CLARION 5

User's Guide

This publication supports Clarion 5. It is possible that it may contain technical or typographical errors. TopSpeed Corporation provides this publication "as is," without warranty of any kind, either expressed or implied.

TopSpeed Corporation
150 East Sample Road
Pompano Beach, Florida 33064
(954) 785-4555

Trademark Acknowledgements:

Printed in the United States of America (0998)

CONTENTS

FOREWORD

Welcome to the *Clarion 5 User's Guide*! This book is a complete reference to the static (non-template) parts of the Clarion development environment.

Once you've become familiar with the Clarion development environment, through *Getting Started* and *Learning Clarion*, you can refer to the *User's Guide* for complete in-depth information, or you can read it for pure entertainment pleasure.

The *User's Guide* contains a chapter or two on each major component of the development environment from the Data Dictionary Editor to the 32-bit Debugger, including examples and tutorials on effective development techniques. In addition, the *User's Guide* includes several appendices addressing various concepts and tools that are important to Windows programming and application development in general.

Although to some extent the *User's Guide* contents tracks the development environment's user interface, it also suggests a rough, but not mandatory, sequence of tasks with which to approach application development. When you want more information on how to accomplish a specific programming task we recommend using the index or the master index to find task-specific information.

Please refer to the *Application Handbook* for complete information on the various templates, the Application Builder Class Library. Please refer to the *Programmer's Guide* for complete information on the database drivers that come with this product. Please refer to the *Language Reference* for complete information on Clarion language syntax and related examples. And don't forget to make liberal use of the extensive on-line help.

Documentation Conventions

Typeface Conventions

Italics	Indicates what to type at the keyboard and variable information, such as *Enter This* or *filename*.TXT.
SMALL CAPS	Indicates keystrokes to enter at the keyboard such as ENTER or ESCAPE, and mouse operations such as RIGHT-CLICK.
Boldface	Indicates commands or options froma menu or text in a dialog window.
UPPERCASE	Clarion language keywords such as MAX or USE.
`LETTER GOTHIC`	Used for diagrams, source code listings, to annotate examples, and for examples of the usage of source statements.

Keyboard Conventions

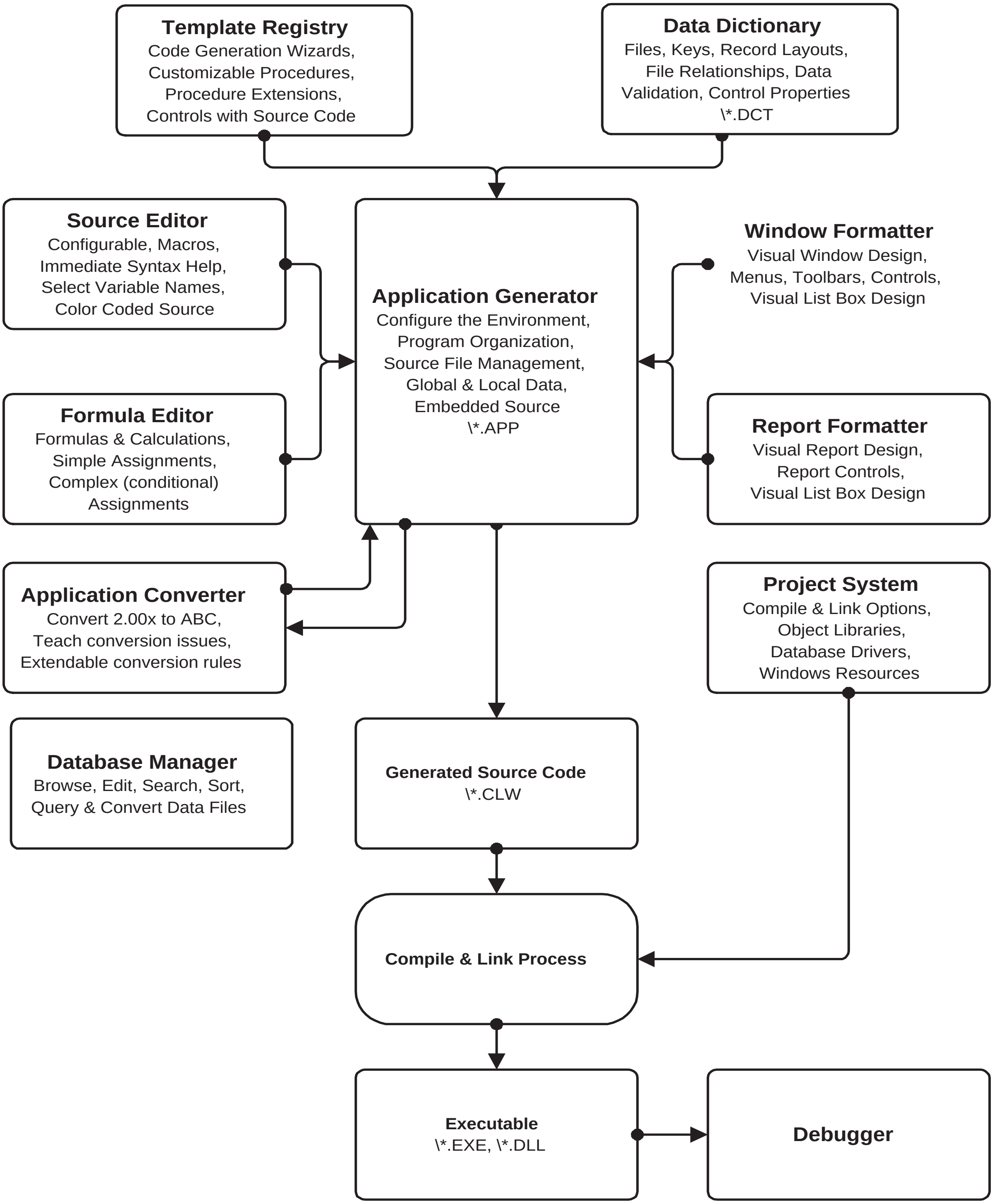
F1	Indicates a single keystroke. In this case, press and release the F1 key.
ALT+X	Indicates a combination of keystrokes. In this case, hold down the ALT key and press the X key, then release both keys.

Other Conventions

Tip: Special Tips, Notes, and Warnings—information that is not immediately evident from the topic explanation.

⚠ Indicates vital information. If you read nothing else, read this.

1 - CLARION'S DEVELOPMENT ENVIRONMENT



Overview

This chapter introduces you to Clarion's development environment. It also introduces you to some terminology and concepts used throughout this book.

Clarion's development environment is a program that helps you write other programs. The development environment manages all the files (source code, executables, resources, etc.) and all the processes (editing, compiling, linking, etc.) needed to successfully develop Windows programs. The *Getting Started* and *Learning Clarion* books provide a wonderful general introduction to Clarion's development environment.

The development environment is highly configurable. You can adjust the environment to the way you work so you can be most productive. This chapter describes the various adjustments you can make to the environment. You may want to browse through this chapter just to get an idea of how flexible and how powerful the development environment is.

Applications and Projects

This book uses the terms "application" and "project" somewhat interchangeably, although there is a fine distinction between these two terms.

Both applications and projects should be distinguished from the Project System, which is the part of the development environment that manages the compile and link process. The Project System compiles and links both applications and projects. See *Project System* for more information.

Applications

By convention, a Clarion application refers to a project whose source code is generated by templates and whose instructions for compiling and linking are stored within the application file (.APP), rather than in a separate Project System file (.PRJ).

Projects

By convention, a Clarion project includes some set of source code, plus the instructions (.PRJ file) for compiling and linking to produce an executable program. Project typically refers to a manually coded program rather than a template generated program.

Basic Environment Commands

Clarion's development environment provides menu commands, toolbar buttons, and keyboard shortcuts to access all the development environment tools. The most commonly used tools have toolbar buttons and keyboard shortcuts. Toolbar buttons are shown beside their corresponding menu commands.

File Commands

Clarion's development environment menu provides several commands to manipulate various files used in the development process. These basic file commands are described here.

New

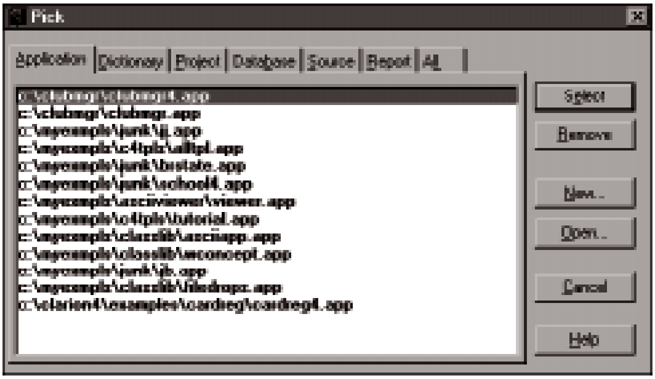
The **New** command creates a new file of the type you select in the folder you select. We recommend you begin development with the Dictionary (.DCT) file.

Open

The **Open** command opens an existing file. Select the file from the Windows file open dialog.

Pick

The **Pick** command opens an existing file. Select the file from Clarion's **Pick** dialog. This dialog shows a list of the most recently accessed files, categorized by file type. Highlight the file, then press the **Select** button. Press the **Remove** button to remove filenames from the list.



```
c:\clubmgr\clubmgr4.app
c:\clubmgr\clubmgr.app
c:\myexmpls\junk\jj.app
c:\myexmpls\c4tpls\alltpl.app
c:\myexmpls\junk\bstate.app
c:\myexmpls\junk\school4.app
c:\myexmpls\asciiviewer\viewer.app
c:\myexmpls\c4tpls\tutorial.app
c:\myexmpls\classlib\asciiapp.app
c:\myexmpls\classlib\wconcept.app
c:\myexmpls\junk\jb.app
c:\myexmpls\classlib\filedrops.app
c:\clarion4\examples\cardreg\cardreg4.app
```

Close

The **Close** command closes the active file and prompts you to save or abandon any pending changes.

Save

The **Save** command saves the active file to disk.

Save as

The **Save as** command saves a copy of the active file to a new destination. The new file becomes the active file, and the original file is abandoned.

Save all

The **Save all** command saves all open files to disk.

Print

The **Print** command prints the active file.

Print

The **Print Setup** command opens the Windows **Print Setup** dialog.

Browse Database

The **Browse Database** command opens the Database Manager to browse and edit the data file you specify. See *Database Manager* for more information.

Convert Application

The **Convert Application** command starts the Clarion Application Conversion Wizard to convert applications developed in one Clarion environment to another Clarion environment (newer Clarion version, different templates, etc.). See *Application Converter* for more information.

Note: If your templates are unchanged, you can open application files with newer versions of Clarion and the file conversion is automatic. The Application Conversion Wizard carries out more complex conversions.

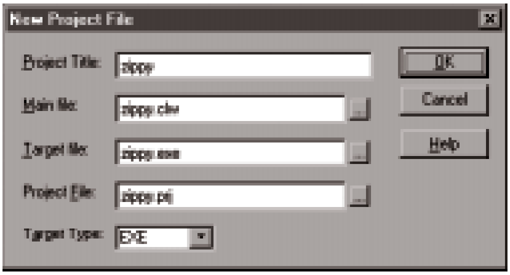
Application and Project Commands—The Project Menu

Clarion's development environment provides several commands which act on the application or project (see *Applications and Projects*). This section provides a list of the commands and what they do. To access these commands, choose **Project** from the menu, or press the corresponding toolbar button.

Set Makes a project active, so that subsequent project commands such as **Make** and **Run** operate on the project. The **Select Project** dialog appears; select a .PRJ or .APP file from the list box.

New

Creates a new project file. Fill in the **Project Title** and **Main file** fields in the **New Project File** dialog.



