делает общую инициализацию оконной процедуры.

Реализация: Как правило, Init метод вызывает метод Init (*объект обработки, отчет, объект предварительного просмотра*) для произведения соответствующей конкретному отчету инициализации. Init метод устанавливает такие свойства, как Preview, Process, Report и TimeSlice.

Пример:

```
PrintPhones  PROCEDURE
report       REPORT,AT(1000,1540,6000,7460),PRE(RPT)
detail       DETAIL,AT(,,6000,280)
STRING(@s20),AT(50,50,5900,170),USE(PHO:Number)
END
END
Previewer  PrintPreviewClass          !объявить объект Previewer
Process    ProcessClass               ! объявить объект Process
ThisWindow  CLASS(ReportManager)      ! объявить полученный объект ThisWindow
Init
PROCEDURE(),BYTE,PROC,VIRTUAL
Kill
PROCEDURE(),BYTE,PROC,VIRTUAL
```

```
END
!секция данных процедуры
            CODE
 ThisWindow.Run                          !выполнить процедуру (init,ask,kill)
ThisWindow.Init  PROCEDURE()
 CODE                                    !текст процедуры
 ThisWindow.Init(Process,report,Previewer)
                                         !вызов соответствующего отчету I
                                         !текст процедуры
```

Смотри также: WindowManager.Init

Kill (закрыть объект ReportManager)

Kill, VIRTUAL, PROC

Метод **Kill** освобождает любую память, размещенную в течение жизни объекта, а также выполняет любой другой требуемый код завершения. Он возвращает значение, показывающее статус завершения. Его возможные возвращаемые значения:

Level:Benign	нормальное завершение
Level:Notify	нет произведенных действий

Метод Kill является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод Kill в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Метод Run вызывает метод Kill. Если значение свойства Kill есть Истина, Kill возвращает Level:Notify и не предпринимает каких-либо других действий. В противном случае метод Kill, помимо всего прочего, вызывает еще и метод WindowManager.Kill.

Метки соответствия различных вариантов возвращаемых значений объявлены в ABERROR.INC.

Возвращаемый тип данных: BYTE

Пример:

```
ThisWindow.Kill  PROCEDURE()
            CODE
```

```
            IF PARENT.Kill() THEN RETURN Level:Notify.
            IF FilesOpened
            Relate:Defaults.Close
            END
            IF SELF.Opened
            INIMgr.Update('Main',AppFrame)
            END
            GlobalResponse =
CHOOSE(LocalResponse=0,RequestCancelled,LocalResponse)
```

Смотри также: WindowManager.Dead, WindowManager.Run

Next (получить следующую запись отчета)

Next, VIRTUAL, PROC

Метод **Next** получает следующую запись отчета и возвращает значение, указывающее, закончен ли отчет, отменен или находится в стадии выполнения. Его возможные возвращаемые значения:

Level:Benign	выполняется нормально
Level:Notify	закончен нормально
Level:Fatal	отменен или закончен ненормально

Метод Next является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод Next в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Метод Next вызывает метод ProcessClass.Next для получения следующей записи отчета. Когда отчет закончен или отменен, метод Next устанавливает значение свойства Response и посылает событие EVENT:CloseWindow для закрытия процедуры окна прогресса.

Возвращаемый тип данных: BYTE

Пример:

```
ReportManager.Open  PROCEDURE
 CODE
  PARENT.Open
  SELF.Process.Reset
```

```
  IF ~SELF.Next()
   IF ~SELF.Report&=NULL
    OPEN(SELF.Report)
    IF ~SELF.Preview &= NULL
     SELF.Report{PROP:Preview} = SELF.PreviewQueue.Filename
    END
   END
  END
```

Смотри также: ProcessClass.Next, WindowManager.Response

Open (действие по обработке события EVENT:OpenWindow)

Open, VIRTUAL

Метод **Open** готовит окно прогресса к отображению. Он предназначен для обработки при открытии окна ряда событий, таких как EVENT:OpenWindow.

Метод Open является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод Open в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: Метод TakeWindowEvent вызывает метод Open. Он, в свою очередь, вызывает метод WindowManager.Open, а затем условно (в зависимости от свойства DeferOpenReport) вызывает метод OpenReport для переустановки объекта Process-класса и получения первой записи отчета.