New Project File

Project Title: zippy

Main file: zippy.clw

Target file: zippy.exe

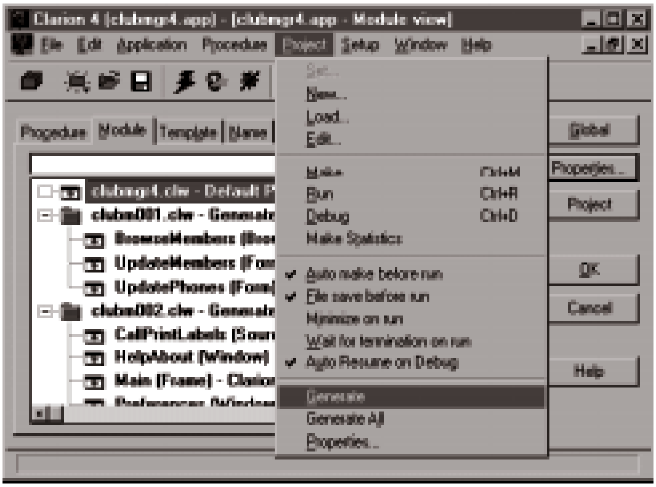
Project File: zippy.prj

Target Type: EXE

Load

Makes a project active, so that subsequent project commands such as **Make** and **Run** operate on the project and displays either the **Application Tree** dialog or the **Project Editor** dialog depending on whether a .PRJ or .APP file was selected.

Edit Edits the active project file with the **Project Editor** dialog.



Clarion 4 (clubmgr4.app) - [clubmgr4.app - Module view]

Make

Generates souce code, then compiles and links the active application or project.

Run

Runs the active program after optionally saving and making the project. See *Run Configuration* for information on configuring the Run command.

Debug

Makes the active project then starts the appropriate debugger. See *Debug Configuration* for information on configuring the Debug command.

Make Statistics

Displays a statistical profile of the most recent make. The **Make Statistics** dialog shows information on the size of each module, including code and data size.

Run Configuration

The following menu options are toggle switches. Selecting the option changes the status of the switch from on to off, or from off to on.

Auto make before run
When checked (on), the **Run** command first initiates a **Make**.

File save before run
When checked (on), the **Run** command first initiates a **Save**.

Minimize on run
When checked (on), the **Run** command minimizes the development environment before running the application.

Wait for termination on run
When checked (on), the **Run** command suspends the development environment until after you terminate the application.

Debug Configuration

The following menu option is a toggle switch. Selecting the option changes the status of the switch from on to off, or from off to on.

Auto Resume on Debug
When checked (on), the **Debug** command applies the source files, breakpoints, and watch variables from the previous debug session (16-bit debugger only).

Application Only Commands—The Project Menu

Generate
Generates source code for any procedures in the project that have *changed* since the last generation.

Generate All
Generates source code for *all* procedures in the project.

Properties
Edits the application's Project System information with the **Project Editor** dialog.

Configuring the Environment

Clarion's development environment manages different types of files involved in the program development process. These files include data dictionaries (.DCT), applications and projects (.APP and .PRJ), source modules (.INC and .CLW), templates (.TRF, .TPL, and .TPW), and objects (.LIB, .DLL).

The Clarion environment contains tools for managing these various types of files. Each of the tools (Dictionary Editor/Viewer, Application Generator, Text Editor, Template Registry, driver/server/VBX registries) is separately configurable as described in this section.

Dictionary Editor Configuration

You can customize some of the default dictionary settings in the **Dictionary Options** dialog. These settings affect the appearance of the Dictionary Editor and the Dictionary Viewer. In addition, some settings affect the source code the Application Generator generates to manage data dictionary files and fields.

To customize the development environment's dictionary settings, choose **Setup ➤ Dictionary Options**. See *Dictionary Editor—Configuring the Dictionary Editor*.

Application Generator Defaults and Configuration

The **Application Options** dialog lets you specify default settings for each *new* application you create as well as for the active application. To access the dialog, choose **Setup ➤ Application Options**. See *Application Generator—Configuring the Application Generator.*

The **Template Registry** stores a list of all the templates available to you when building an application. To create an application, you must have at least one template class registered. To access the **Template Registry**, choose **Setup ➤ Template Registry**. See *Application Generator—Templates and the Template Registry*.

Text Editor Configuration

To personalize your editing environment, use the **Editor Options** dialog and, optionally, the ..\BIN\C5EDT.INI file. To open the **Editor Options** dialog, choose **Setup ➤ Editor Options**. Select the corresponding tab to set specific Text Editor options. See *Text Editor—Configuring the Text Editor*.

Environment INI File

The Clarion development environment has its own configuration (.INI) file that you can edit to customize the environment appearance and behavior. See the *Programmer's Guide* for more information on the environment .INI file.

Helper (Object) Program Registration

Both the development environment and the programs you create can call on other programs (typically DLLs) to accomplish certain tasks. In order to use these "helper" programs, the environment needs to know about them. You can provide the information the environment needs by registering these various programs.

Specifically, registerable programs include database drivers (programs that handle I/O for specific file systems such as Oracle, TopSpeed, Btrieve, etc.), database servers (programs that handle file definition and database administration for specific file systems such as Oracle, TopSpeed, Btrieve, etc.), and VBXs (programs that provide a variety of pre-programmed functionality—see *Custom Controls* in this book for more information).

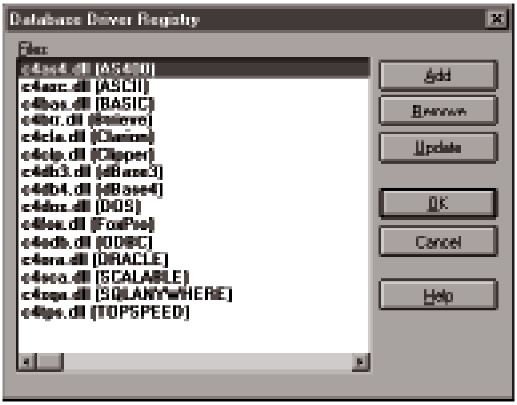
Database Driver Registry

To communicate with various file systems and databases, Clarion's runtime library uses database drivers. A database driver is a special .dll that translates Clarion file I/O commands into native database commands. There is a database driver for each file system Clarion supports (for example, ORACLE, MSSQL, Btrieve, Clipper, dBase, FoxPro, etc.). See *Database Drivers* in the *Application Handbook* for more information.

Before your application can use a particular database driver, the driver must be registered with the Clarion development environment. The in-the-box drivers are already registered when you install Clarion. You must register any add-on drivers.

Registering Database Drivers

To register a database driver, choose **Setup ➤ Database Driver Registry**.



Database Driver Registry

Files:

c4as4.dll [AS400]
c4asc.dll [ASCII]
c4bas.dll [BASIC]
c4btr.dll [Btrieve]
c4cla.dll [Clarion]
c4clp.dll [Clipper]
c4db3.dll [dBase3]
c4db4.dll [dBase4]
c4dos.dll [DOS]
c4fox.dll [FoxPro]
c4odb.dll [ODBC]
c4ora.dll [ORACLE]
c4sca.dll [SCALABLE]
c4sqa.dll [SQLANYWHERE]
c4tps.dll [TOPSPEED]

1. Start Clarion (DOUBLE-CLICK on the icon).
2. Choose **Setup ➤ Database Driver Registry**.

   This opens the **Database Driver Registry** dialog.
3. Press the **Add** button.

   This opens a Windows file dialog to the Clarion \BIN\ directory.

> **Tip: If this dialog is empty, your Windows Explorer is probably set to hide system files—this includes DLLs. Change your Windows setting to show system files to display the database driver .dlls.**

4. Highlight the database driver in the list box, then press the **Open** button.

   Database driver DLLs are named C5xxx.DLL, where xxx is a three character file system abbreviation.

> **Note: Registering the driver automatically registers both 16-bit and 32-bit drivers.**

Choose from:

C5asc.dll	ASCII
C5bas.dll	Basic
C5btr.dll	Btrieve
C5cla.dll	Clarion
C5clp.dll	Clipper
C5db3.dll	dBaseIII
C5db4.dll	dBaseIV
C5dos.dll	DOS
C5fox.dll	FoxPro
C5odb.dll	ODBC
C5ssql.dll	Scalable SQL
C5tps.dll	TopSpeed

Call TopSpeed Sales at (800) 354-5444 for information on additional client/server (SQL) database drivers.

5. Press the **OK** button to save the registry.

Maintaining the Driver Registry

In addition to the **Add** button, the **Database Driver Registry** provides an **Update** button and a **Remove** button for registry maintenance.

Update

Reloads information contained in the driver registry from *all* the registered drivers. If a driver DLL is not present, its registry entry is deleted.

Remove

You may want to remove unused drivers to keep your driver list to a manageable size. To remove a driver from the registry, highlight it, then press the **Remove** button.

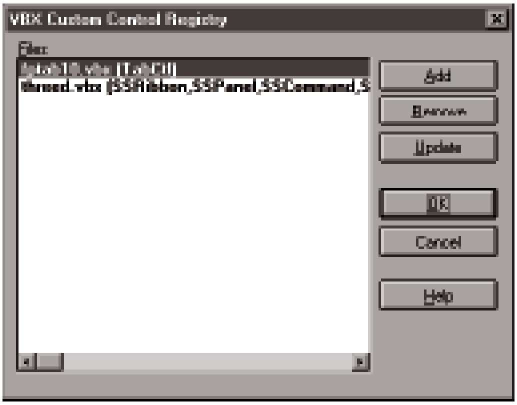
VBX Custom Control Registry

VBX controls are "add-in" controls sold by many third party vendors. These perform a wide variety of tasks, from sliders and gauge controls to TWAIN image capture. The **Window Formatter** lets you directly place these controls once you register the .VBX libraries. See *Custom Controls* for more information.

Before your application can use a particular VBX, it must be registered with the Clarion development environment.

Registering VBX Controls

To register a VBX, choose **Setup ➤ VBX Custom Control Registry**.



tabvb10.vbx [TabCtl]
threed.vbx [SSRibbon,SSPanel,SSCommand,S

1. Start Clarion (DOUBLE-CLICK on the icon).
2. Choose **Setup ➤ VBX Custom Control Registry**.

   This opens the **VBX Custom Control Registry** dialog.
3. Press the **Add** button.

   This opens the standard file selection dialog.
4. Navigate to the directory containing the VBX library.
5. Highlight the VBX in the list box, then press the **Open** button.
6. Press the **OK** button to save the registry.

Maintaining the VBX Registry

In addition to the **Add** button, the **VBX Custom Control Registry** provides an **Update** button and a **Remove** button for registry maintenance.

Update

Reloads information contained in the VBX registry from *all* the registered VBXs. If a VBX DLL is not present, its registry entry is deleted.

Remove

You may want to remove unused VBXs to keep your VBX list to a manageable size. To remove a VBX from the registry, highlight it, then press the **Remove** button.

Search Paths—the Redirection File

Clarion's development environment sets the working directory to the one in which the current application or project file resides. Additionally, Clarion uses the redirection file (\CLARION5.RED) to keep track of directories for the various application or project components. This redirection file tells the development environment where to find files and where to create new files.

Clarion checks for a (local) CLARION5.RED file in the application directory first. If it finds no local redirection file, it uses the default redirection file—installed by default to ..\CLARION5\BIN\CLARION5.RED.

> **Note: Backup files are always created in the directory where the original file is located.**

To edit the default redirection file, choose **Setup ➤ Edit Redirection File**. The text editor opens the ..\BIN\CLARION5.RED file for editing.

Redirection File Syntax

Each line of the redirection file is in the format:

$\texttt{filemask = directory}_1\ \texttt{[;directory}_2\texttt{]... [;directory}_n\texttt{]}$

The *filemask* is a file name or a file mask using the standard DOS wild card characters: * and ?.

The *directory* is a pathname identifying the directory or folder to search for the *filemask* files. The first *directory* is where any new *filemask* files are created. This is only true for files created and saved by the development environment, such as .OBJ, .DBD, .LIB, .EXE, and .CLW. The additional *directory* entries name additional search paths for existing *filemask* files.

Redirection Macros

The redirection file *directory* can contain macros. Redirection macros are labels surrounded by the percent sign (%). Whereever it encounters a redirection macro, the Clarion environment substitutes the macro's substitution value. You define redirection macros and their substitution values in the [Redirection Macros] section of the ..\BIN\CLARION5.INI file. For example to define a TEMP macro add the following to CLARION5.INI:

```
[Redirection Macros]
TEMP=c:TEMP
```

To use the TEMP macro, add the following to CLARION5.RED:

```
*.dbd = %TEMP%\obj
```

The Clarion environment expands the redirection line to:

```
*.dbd = c:TEMP\obj
```

The ROOT Macro

The redirection file *directory* can contain the predefined %ROOT% macro. By default, the %ROOT% macro expands to the drive and path one level above that from which the environment program (CLARION5.exe) is executing. For example, if the environment program is in C:\CLARION5\BIN, the environment substitutes C:\CLARION5 for %ROOT%. The default redirection file uses the %ROOT% macro to work with Clarion's default directory structure, regardless of where you install Clarion.