Пример:

```
WindowManager.TakeWindowEvent      PROCEDURE
RVal BYTE(Level:Benign)
 CODE
 CASE EVENT()
 OF EVENT:OpenWindow
  IF ~BAND(SELF.Inited,1)
   SELF.Open                       !обработать событие EVENT:OpenWindow
  END
  IF SELF.FirstField
   SELECT(SELF.FirstField)
  END
 OF EVENT:LoseFocus
  IF SELF.ResetOnGainFocus
   SELF.ForcedReset = 1
  END
 OF EVENT:GainFocus
```

```
  IF BAND(SELF.Inited,1)
    SELF.Reset
  ELSE
    SELF.Open                          ! обработать событие EVENT:GainFocus
  END
 OF EVENT:Sized
  IF BAND(SELF.Inited,2)
    SELF.Reset
  ELSE
    SELF.Inited = BOR(SELF.Inited,2)
  END
 OF EVENT:Completed
  RVal = SELF.TakeCompleted()
 OF EVENT:CloseWindow OROF EVENT:CloseDown
  RVal = SELF.TakeCloseEvent()
 END
 RETURN RVal
```

Смотри также: DeferOpenReport, OpenReport, WindowManager.Open, WindowManager.TakeWindowEvent

OpenReport (подготовить отчет к выполнению)

OpenReport, PROTECTED, VIRTUAL

Метод **OpenReport** подготавливает отчет к выполнению. Является хорошим местом для добавления каких-нибудь фильтров или ключей, значение которых определяется во время выполнения.

Этот метод защищен, поэтому он может быть вызван только либо методом ReportManager, либо методом в классе, полученном из ReportManager.

Метод OpenReport является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод OpenReport в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Реализация: В зависимости от значения свойства DeferOpenReport метод OpenReport вызывается либо методом TakeWindowEvent, либо методом Open. Метод OpenReport вызывает метод Process.Reset для переустановки объекта Process-класса, метод Next для получения первой записи отчета, а также открывает структуру Отчет.

Метод OpenReport переустанавливает свойство DeferOpenReport равным нулю, так что, если имела место отсрочка, то OpenReport происходит только с первым событием таймера.

Пример:

```
ReportManager.Open  PROCEDURE
 CODE
            PARENT.Open
            IF ~SELF.DeferOpenReport
            SELF.OpenReport                 !если не отложено, то вызвать OpenReport
            END
MyReportManager.TakeWindowEvent PROCEDURE
                                            !procedure data
 CODE
IF EVENT() = EVENT:Timer
IF SELF.DeferOpenReport
SELF.OpenReport                             ! если отложено, то вызвать OpenReport по
                                            !таймеру
ELSE
MyReportManager.OpenReport  PROCEDURE
 CODE
SELF.Process.SetFilter(UserFilter)          ! установить динамический фильтр
            SELF.DeferOpenReport = 0
            SELF.Process.Reset
            IF ~SELF.Next()
            IF ~SELF.Report&=NULL
            OPEN(SELF.Report)
            IF ~SELF.Preview &= NULL
            SELF.Report{PROP:Preview} = SELF.PreviewQueue.Filename
            .                               .   .
```

Смотри также: DeferOpenReport, Next, Open, TakeWindowEvent, Process.Reset

TakeCloseEvent (действия по обработке события EVENT:CloseWindow)

TakeCloseEvent, VIRTUAL, PROC

TakeCloseEvent метод обрабатывает EVENT:CloseWindow для ReportManager и возвращает значение, показывающее, когда обработка ACCEPT-цикла закончена и следует остановиться.

TakeCloseEvent возвращает значение Level:Benign для указания на то, что обработку этого события следует продолжить в обычном режиме; Level:Notify - что обработка этого события закончена и ACCEPT-цикл должен вернуться в начало;

Level:Fatal - что событие не может быть обработано и ACCEPT-цикл следует прервать.

Метод TakeCloseEvent является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeCloseEvent в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Метод TakeEvent вызывает метод TakeCloseEvent. Метод TakeCloseEvent вызывает метод AskPreview для предварительного просмотра или печати отчета, а затем закрывает отчет.