You may override the %ROOT% macro's default substitution value by explicitly setting a value in the CLARION5.INI file. For example:

```
[Redirection Macros]
ROOT=d:C5Beta2
```

Redirection File Sections

The redirection file can be separated into sections that are conditionally ignored or used depending on Project System settings. The sections are of the form [NNNNN] where NNNNN is one of the following:

Section Name	**Project System Switch**
16	Target OS=16-bit
32	Target OS=32-bit
DEBUG	Debug Mode
RELEASE	Build Release System
DEBUG16	Debug Mode + Target OS=16-bit
DEBUG32	Debug Mode + Target OS=32-bit
RELEASE16	Build Release System + Target OS=16-bit
RELEASE32	Build Release System + Target OS=32-bit
COMMON	none - COMMON is always used

Redirection lines within a section are only used if the section's corresponding Project System switches are true (COMMON is always true). Redirection lines without a section are always used. For example:

```
[DEBUG16]
*.obj = c:\test
[DEBUG32]
*.obj = c:\test32
[RELEASE]
*.obj = c:\release
[COMMON]
*.* = work
```

In this example if the project is 32-bit and the Build Release System box is checked, then .obj files are created in c:\release.

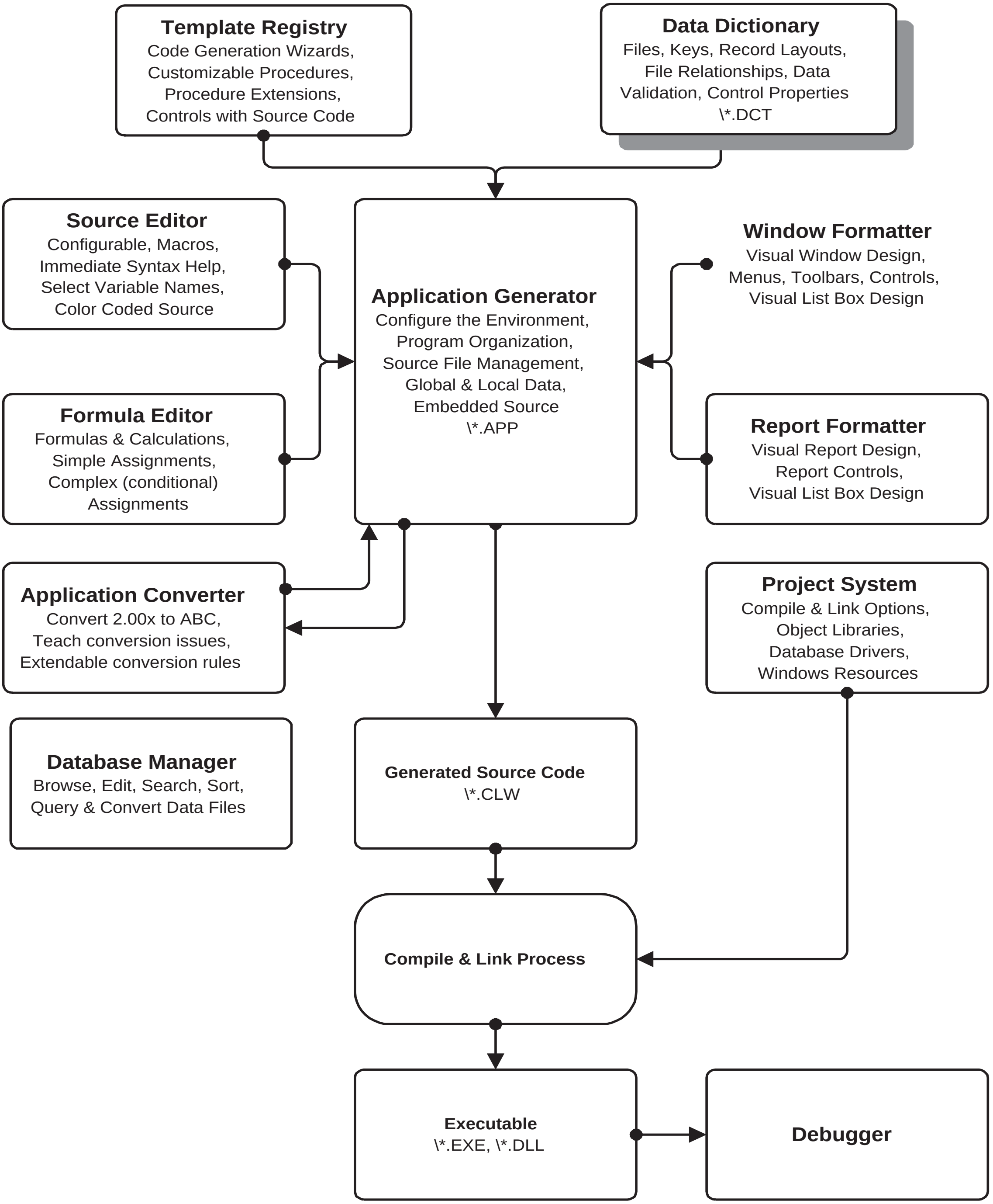
cleared, then .obj files are created in c:\test32 from line 2.

The Default Redirection File

```
-- Default Redirection for Clarion 5

[Debug16]
*.obj = %ROOT%\obj
*.lib = %ROOT%\obj
*.res = %ROOT%\obj
*.rsc = %ROOT%\obj
*.dbd = %ROOT%\obj

[Release16]
*.obj = %ROOT%\obj\release
*.lib = %ROOT%\obj\release
*.res = %ROOT%\obj\release
*.rsc = %ROOT%\obj\release

[Debug32]
*.obj = %ROOT%\obj32
*.lib = %ROOT%\obj32
*.res = %ROOT%\obj32
*.rsc = %ROOT%\obj32

[Release32]
*.obj = %ROOT%\obj32\release
*.lib = %ROOT%\obj32\release
*.res = %ROOT%\obj32\release
*.rsc = %ROOT%\obj32\release

[Common]
*.dll = .;%ROOT%\bin
*.tp? = %ROOT%\template
*.trf = %ROOT%\template
*.*   = .; %ROOT%\examples; %ROOT%\libsrc; %ROOT%\images; %ROOT%\template;
%ROOT%\convsrc
*.lib = %ROOT%\lib
*.obj = %ROOT%\lib
*.res = %ROOT%\lib
```

Note: The default redirection file is designed to work with Clarion's default directory structure. If you change the directory structure, you should make corresponding changes to the redirection file.

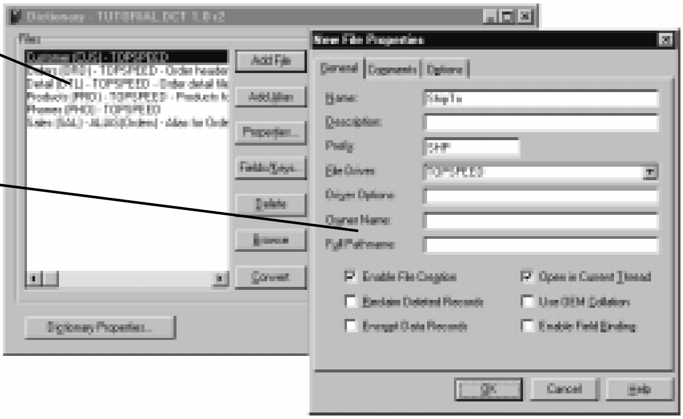
2 - DICTIONARY EDITOR


Template Registry
Code Generation Wizards, Customizable Procedures, Procedure Extensions, Controls with Source Code
Data Dictionary
Files, Keys, Record Layouts, File Relationships, Data Validation, Control Properties
*.DCT
Source Editor
Configurable, Macros, Immediate Syntax Help, Select Variable Names, Color Coded Source
Application Generator
Configure the Environment, Program Organization, Source File Management, Global & Local Data, Embedded Source
*.APP
Window Formatter
Visual Window Design, Menus, Toolbars, Controls, Visual List Box Design
Formula Editor
Formulas & Calculations, Simple Assignments, Complex (conditional) Assignments
Report Formatter
Visual Report Design, Report Controls, Visual List Box Design
Application Converter
Convert 2.00x to ABC, Teach conversion issues, Extendable conversion rules
Project System
Compile & Link Options, Object Libraries, Database Drivers, Windows Resources
Database Manager
Browse, Edit, Search, Sort, Query & Convert Data Files
Generated Source Code
*.CLW
Compile & Link Process
Executable
*.EXE, *.DLL
Debugger

About This Chapter

This chapter shows you how to set up a data dictionary. The templates and the Application Generator rely on the information in the data dictionary to generate Clarion language statements for a wide range of your application's functionality. This chapter explains:

- What a data dictionary is and does.
- How the data files you design and options you choose in the dictionary can determine the efficiency of your application's data storage. This includes a brief discussion of relational database theory.
- How to add file definitions to a data dictionary by importing from existing data files.
- How to set default data validation, control formatting, and entry options for end user data entry.
- How to define file relationships and enforce relational integrity (RI) between the data in related files.
- How to specify default screen control options. Clarion even lets you specify controls such as spin boxes or custom list boxes from within the data dictionary. The Application Generator automatically uses these controls whenever the field is referenced.



Dictionary - TUTORIAL.DCT 1.0.x2

Files
Customer (CUS) - TOPSPEED
Orders (ORD) - TOPSPEED - Order header
Detail (DTL) - TOPSPEED - Order detail file
Products (PRO) - TOPSPEED - Products file
Phones (PHO) - TOPSPEED
Sales (SAL) - ALIAS(Orders) - Alias for Orders

New File Properties

General | Comments | Options

Name: ShipTo
Description:
Prefix: SHP
File Driver: TOPSPEED
Driver Options:
Owner Name:
Full Pathname:

☑ Enable File Creation
☑ Open in Current Thread
□ Reclaim Deleted Records
□ Use OEM Collation
□ Encrypt Data Records
□ Enable Field Binding

The Dictionary dialog lists all files in the database, including file aliases.

Define each data file in a separate File Properties dialog.

- How to configure the Dictionary Editor.

About the Data Dictionary

The Data Dictionary is the central repository for information about your application's files, fields, keys, and file relationships. The information stored includes how and where the data is stored on disk, as well as how the data is presented to end users on reports and computer screens.

The Dictionary file (.DCT) stores

- file names
- file descriptions
- file key and index definitions
- file relationship definitions
- database connection information
- relational integrity (RI) rules
- field data types
- field descriptions
- field balloon help
- field status bar messages
- field validation rules
- field entry pictures and formatting
- default screen and report controls for each field
- much more

Benefits of Using a Data Dictionary

The benefit of having all this information stored in a central place is that it saves huge amounts of time in developing and maintaining applications. Plus it can give your application a consistent look and feel so that end users have shorter learning curves. These benefits happen because fields, file relationships, validation rules, etc. are defined only once, instead of being redefined each time a field is referenced in the application.

The information stored in the Data Dictionary defines a *default* method for handling data. By defining defaults here in the Data Dictionary, you establish a method of handling each file and field. This method is used *every* time you reference the file or field with the Application Generator. This means *you design your data handling method only once*, no matter how many times your application makes use of a field, and no matter how many applications use this dictionary; however *you still retain the ability to modify this default method* in any particular case.

Dictionary Editor Functions

Following is a list of the main functions the Dictionary Editor performs and the dialog that performs each function.

- ❏ Manage files and file relationships in the **Dictionary** dialog.
- ❏ Choose the file driver and specify the names and locations of data files in the **File Properties** dialog.
- ❏ Define specific fields and the types of data they hold in the **Field Properties** dialog.
- ❏ Define specific keys and their components in the **Key Properties** dialog.
- ❏ Define specific file relationships in the **Relationship Properties** dialog.

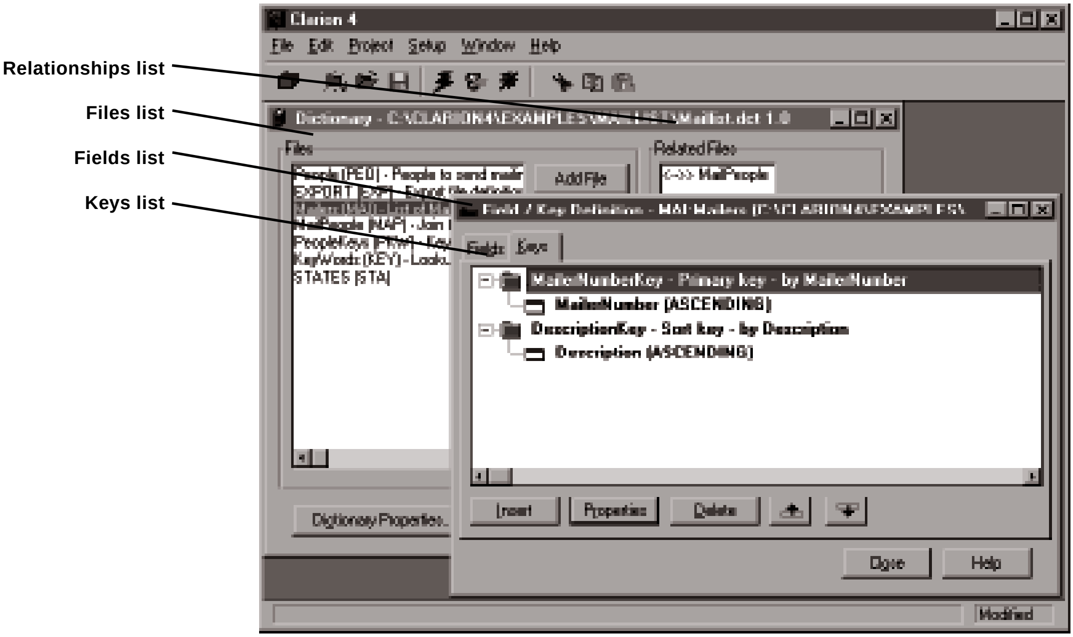
Two Entries to theDictionary Editor

You can access, view,and manipulate Clarion's Dictionary Editor in two ways: through the **Dictionary** dialog, or through the Dictionary Toolbox.

> **Tip: You can configure the information displayed in the Dictionary dialog and the Dictionary Toolbox. See *Configuring the Dictionary Editor* for more information.**

The Dictionary Dialog

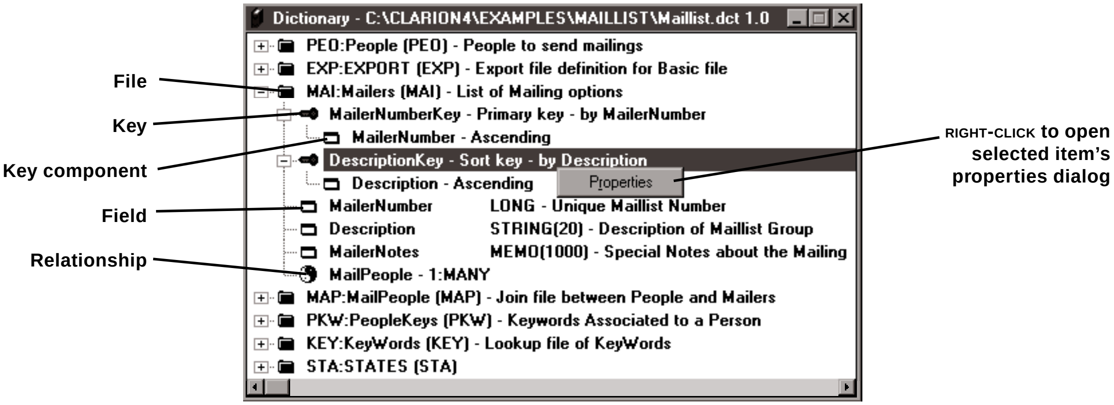
The **Dictionary** dialog provides a hierarchy of dialogs that display and update information about your database. These dialogs organize the database information into discrete lists—a file list, a relationship list, plus a field list and a key list for each file. Each list item has a properties dialog for manipulating the item's properties.


Relationships list
Files list
Fields list
Keys list
Clarion 4
File Edit Project Setup Window Help
Dictionary - C:\CLARION4\EXAMPLES\MAILLIST\Maillist.dct 1.0
Files
People [PEO] - People to send mail
EXPORT [EXP] - Export file definitio
Mailers [MAI] - List of Ma
MailPeople [MAP] - Join t
PeopleKeys [PKY] - Key
KeyWords [KEY] - Looku
STATES [STA]
Add File
Related Files
<-->> MailPeople
Dictionary Properties...
Field / Key Definition - MAI:Mailers (C:\CLARION4\EXAMPLES\
Fields
Keys
MailerNumberKey - Primary key - by MailerNumber
MailerNumber (ASCENDING)
DescriptionKey - Sort key - by Description
Description (ASCENDING)
Insert
Properties
Delete
Close
Help
Modified

The **Dictionary** dialog can add new items to the active data dictionary (files, fields, keys, etc.), can update all existing items, and can call the Database Manager to browse, edit, and convert data files. However, you cannot use the **Dictionary** dialog to access a dictionary that is already in use within the development environment. The environment prevents you from doing this. To open the **Dictionary** dialog, choose **File ➤ Open,** then from the **Files of type** drop-list, select *Dictionary (*.dct)*.

The Dictionary Toolbox

The Dictionary toolbox provides a hierarchical list representing your database. This list shows database files, keys, key components, fields, and relationships in an expanding hierarchical tree. Each list item has a properties dialog for manipulating the item’s properties. RIGHT-CLICK on an item to open its properties dialog. CLICK on the plus (+) sign to expand the list; CLICK on the minus (-) sign to contract it.



Dictionary - C:\CLARION4\EXAMPLES\MAILLIST\Maillist.dct 1.0

```
+ PEO:People (PEO) - People to send mailings
+ EXP:EXPORT (EXP) - Export file definition for Basic file
- MAI:Mailers (MAI) - List of Mailing options
    - MailerNumberKey - Primary key - by MailerNumber
        MailerNumber - Ascending
    - DescriptionKey - Sort key - by Description
        Description - Ascending
    MailerNumber      LONG - Unique Maillist Number
    Description       STRING(20) - Description of Maillist Group
    MailerNotes       MEMO(1000) - Special Notes about the Mailing
    MailPeople - 1:MANY
+ MAP:MailPeople (MAP) - Join file between People and Mailers
+ PKW:PeopleKeys (PKW) - Keywords Associated to a Person
+ KEY:KeyWords (KEY) - Lookup file of KeyWords
+ STA:STATES (STA)
```

The Dictionary toolbox can update all the existing items in the active data dictionary except the dictionary properties and any other properties that are already in use within the development environment. The Dictionary toolbox cannot add new items (new files, fields, keys, etc.) and does not call the Database Manager to browse, edit, or convert files.

You can use the Dictionary toolbox to access a dictionary that is already in use within the development environment, with the noted limitations. From the **Dictionary** dialog, choose **Edit ➤ View as Toolbox**, or from the **Application Tree** dialog, choose **Application ➤ View Dictionary**.

Designing Your Dictionary and Your Database

This section provides a quick review of relational database theory. Planning and organizing your application's database design up front can result in a more efficient application for the end user, not to mention saving hours of coding and maintenance time.

The relational model concerns itself with three aspects of data management: *structure*, *integrity,* and *manipulation*. For our purposes, we will discuss the three practical requirements of these aspects: data normalization, keys, and relational operations.

Normalization

At its simplest, data normalization means that a data item should be stored at only one location. To avoid duplication within the database, a good design splits data into separate files.

For instance, assume a very simple order-entry system storing the following data:

```
Customer Number
Customer Name
Customer Address
ShipTo Address
Order Number
Order Date
Product Number
Quantity Ordered
Unit Price
```

You could store all the data in each record of one file, but that would be inefficient (unless the business has *no* repeat customers). A second order from a customer would repeat all the Customer data, for example. To eliminate this duplication, you could split the data into three files:

Customer File:	Order File:	Item File:
Customer Number	Order Number	Product Number
Customer Name	ShipTo Address	Quantity Ordered
Customer Address	Order Date	Unit Price

This organizes the data in a logical scheme and eliminates duplication. The process of relating each record to another record in another file requires adding fields to at least two of the files, so that the files can share common values. This will be discussed in a section below.

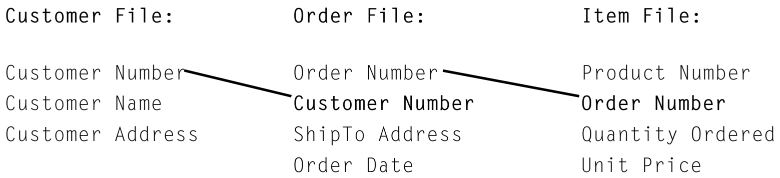
Strict relational theory specifies that:

- The database consists of one or more *tables*, which roughly correspond to Clarion's Data Dictionary *files*.

- The table consists of columns (which at the file level we refer to as *fields*) and zero or more rows (*records*).
- Each record contains exactly one value for each field.

Keys

In the simple order-entry system above, to *relate* the records in the customer, order and item files to one another, we could add one field each to two of the files as follows:



```
Customer File:          Order File:             Item File:

Customer Number         Order Number            Product Number
Customer Name           Customer Number         Order Number
Customer Address        ShipTo Address          Quantity Ordered
                        Order Date              Unit Price
```

Relational database theory states:

- A primary key should exist for each table. A primary key is a unique field or unique combination of fields. The primary key must not accept a null or blank value.
- A foreign key can match the primary key in another table. If table “A” includes a foreign key that matches table “B’s” primary key, then every value in the key in table “B” must either be equal to a value in the primary key in a record in “A,” or be null.

In the example above, the Customer Number is the primary key (there could be two “John Smith’s,” but not two customer #1001’s). The Customer Number field is added to the Order file, as a foreign key.

You can define three types of relationships between files:

- *One-to-Many*. One record in a file relates to many in another. In the example above, a single customer number may relate to many records in the Order file. In business database applications, this is the most common relationship. It is also referred to as a Parent-Child relationship.
- *One-to-One*. Exactly one record in a file relates to one record in another file. This is best suited for when one file may or may not have data in some fields. If all the fields were in one file, disk space would be wasted on empty fields.

  In the example above, if the ShipTo address was rarely different than the Customer Address, you could place it in another file.
- *Many-to-Many*. Multiple records in a file relate to multiple records in another. To apply it to the example, assume the Order-

Entry system were made to fit a manufacturing concern which buys parts and makes products. If a part could be used in many different products, and a product could use many parts, two additional files might look like:

```
Parts File:             Product File:

Part Number             Product Number
Part Description        Product Description
```

Relational Operations

Relational database theory provides a set of operators for manipulating data. The three operations that theoreticians specify for relational database systems are *Select*, *Project*, and *Join*. A system does not have to explicitly support the statements as long as it supports their functionality. For theoretical purposes, a table simply consists of a set of columns (or fields), plus zero or more rows (records) of data values.

- A *Select* extracts a row subset of a given table—in other words, a subset of records which satisfy a given condition.
- A *Project* extracts a column subset of a given table—in other words, a subset of specified fields, which then eliminates extraneous records (example below).
- A relational *Join* takes two tables and joins them together to form a new, wider table.

Select (not the same as SQL's "Select") provides the means to evaluate a table and extract a record or records. The database must have the ability to evaluate the information a single record at a time—in isolation—without looking at the other rows. In the example, extracting a record or records (spanning all files) that meet the condition "Customer Number = 100" is an example of a relational select.

Project extracts unique values by field. In the example above, assuming that the Item file has many duplicates, to Project the file "Item" over the field "Product Number" yields a new table of all the products ordered by customers (not necessarily matching all products made, in the Product file). All the products sold would have one and only one listing.

Join: Going back to the example, to work with all the combinations of parts and products possible, there must be a special relationship between these two files. The solution is to define a third file, called a "Join" file. This file creates two One-to-Many relationships. The relationships between the three files would be defined:

```
Parts File:

Part Number                 (Primary key)
Part Description

Parts2Prod File:

Part Number                 (1st Primary key component and Foreign key)
Product Number              (2nd Primary key component and Foreign key)
Quantity Used

Product File:

Product Number              (Primary key)
Product Description
```

The Parts2Prod file has a multiple component Primary key and two Foreign keys. The relationship between Parts and Parts2Prod is One-to-Many. The relationship between Product and Parts2Prod is also One-to-Many. This makes the Join file the “middleman” between two files with a Many-to-Many relationship.