Возвращаемый тип данных: BYTE

Пример:

```
MyWindowManager.TakeEvent PROCEDURE
RVal BYTE(Level:Benign)
I   USHORT,AUTO
 CODE
            IF ~FIELD()
            RVal = SELF.TakeWindowEvent()
            IF RVal THEN RETURN RVal.
            END
            CASE EVENT()
            OF EVENT:Accepted;                  RVal = SELF.TakeAccepted()
            OF EVENT:Rejected;                  RVal = SELF.TakeRejected()
            OF EVENT:Selected;                  RVal = SELF.TakeSelected()
            OF EVENT:NewSelection;    RVal = SELF.TakeNewSelection()
            OF EVENT:Completed;                 RVal = SELF.TakeCompleted()
            OF EVENT:CloseWindow OROF EVENT:CloseDown
            RVal = SELF.TakeCloseEvent()
            END
            IF RVal THEN RETURN RVal.
            IF FIELD()
            RVal = SELF.TakeFieldEvent()
            END
            RETURN RVal
```

Смотри также: AskPreview, WindowManager.TakeEvent

TakeWindowEvent (действия по обработке несвязанных с полем событий)

TakeWindowEvent, VIRTUAL, PROC

TakeWindowEvent метод обрабатывает все несвязанные с полем события для окна прогресса и возвращает значение, показывающее, когда обработка ACCEPT-цикла закончена и следует остановиться. TakeWindowEvent возвращает значение Level:Benign для указания на то, что обработку этого события следует продолжить в обычном режиме; Level:Notify - что обработка этого события закончена и ACCEPT-цикл должен вернуться в начало; Level:Fatal - что событие не может быть обработано и ACCEPT-цикл следует прервать.

Метод TakeWindowEvent является виртуальным методом, так что другие основные методы класса могут непосредственно вызывать метод TakeWindowEvent в этом классе. Это значительно облегчает разработку вашей собственной программной версии этого метода.

Этот метод имеет атрибут PROC, так что можно вызывать его как процедуру и игнорировать возвращаемое значение.

Реализация: Метод TakeEvent вызывает метод TakeWindowEvent.

Метод TakeWindowEvent обрабатывает события EVENT:Timer для отчета. Метод TakeWindowEvent или вызывает OpenReport (если значение свойства DeferOpenReport есть Истина), или начинает обрабатывать “цикл” записей отчета. Каждое событие таймера начинает “цикл” обработки записи отчета, который заканчивается по истечении промежутка TimeSlice.

Метод TakeWindowEvent вызывает методы TakeRecord и Next для каждой записи в пределах цикла обработки.

TakeWindowEvent регулирует число записей, обрабатываемых за один проход цикла, с тем, чтобы заполнить TimeSlice и оптимизировать разделение машинных ресурсов.

Наконец, TakeWindowEvent вызывает метод WindowManager.TakeWindowEvent для обработки любых других несвязанных с полем событий.

Возвращаемый тип данных: BYTE

Пример:

```
MyWindowManager.TakeEvent PROCEDURE
RVal BYTE(Level:Benign)
I   USHORT,AUTO
 CODE
            IF ~FIELD()
            RVal = SELF.TakeWindowEvent()
            IF RVal THEN RETURN RVal.
            END
            CASE EVENT()
            OF EVENT:Accepted;                  RVal = SELF.TakeAccepted()
            OF EVENT:Rejected;                  RVal = SELF.TakeRejected()
            OF EVENT:Selected;                  RVal = SELF.TakeSelected()
            OF EVENT:NewSelection;    RVal = SELF.TakeNewSelection()
            OF EVENT:Completed;           RVal = SELF.TakeCompleted()
            OF EVENT:CloseWindow OROF EVENT:CloseDown
            RVal = SELF.TakeCloseEvent()
            END
            IF RVal THEN RETURN RVal.
            IF FIELD()
            RVal = SELF.TakeFieldEvent()
            END
            RETURN RVal
```

Смотри также: DeferOpenReport, Next, TimeSlice, TakeRecord, WindowManager.TakeEvent, WindowManager.TakeWindowEvent