Usually a Join file contains additional information. In this example, the Quantity Used logically belongs in the Parts2Prod file.

The Dictionary Editor

The Clarion language supports the three aspects of data management that relational database theory concerns itself with. The Dictionary Editor is a tool for planning the structure and integrity of the database, two of the relational model’s “rules.” The Dictionary Editor also lets you “preconstruct” some of the relational operations specified by database theorists; Clarion language statements handle the remaining operations.

- The Dictionary Editor lets you easily set up the proper database structure by defining *files, fields, and relationships*.
- The Dictionary Editor lets you easily plan both *primary and foreign keys* for your database, as per the relational model’s integrity rules.
- The Dictionary Editor lets you easily implement *referential integrity (RI) constraints* that automatically keep related files in sync by “cascading” changes across files and by “restricting” or limiting changes or deletions that would cause inconsistencies between files.

Creating a Data Dictionary

This section provides an overview of the *general* process of creating a data dictionary, that is, defining files, fields, keys, and file relationships. This overview procedure leaves many options at their defaults and provides basic descriptions of what the dialog boxes in the Dictionary Editor do. The dialog boxes and the options they contain are explained more fully in the remainder of this chapter.

Define the files in your database

1. Choose **File ➤ New ➤ Dictionary** from the development environment menu.
2. Specify the path (**Folders**) and **File Name** for your dictionary file, then press the **Save** button.

   This opens the **Dictionary** dialog.
3. Press the **Add File** button, then, when asked if you want to use Quick Load, press the **No** button.

   This opens the **New File Properties** dialog. See *Quick Load* for a brief discussion of using Quick Load to add files to your data dictionary.
4. On the **General** tab, type the **Name**, the **Prefix**, and choose the **File Driver** for your data file, then press **OK** to close the dialog.

   The prefix is prepended to field names to guarantee their uniqueness. The File Driver is simply an indicator of the file system your data belongs to: TopSpeed, Btrieve, Clipper, etc.
5. Repeat steps 3 and 4 for additional files in your database.

Define the fields in each file

1. Press the **Fields/Keys** button to open the **Field/Key Definition** dialog.
2. On the **Fields** tab, press the **Insert** button to define a new field.

   This opens the **New Field Properties** dialog.
3. On the **General** tab, type in the field **Name**, choose the **Data Type**, and specify its length in **Characters**.
4. Select the **Validity Checks** tab, then choose an option for data entry validation.
5. Select the **Window** tab to specify default screen controls for your application's windows and dialogs.

   You can define the type of control (entry field, spin box, radio button, check box, etc.), its size, position, color, messages, help ids, etc. See *Controls and Their Properties* for more information.

6. Select the **Report** tab to specify default report controls for your application's reports.

7. Press **OK** to end this field and define the next.

   The **New Field Properties** dialog reopens, ready for the next field.

8. Repeat steps 3 through 7 for additional fields within this file.

   After each field, a blank **New Field Properties** dialog opens, ready to define the next field.

9. After adding the last field, press the **Cancel** button in the **New Field Properties** dialog to return to the **Field/Key Definition** dialog.

Define the Keys in your files

1. Select the **Keys** tab, then press the **Insert** button to open the **New Key Properties** dialog.

2. On the **General** tab, type the **Key Name**.

3. Select the **Fields** tab, then press the **Insert** button to open the **Insert Key Components** dialog.

   A key is based on one *or more* fields in your file. The **Insert Key Components** dialog lets you specify which fields make up the key.

4. Highlight a component field from the list by CLICKING on it, then press the **Select** button.

   Press the **Insert** button again to add any additional component fields to your key.

5. Press **OK** to end this key and define the next.

   The **New Key Properties** dialog reopens, ready for the next key.

6. Repeat steps 2 though 5 for other keys in this file.

7. After adding the last key, press the **Cancel** button to return to the **Field/Key Definition** dialog.

8. Press the **Close** button to return to the **Dictionary** dialog.

Define the relationships between your files

1. Select a file to relate to another, then press the **Add Relation** button on the Related Files side of the **Dictionary** dialog.

2. Choose the **Type** of relationship from the drop-down list.

3. Choose the **Related File** from the drop-down list.

   This is simply the *other* file in the relationship.

4. From the respective drop-down lists, choose the **Primary Key** for the original file and the **Foreign Key** for the related file.

5. Press the **Map By Name** or **Map By Order** button to establish a link between the primary and foreign keys.

6. Select appropriate Referential Integrity (RI) constraints from the **On Update** and **On Delete** drop-down lists.

7. Press the **OK** button to return to the **Dictionary** dialog.

Save the Dictionary

1. Choose **File ➤ Save As** to save the .DCT file.

Opening the Dictionary Editor

You generally create a data dictionary as the first step in creating an application. Therefore, you will access it first from the main menu.

Create a new dictionary file

1. Choose **File ➤ New ➤ Dictionary** from the development environment menu.

   This opens the standard Windows file dialog.

2. Specify the path (folder) and **File name** for your dictionary file, then press the **Save** button.

Open an existing dictionary file

1. Choose **File ➤ Open,** then from the **Files of type** drop-list, select the *Dictionary (*.dct)*.
2. Change drives or directories as necessary and locate the dictionary file you wish to open. DOUBLE-CLICK on its name in the **File Name** list, or select it then press the **Open** button.

   or

1. From the **Application Tree** dialog, choose **Application ➤ View Dictionary**.

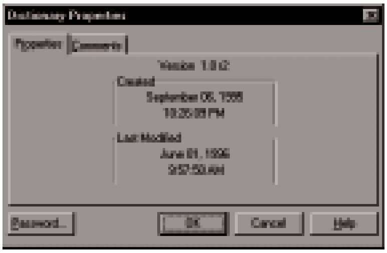
   See *Two Entries to the Dictionary Editor* for more information.

You may use the same data dictionary for more than one application. An application, however, can only have one data dictionary.

> **Tip:** **Clarion 5 automatically reads and converts Clarion for DOS3.007 data dictionaries (and above). It will import all attributes except display size attributes for memo fields. Also, because relational model rules are more strictly enforced in Clarion 5 , some relationships may not be complete.**

Add description to the data dictionary

1. Press the **Dictionary Properties** button at the bottom of the dialog.
2. On the **Comments** tab, type the description in the space provided.



Version 1.0.c2

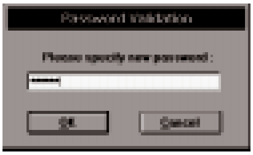
Created
September 05, 1995
10:26:09 PM

Last Modified
June 01, 1996
9:57:50 AM

The description is solely for your convenience, and has no effect on the application. It is useful when other programmers take over your project, or for when you return to the project after an absence.

Add a password to the data dictionary

1. Press the **Password** button.
2. When the **Password Validation** dialog appears, type a password in the space provided, then press the **OK** button.
3. When the **Password Verification** dialog appears, type the *same* password, then press the **OK** button.



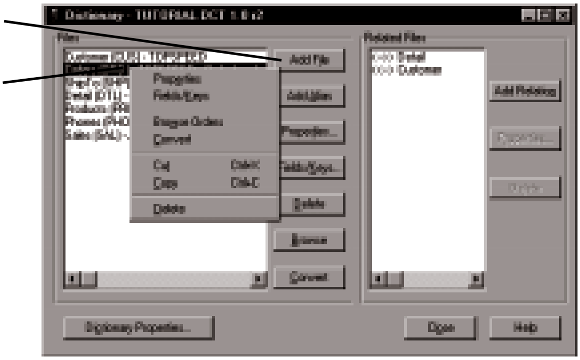
Please specify new password :

The password can help protect your application from unauthorized access.

Adding Files to the Dictionary

The first function of the dictionary is to specify the data *files* for the application. Define the files by adding them to the left side of the **Dictionary** dialog. Either of the two “Add” buttons to the right of the list allow you to add to the list.

> **Tip: You can configure the information displayed in the Dictionary dialog. See *Configuring the Dictionary Editor* for more information.**



Press the Add File button to add a new file.

RIGHT-CLICK **on the file name or the relationship to access a popup menu of choices.**

- ❏ *To add a file to the files list*, press the **Add File** button. This opens the **New File Properties** dialog. Alternatively, you can create a file definition from existing data with the **File ➤ Import File** command.
- ❏ *To add an alias to the list*, press the **Add Alias** button. This displays the **New Alias** dialog. See the *Adding a File Alias* section, below.

Quick Load

When you press the **Add File** button you are optionally prompted to use Quick Load to add your file to the data dictionary (see *Configuring the Environment—Application Defaults and Configuration* for instructions on disabling this prompt). Quick Load lets you specify only the most basic information about your file and its fields; Quick Load supplies all other required attributes by default. This is especially useful for quickly producing a working application that can be fine-tuned later.

Alternatively, if you have done extensive project planning and specification, you may prefer to add your file without using Quick Load so you can take advantage of the many file and field attributes supported by Clarion’s data dictionary. For example, the data dictionary supports data entry validation, but validation defaults to none if you use Quick Load. The following section assumes you are not using Quick Load to add the file. See *Adding a File with Quick Load* in the *Getting Started* book for more information.

Importing File Definitions

The Dictionary Editor lets you quickly add a data file definition to the dictionary by creating a definition based on an existing data file. We strongly recommend using the import command when accessing existing data because it is the fastest and most accurate way to define existing data files.

1. From the Dictionary dialog, select **File ➤ Import File**.

   This opens the **Select File Driver** dialog.

2. Pick a file driver from the drop-down list, then press the **OK** button.

   What happens at this point depends on the driver you choose. Typically, SQL drivers or other client/server drivers require login information, then let you choose one or more tables whose definitions to import. The ODBC driver requires you to specify an ODBC data source whose definition to import. The non-client/server drivers require that you specify a file whose definition to import. See *Database Drivers* in the *Programmer's Guide* for driver specific import instructions.

3. Once you have chosen the file, table, or data source to import, press the **OK** button to close each dialog.

   The Dictionary Editor creates the file definition then opens the **Edit File Properties** dialog. You may make changes to the imported file definition; however, in most cases it is not necessary.

4. Make any changes to your new file definition, then press the **OK** button.

   The Import Wizard adds the file definition to the dictionary, including its field and key definitions.

Tip: Some file, field, and key properties vary with the file driver. For example, for dBase drivers, the NAME attribute on a numeric field can define its precision. See *Database Drivers* in the *Programmer's Guide* for the specific file driver for more information.

File Properties

Define or change a file definition with the **New/Edit File Properties** dialog. This dialog lets you desscribe general file characteristics and choose its file driver.

Once the file is added, you may define fields, keys, set relationships, and other properties for the data file. The Application Generator uses this information to write the FILE structure declaration (see *FILE* in the *Language Reference*), plus file I/O routines as required by your application.

General

Usage

Mark the structure as a File, Global Data group, or Field Pool. You can define relationships for files, but not for Global or Pool structures.

File The structure represents a FILE. The Application Generator generates a FILE declaration as well as code to read and write the FILE. You can use the fields in the FILE as the parents of derived fields. See **Derived From** in the **Field Properties** dialog.

Global The structure represents a group of global data declarations. The Application Generator generates a global data declaration for each field in the structure. You can use the global fields as the parents of derived fields. See **Derived From** in the **Field Properties** dialog. This selection enables the **Generate Last** check box.

Pool The structure represents a Field Pool. The Application Generator generates no code for this structure. You can use the fields in the pool as the parents of derived fields. See **Derived From** in the **Field Properties** dialog.

Generate Last Check this box to have the ABC Templates generate global data field declarations last within the program module (after file declarations). Clear the box to declare global data before the file declarations.

This box is only enabled for global data and is not recognized by the Clarion Templates.

Name

Type the file name as you wish to refer to it in your source code. This serves as the label for the Clarion FILE structure. Specify a valid Clarion label (see the *Language Reference*). If you do not specify a filename in the **Full Pathname** field, this label also serves as the filename for the file.

Tip: You can specify variable file names to take advantage of Novell Paths.

1. Prefix the file name variable with “!” For example, !Glo:CustFile.

2. Create the variable (see *Application Generator—Global Variables*).

3. Embed the following code before accessing the file (see *Application Generator—Embedded Source Code*).

```
Glo:CustFile = Server/Vol:\dir\dir\Cust.btr
```

Description

Enter a string description for the file. Clarion automatically displays the descriptions in certain dialogs, allowing you to quickly recognize the file contents.

Prefix

As you enter the data file **Name**, Clarion automatically extracts the first three letters to use as a label prefix when referring to the file. Optionally specify up to 14 characters of your choice in this field.

The prefix allows your application to distinguish between the same variable names occurring in different data structures. A field called *Invoice* may exist in two different files: *Orders* and *Sales*. By establishing a unique prefix for *Orders* (ORD) and *Sales* (SAL), the application may refer to fields as ORD:Invoice and SAL:Invoice. See *Field Qualification Syntax* in the *Language Reference* for more information.

File Driver

Specify the file type: TopSpeed, Clarion, Btrieve, ASCII, etc. When using the Application Generator, Clarion automatically links in the correct database file driver library. See *Database Drivers* in the *Programmer’s Guide* for a discussion of the relative advantages of each driver.

Remember that file systems vary in their support of some of the attributes which you add to the FILE structure in this dialog box. See *Database Drivers* in the *Programmer’s Guide* for more information on each attribute.

Driver Options

A driver string or strings specific to the file driver. A forward slash precedes each string. This conveys additional instructions to the file driver and generates the second parameter for the DRIVER attribute of the FILE statement. See *Database Drivers* in the *Programmer’s Guide for* specific information on the strings available for each file driver.

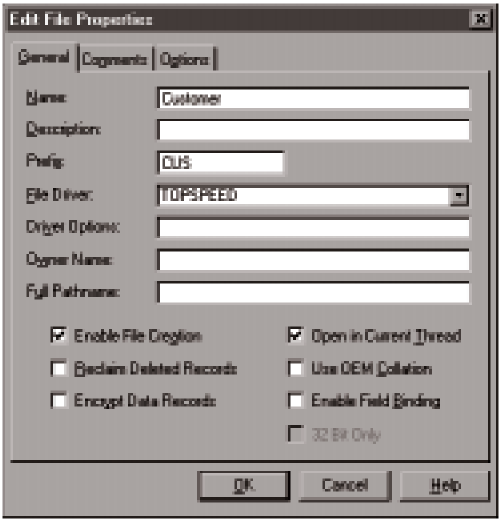
Owner Name

A string constant or the label of a variable specifying the password or connection string (SQL) for access to the file. This adds the OWNER attribute to the FILE statement. For additional security you can check the **Encrypt** box (below).

We recommend using a variable password that is lengthy and contains special characters because this more effectively hides the password value from anyone looking for it. For example, a password like “dd....#$...*&” is much more difficult to “find” than a password like “SALARY.”

Tip: To specify a variable instead of a constant OWNER attribute, type an exclamation point (!) followed by the variable name. For example: !GLO:ConnectString.

See *Database Drivers* in the *Programmer’s Guide* for more information on SQL connect strings.



Name: Customer
Description:
Prefix: CUS
File Driver: TOPSPEED
Driver Options:
Owner Name:
Full Pathname:

☑ Enable File Creation
☑ Open in Current Thread
□ Reclaim Deleted Records
□ Use OEM Collation
□ Encrypt Data Records
□ Enable Field Binding
□ 32 Bit Only

Full Pathname

The fully qualified pathname for the data file. You may omit the file extension—Clarion will supply the correct extension depending on the file driver chosen. This supplies the parameter for the NAME attribute.

If you omit this field, Clarion supplies a default by appending the first eight letters in the **Name** field to the active path.

When using the TopSpeed driver, if you wish to store multiple tables in a single physical file, separate the file and table names with “`\!`,” as in `TUTORIAL\!ORDERS`. This refers to the ORDERS table in the TUTORIAL.TPS file. See *Database Drivers* in the *Programmer’s Guide* for more information.

When using an ODBC driver to define a FILE such as Microsoft Access, which can store multiple tables in a single file, place the table name in this field. Typically, the name of the physical file which includes the table is listed in the ODBC.INI file; the ODBC driver manager provides this information to the driver. See *Database Drivers* in the *Programmer's Guide* for more information.

Tip: To specify a variable name instead of the actual file name, type an exclamation point (!) followed by the variable name. For example: !FileNameVar.

Enable File Creation

Optionally specify that the application should create the data file if it does not exist at run time. This adds the CREATE attribute to the FILE statement. You may override this setting for all files or for each file within an application. See the *Application Handbook—Template Overview—File Handling*.

Reclaim Deleted Records

This option depends on the file driver. It specifies that the file driver reuse file space formerly taken up by deleted records. Otherwise, the application adds new records to the end of the file. This adds the RECLAIM attribute to the FILE statement.

Encrypt Data Records

Toggles file encryption for the file systems that support it. You must also specify an **Owner Name** (see above). This adds the ENCRYPT attribute to the FILE statement. See *Database Drivers* in the *Programmer's Guide* to see which file systems support encryption.

Encrypting the file encodes its data so that only your application can decode it.

Open in Current Thread

Optionally specify that each execution thread that uses this file allocates memory for its own separate record buffer. We recommend this for multiple document (MDI) applications. You may override this setting for all files or for each file within an application. See the *Application Handbook—Template Overview—File Handling*. This adds the THREAD attribute to the FILE statement.

Use OEM Collation

The OEM attribute specifies that the FILE on which it is placed contains non-English language string data. These strings are automatically translated from the OEM ASCII character set data contained in the file to the ANSI character set for display in Windows. All string data in the record is automatically translated from the ANSI character set to the OEM ASCII character set before the record is written to disk. This adds the OEM attribute to the FILE statement.

The specific OEM ASCII character set used for the translation comes from the DOS code page loaded by the COUNTRY.SYS file. This makes the data file specific to the language used for that code page, and means the data may not be usable on a computer with a different code page loaded.

Enable Field Binding

Optionally specify that all fields in the RECORD structure are available for use in dynamic expressions at run time (using BIND and EVALUATE). The compiler allocates memory to hold the full Prefix:Name for each variable. This adds the BINDABLE attribute to the FILE statement.

Export Fields

Check this box to ?

32 Bit Only

Optionally specify that the RECORD structure is too large for 16-bit applications.

Freeze

Check this box to prevent any derived fields in the file from refreshing. See **Derived From** in the **Field Properties** dialog. See also the **Refresh Field, Refresh File, Refresh Dictionary,** and **Distribute Field** menu commands.

Comments

Comments

Select the **Comments** tab to type a text description of up to 1000 characters.

Options

Do Not Auto-Populate This File

Checking this box tells the Application Wizard *not* to generate Browse, Update, or Report procedures for this file.

User Options

The text typed into this field is available to any Utility Templates that processes this file in the %FileUserOptions symbol. The Utility Templates determine the proper syntax for these user options. See also TOOLOPTIONS and %FileToolOptions in the *Programmer’s Guide*.

Adding File Aliases to the Dictionary

An alias creates a second reference for a file without duplicating the file on disk. You can add an alias only for files already in the Dictionary.

Why Use Aliases

A file alias creates an additional record buffer for a file *on the same thread*. That is, an alias lets you define and use two or more *different* relationships between the *same two files on the same thread*. This is really the only compelling reason to use file aliases, since aliases use additional memory, resources, and can create confusion.

Tip: When using aliases it is best to use a file driver that stores keys internally, such as TopSpeed or Btrieve, to conserve file handles.

For example, let's say your hospital application has a patient file and a doctor file. A patient has several doctors: an admitting doctor, a primary doctor, and a surgeon. In database terms, the patient record has three *different* fields containing doctor IDs, and all three are linking fields to the doctor file (thus, three relationships between the same two files). If you want to automatically display all the patient's doctors on the same window, you need a record buffer for each link, otherwise, you can show only one doctor at a time—the last one retrieved.

By defining aliases for the doctor file, you can supply additional buffers to hold more than one doctor record at a time. Do not confuse this with the THREAD attribute for a file. The THREAD attribute provides for a separate record buffer for each *different* thread, whereas an alias provides an additional record buffer on the *same* thread.

The File Alias Dialog

To add an alias, press the **Add Alias** button in the **Dictionary** dialog to open the **New File Alias** dialog. To modify the alias properties at a later time, highlight the alias name on the **Dictionary** dialog list, then press the **Properties** button to open the **Edit File Alias** dialog.

You can edit the fields and keys for the alias by pressing the **Fields/Keys** button. The **Field/Key Definition** dialog lists the fields and keys for the *original* file; any changes you make will update the originals.

Tip: When using aliases, you must open the file in Share mode.

The **File Alias** dialog includes the following tabs and fields:

General

Name

Type an alias “name”, as you wish to refer to it in your source code. The name must be a valid Clarion label.

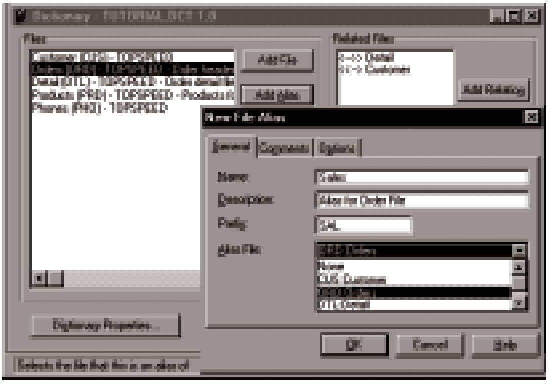
Description

Enter a text description for the alias. Clarion displays the descriptions in dialogs such as the **Dictionary** dialog.

Prefix

As you enter the alias **Name**, Clarion automatically extracts the first three letters to use as a label prefix when referring to the alias. Optionally specify up to 14 characters of your choice in this field.

The prefix lets your application distinguish between the same variable names occurring in different data structures. A field called *Invoice* may exist in two different files: *Orders* and *Sales*. By establishing a unique prefix for *Orders* (ORD) and *Sales* (SAL), the application may refer to fields as ORD:Invoice and SAL:Invoice. See *Field Qualification Syntax* in the *Language Reference* for more information.



Customer (CUS) - TOPSPEED
Orders (ORD) - TOPSPEED - Order header
Detail (DTL) - TOPSPEED - Order detail file
Products (PRO) - TOPSPEED - Products file
Phones (PHO) - TOPSPEED

1-->> Detail
1<-1 Customer

Name: Sales
Description: Alias for Orders File
Prefix: SAL
Alias File: ORD:Orders

None
CUS:Customer
ORD:Orders
DTL:Detail

Selects the file that this is an alias of

Alias File

Choose a file from the drop-down list. This is the *original* file that the alias “references.” The drop-down list shows only the files previously defined in the **Dictionary** dialog.

Comments

Comments

Select the **Comments** tab to type a text description of up to 1000 characters.

Options

Do Not Auto-Populate This File Alias

Checking this box tells the Application Wizard *not* to generate Browse, Update, or Report procedures for this file alias.

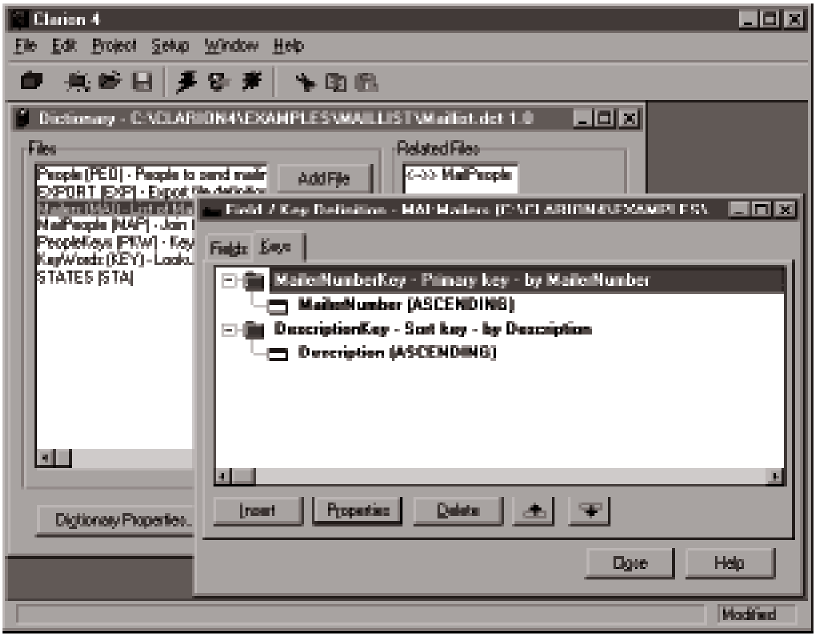
User Options

The text typed into this field is available to any Utility Templates that processes this file in the %FileUserOptions symbol. The Utility Templates determine the proper syntax for these user options.

Adding or Modifying Fields

Once you define a file, you may define its fields. Highlight the file name in the **Dictionary** dialog window then press the **Fields/Keys** button, or RIGHT-CLICK the file name then choose **Fields/Keys** from the popup menu. If you highlight an alias, the Dictionary Editor automatically displays the fields in the original file. Any changes then modify the original file as well as the alias.

Tip: **You can configure the information displayed in the Field/Key Definition dialog. See *Configuring the Dictionary Editor* for more information.**


Clarion 4
File Edit Project Setup Window Help
Dictionary - C:\CLARION4\EXAMPLES\MAILLIST\Maillist.dct 1.0
Files
People [PEO] - People to send mail
EXPORT [EXP] - Export file definition
Mailers [MAI] - List of Ma
MailPeople [MAP] - Join
PeopleKeys [PKW] - Key
KeyWords [KEY] - Looku
STATES [STA]
Add File
Related Files
<->> MailPeople
Dictionary Properties...
Field / Key Definition - MAI:Mailers (C:\CLARION4\EXAMPLES\
Fields
Keys
MailerNumberKey - Primary key - by MailerNumber
MailerNumber (ASCENDING)
DescriptionKey - Sort key - by Description
Description (ASCENDING)
Insert
Properties
Delete
Close
Help
Modified

The **Field/Key Definition** dialog contains two tabs. The **Fields** (left) tab lists the fields. The **Keys** (right) tab lists the keys.

- ❏ To add a new field, select the **Fields** tab, then press the **Insert** button.
- ❏ To modify an existing field, select the field name then press the **Properties** button.
- ❏ To delete an existing field, select the field name then press the **Delete** button.
- ❏ To move a field within the Fields list, select the field name then press the ▲ or ▼ button. This reorders the field labels within the generated FILE structure.

Tip: **Use the ▲ and ▼ buttons to move fields into and out of a GROUP.**

When you press the **Insert** button or **Properties** button for a field, the Dictionary Editor opens the **Field Properties** dialog.

Defining Field Properties

The **Field Properties** dialog lets you set field related options and attributes.

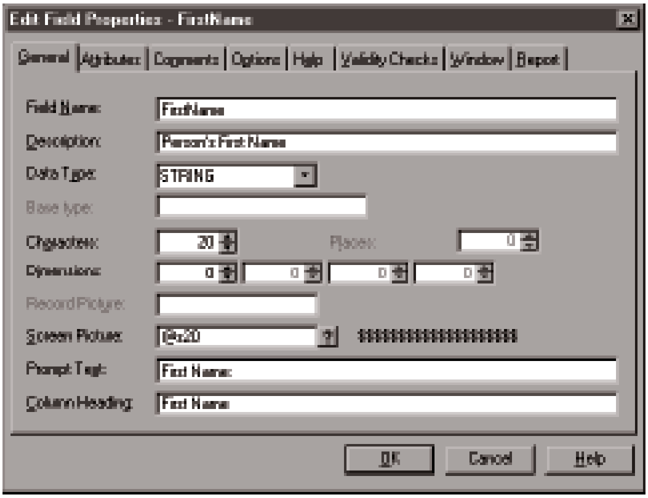
The Dictionary Editor lets you quickly add fields one after another. Each time you close the **New Field Properties** dialog for one field, the dialog reopens, ready for the next field. After completing the last field, press **Cancel** to return to the **Field/Key Definition** dialog.

Tip: **This dialog is *identical* to the dialog for defining and editing memory variables. All the Clarion language attributes applicable to a field in a file are also applicable to memory variables. However, there are a few additional attributes that are *only* applicable to memory variables. Controls which refer to attributes applicable only to memory variables are disabled when defining a field in a file.**

General

Field Name

Type a valid Clarion label. Valid field names may vary slightly according to the file driver.



Edit Field Properties - FirstName

Field Name: FirstName

Description: Person's First Name

Data Type: STRING

Base type:

Characters: 20

Places: 0

Dimensions: 0 0 0 0

Record Picture:

Screen Picture: @s20 ####################

Prompt Text: First Name:

Column Heading: First Name

Derived From

Press the ellipsis button (...) to select another (parent) field in the dictionary from which to copy all field attributes, except the field name. The parent field may be any other field in the data dictionary, including global data fields, field pool fields, or file fields.

Press the refresh button to reapply the attributes from the parent field. Use the **Freeze** check box below to prevent a refresh from the parent. See also the **Refresh Field, Refresh File, Refresh Dictionary,** and **Distribute Field** menu commands.

Description

Type a text description up to 40 characters. The description appears in the list in the **Field Properties** dialog. Also, see **Comments** below.

Data Type

Choose a data type from drop-down list. Clarion supports the following types which specify how the data is stored on disk and accessed in memory by the file driver. The types available vary according to the selected file driver. See *Database Drivers* in the *Programmer's Guide* for more information. See also *Choosing a Datatype* in this chapter.

STRING A fixed length character string, usually up to 65,520 characters in length, depending on the file system.

PICTURE Provides a “storage picture” for a String field. Picture is not a separate data type, but declares the field as a STRING whose length equals the size of its picture. Fill in the **Record Picture** field with the Storage Picture Token. See the *Language Reference* for a complete list of picture tokens, including examples.

CSTRING A character string terminated by a null, up to 65,520 characters in length. Corresponds to the C Language string data type, and the “ZString” field type in Btrieve.

PSTRING A character string with a leading length indicator, up to 255 characters in length. Corresponds to the Pascal Language string data type, and the “LString” field type in Btrieve.

BYTE Can contain an unsigned integer, from 0 to 255.

SHORT Can contain an integer, from -32,768 to 32,767.

USHORT Can contain an integer, from 0 to 65,535.

LONG Can contain an integer, from -2,147,483,648 to 2,147,483,647.

ULONG Can contain an integer, from 0 to 4,294,967,295.

DATE Corresponds to the “Date” field type in Btrieve.

TIME Corresponds to the “Time” field type in Btrieve.

SREAL Can contain a real number between 0 ± 1.175494535e^{-38} and 0 ± 3.40282347e^{+38}. Corresponds to the Intel 8087 short real format.

REAL Can contain a real number between 0 ± 2.225073858507201e^{-308} and 0 ± 1.79769313496231e^{+308}. Corresponds to the Intel 8087 long real format.

BFLOAT4 A real number between 0 ± 5.87747e^{-39} and 0 ± 1.70141e^{+38}. Corresponds to the four-byte Microsoft BASIC single precision format.

BFLOAT8 Can contain a real number between 0 ± 5.87747e^{-39} and 0 ± 1.7014118346e^{+38}. Corresponds to the eight-byte Microsoft BASIC double precision format.

DECIMAL Contains a real number between -999,999,999,999,999,999,999,999,999,999 and 9,999,999,999,999,999,999,999,999,999,999 in a packed decimal format. It offers 31 digits of precision. You must define at least one “place” to the left of the decimal point. The left-most byte contains the sign.

> **Tip:** **The Decimal type generally provides the best all around performance for mathematical calculations. The compiler optimizes the operation by multiplying values by powers of ten before processing; this greatly speeds up performance on systems without math coprocessors, at no cost in mathematical precision.**

PDECIMAL Contains a real number between -999,999,999,999,999,999,999,999,999,999 and 9,999,999,999,999,999,999,999,999,999,999 in a packed decimal format. It offers 31 digits of precision. You must define at least one “place” to the left of the decimal point. The right-most byte contains the sign, compatible with Btrieve and IBM/EPCDIC formats.

MEMO A variable length text field, up to 65,520 characters in length. MEMOs are allocated in 256 byte chunks.

To specify that a memo field may hold binary data, check the **Binary** box. This is dependent on the file driver. See *Database Drivers* in the *Programmer’s Guide* for more information.

BLOB Can contain variable length binary data larger than 64K. Similar to memos, BLOBs (Binary Large OBjects) are always variable length, with no length specified. They are database driver dependent, currently supported only by the TopSpeed driver.

GROUP A compound data structure that contains other fields with various data types. This corresponds to a C Language STRUCT.

Type the label for the group in the **Field Name** field. With each successive **New Field Properties** dialog, define the elements within the group.

TYPE Declares an instance of a user-defined data type. See *TYPE* in the *Language Reference*. The user-defined data type can be a GROUP, a QUEUE, or an object. Selecting TYPE enables the Base type field. See *Base type* below.

Base type

Specify the label of a user-defined datatype. The user-defined data type can be a GROUP, a QUEUE, or an object. See *TYPE* in the *Language Reference*. For example:

```
G1  GROUP,TYPE  !a user-defined data type
S1    STRING(10)
S2    STRING(10)
    END
```

Use the **Base type** field to specify the G1 label so the Application Generator generates something like the following:

```
MyTypeField   G1
```

Reference

To create a reference variable, check this box. A reference variable stores a reference to another variable or object, including but not limited to its memory address. This box is enabled only when defining memory variables. See the *Language Reference* for more information.

Characters

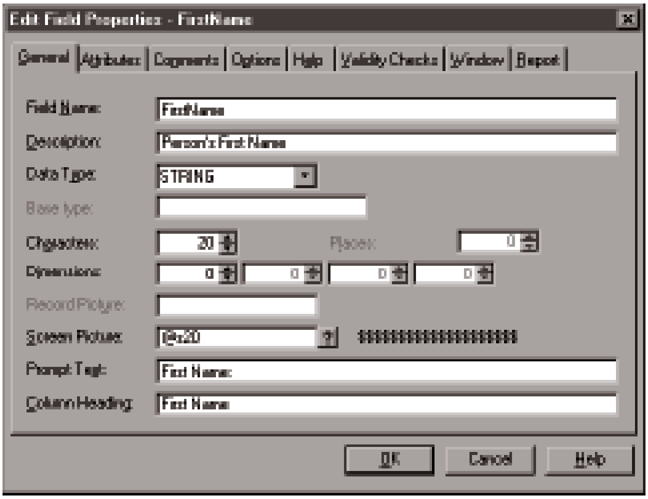
Specify the length of the field in bytes.

Places

For decimal data types, specify the number of places to the right of the decimal.

Dimensions

To declare the field as an array, and to specify the array dimensions, specify a size for up to four dimensions. Total array size may not exceed 65,520 bytes. See the *Language Reference* for more information on dimensioned variables and arrays.



Edit Field Properties - FirstName

Field Name: FirstName

Description: Person's First Name

Data Type: STRING

Base type:

Characters: 20 Places: 0

Dimensions: 0 0 0 0

Record Picture:

Screen Picture: @s20 ####################

Prompt Text: First Name:

Column Heading: First Name

Record Picture

To declare a STRING data type whose length equals the size of its picture, type a valid string picture here. You must choose *PICTURE* in the **Data Type** drop-list to enable this field. See the *Language Reference* for more information on pictures.

Screen Picture

To specify a display picture, regardless of data type, type a picture token here or press the ellipsis button to use the **Edit Picture String** dialog (see the *Picture Editor* chapter). When the Application Generator creates window and report controls for the field, this serves as the default display picture for the control. See the *Language Reference* for more information on pictures. See also *Controls and Their Properties—Common Control Attributes*.

CLICK the ▨ button next to the **Screen Picture** field to lock the screen picture even if the data type changes. CLICK the 🔒 button to unlock the screen picture.

Prompt Text

Specify the default text for the field's prompt. The Application Generator places this text in the PROMPT control associated with this field.

Tip: To specify that no prompt at all is associated with this field, clear the Prompt Text field, then press the Reset Controls button on the Window tab.

Column Heading

Type the default column title here. The Application Generator uses this for reports and list boxes.

Freeze

Check this box to prevent field attribute refresh from the parent field. Use the **Derived From** field above to set the parent field.

See also the **Refresh Field, Refresh File, Refresh Dictionary,** and **Distribute Field** menu commands.

Attributes

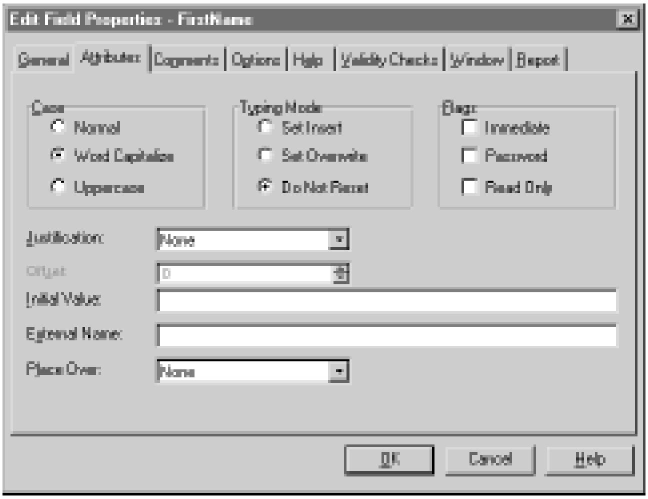
Case

Specify the default capitalization mode for the field’s entry controls.

Normal Specifies no enforced capitalization. The Application Generator adds nothing to the field’s entry control.

Word Capitals Specifies word capitalization. The Application Generator adds the CAP attribute to the field’s entry control.

Uppercase Specifies all capital letters. The Application Generator adds the UPR attribute to the field’s entry control.



Case
○ Normal
● Word Capitalize
○ Uppercase

Typing Mode
○ Set Insert
○ Set Overwrite
● Do Not Reset

Flags
□ Immediate
□ Password
□ Read Only

Justification: None

Offset: 0

Initial Value:

External Name:

Place Over: None

Typing Mode

Specify the default typing mode for the field’s entry controls.

Set Insert Specifies insert mode. The Application Generator adds the INS attribute to the field’s entry control. New characters are inserted at the cursor.

Set Overwrite Specifies overwrite mode. The Application Generator adds the OVR attribute to the field’s entry control. New characters overwrite characters at the cursor.

Do Not Reset Specifies no change to the typing mode. The Application Generator adds nothing to the field's entry control.

Flags

Specify other default attributes for the field's entry controls.

Immediate Check this box to specify immediate event notification for the field's controls. The Application Generator adds the IMM attribute to the field's entry control.

Password Check this box to specify the non-display attribute for the field's controls. The Application Generator adds the PASSWORD attribute to the field's entry control. Characters typed into the control appear as asterisks, plus standard Copy and Cut are disabled so users cannot copy a password and paste it into a text editor or another control.

Read only Check this box to specify the display only attribute for the field's controls. The Application Generator adds the READONLY attribute to the field's entry control.

Justification

Select a justification for the field's controls from the drop-down list. The Application Generator adds the LEFT, RIGHT, CENTER or DECIMAL attribute to the field's entry control. LEFT left justifies the leftmost character. RIGHT right justifies the rightmost character. CENTER centers the center character. DECIMAL right justifies the decimal point to the offset specified immediately below.

Tip: For decimal justification specify an offset equal to 4 times the number of places to the right of the decimal point.

Offset

Specify an offset to the justification. If justification is left, offset moves the lefmost character back to the right. If justification is right, offset moves the rightmost character back to the left. If justification is decimal, offset moves the decimal point to the left, revealing fractional digits that would otherwise be hidden. If justification is center, offset moves the center character left for negative values and right for positive values. The Application Generator uses this setting as the parameter for the LEFT, RIGHT, CENTER or DECIMAL attribute of the field's entry control. The measurement unit is Dialog Units.

Initial Value

Specify an initial (default) value for the field.

Database Fields Specifying an initial value for a database field generates an *assignment statement*. For example:

```
MyField = MyVariable + TODAY()
```

Therefore you may type any valid Clarion expression. The dialog does not validate your entry; however, at compile time the compiler identifies an invalid expression.

Memory Variables

Specifying an initial value for a global, module, or local memory variable generates a *data declaration statement*. For example:

```
MyString  STRING('InitialString')
```

or

```
MyNumber  LONG(100)
```

Type a string constant for a string field or a number for a numeric field. The Application Generator automatically generates surrounding quotes for string fields.

> **Tip: Functions, variables, and expressions are valid initial values for database fields, but not for memory variables (global, module, or local)! Use surrounding quotes to specify literal values for database fields; do not use quotes to specify literal values for memory variables.**

External Name

Specify an external name for the field. This adds the NAME attribute to the field's declaration—see the *Language Reference* for more information.

The function of the NAME attribute varies with the file driver. See *Database Drivers* in the *Programmer's Guide* for more information on specific file drivers. Usually, the NAME attribute specifies the native file system name for the field when it is different than the Clarion field label.

Place Over

Select another field name from the drop-down list to allow the current field to redefine the other field's location in memory. This adds the OVER attribute to the field's declaration—see the *Language Reference* for more information.

Storage Class

Specify the storage class for memory variables. Choose from:

DEFAULT The Application Generator adds no attribute.The variable is allocated from stack memory, which means the variable is reallocated for each new instance of the procedure.

EXTERNAL - LOCAL
The Application Generator adds the EXTERNAL attribute which specifies the variable is defined in an external library and is allocated no memory by this program.

EXTERNAL - DLL
The Application Generator adds EXTERNAL and DLL (dll_mode) which specifies the variable is defined externally in a DLL. dll_mode is a switch indicating whether the DLL attribute is active or not. The DLL attribute is required for EXTERNAL variables in 32-bit applications.

STATIC The Application Generator adds the STATIC attribute, which specifies the variable is allocated static memory instead of stack memory. This makes the variable "persistent" from one instance of the procedure to the next.

THREAD The Application Generator adds the THREAD attribute which specifies the variable is allocated static memory separately for each execution thread in the program. Thus the value of the variable depends on which thread is executing.

See the *Language Reference* for more information on these memory allocation attributes.

Comments

Comments Type up to 1000 characters of comments or text description. The description is solely for your convenience and has no effect on the application. It is useful when other programmers take over your project or when you return to the project after an absence.

Options

Do Not Populate This Field
To cause Clarion's Wizards to omit this field from browses, forms, and reports, check this box.

Population Order
Specify where Clarion's Wizards place this field on browses, forms, and reports. Chose from:

Normal Populates fields in the same order they appear in the Data Dictionary.

First Places these fields before all **Normal** and **Last** fields.

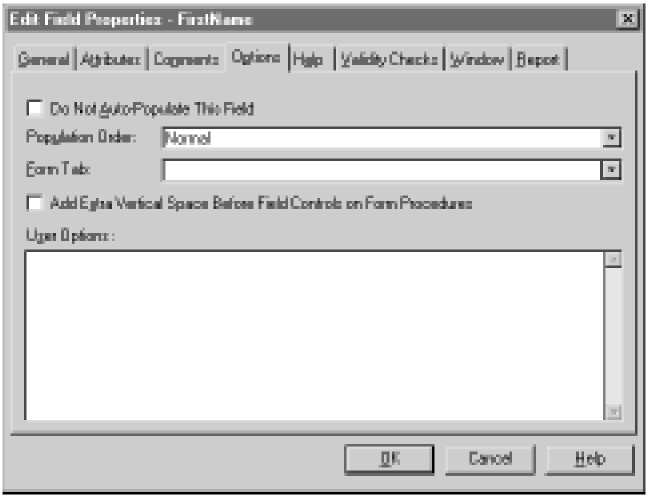
Last Places these fields after all **First** and **Normal** fields.

Form Tab

Specify the property sheet tab on which Clarion’s Wizards will populate this field for form (update) procedures. You may type a new tab name, or select from the drop-down list. Each new tab name is added to the drop-down list. The default tab name is “General.”

Add Extra Vertical Space Before Field Controls on Form Procedures

Check this box to cause Clarion’s Wizards to add extra vertical space before this field on form (update) procedures. Default vertical spacing is 4 dialog units. Checking this box doubles the default spacing.



☐ Do Not Auto-Populate This Field

Population Order: Normal

Form Tab:

☐ Add Extra Vertical Space Before Field Controls on Form Procedures

User Options:

User Options The text typed into this field is available to any Utility Templates that process this file in the %FieldUserOptions symbol. The individual Utility Templates determine the proper syntax for these user options. See also TOOLOPTIONS and %FieldToolOptions in the *Programmer’s Guide*.

Help

Help ID

Specify a help topic here. The Application Generator adds the HLP attribute to the field’s entry control.

Message

Specify a status bar message for controls referencing the field. When the control has focus, the text appears on the status bar, provided the application has one. The Application Generator adds the MSG attribute to the field’s entry control.

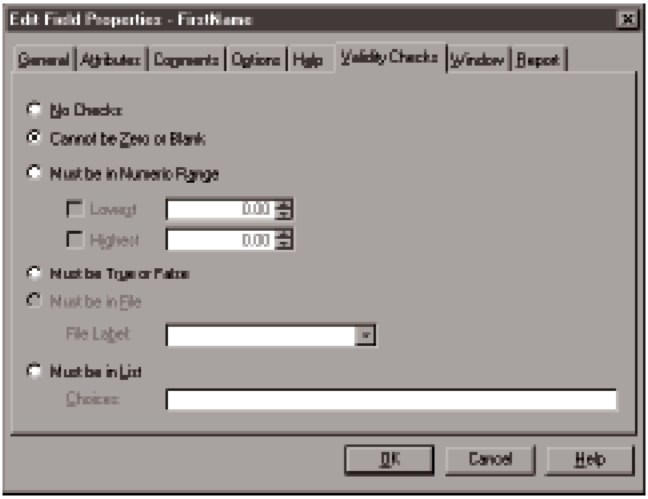
Tool Tip

Specify a popup message for controls referencing the field.

When the mouse cursor is idle over the field, the text appears immediately below the cursor in a popup box. The Application Generator adds the TIP attribute to the field’s entry control.

Validity Checks

To validate a user entry to this field, select the **Validity Checks** tab in the **Field Properties** dialog, then choose a validation option by CLICKING on one of the radio buttons.



○ No Checks
● Cannot be Zero or Blank
○ Must be in Numeric Range
□ Lowest 0.00
□ Highest 0.00
○ Must be True or False
○ Must be in File
File Label
○ Must be in List
Choices

The Application Generator uses this information to generate data validation code. At runtime, when the user tabs off the field and shifts focus to another control, or presses **OK** on the data entry dialog, the application sounds a warning beep and sets focus back to the control if the data entered is not valid.

Tip: **When setting a validity check, provide the user with a helpful status bar message. For example, if you specify that a numeric field must hold a value between 1 and 50, place a message such as “Type a number between 1 and 50” in the Message field (see Help Tab above).**

The validity checks constrain data entry to the criteria you select:

No Checks
Specifies no validation. This is the default.

Note: **Some validation can still occur, depending on the control you use to display the data, and the attributes of the control. For example, an ENTRY control with the REQ attribute automatically enforces a non-blank entry.**

Choices Type the choices to display in the format “Choice1|Choice2|Choice3.” Separate the choices with a pipe (|) character (usually SHIFT+\). The Application Generator adds the FROM attribute to SPIN, LIST, and COMBO controls, or adds text to RADIO controls (see the *Language Reference*).

Values Type the value to assign when the end user selects the corresponding choice. Type the values in the format “value1|value2|value3.” Separate the values with a pipe (|) character (usually SHIFT+\). The Application Generator adds the VALUE attribute to RADIO controls (see the *Language Reference*).

Cannot be Zero or Blank

Specifies a required field. The Application Generator adds the REQ attribute (see the *Language Reference*).

The REQ attribute behaves differently for tabbed dialogs than for single page dialogs. Because the user has the option of never selecting secondary tabs (pages), special steps are required to enforce entry of required fields that reside on secondary tabs:

Put all required fields on the first tab and add the REQ attribute to the tab and to the required entry fields; or

Make a “Wizard” to make sure the user process all tabs; or

At the beginning or end of the procedure, embed code that selects all tabs with required fields and add the REQ attribute to the required entry fields and to their parent tabs.

Must be in Numeric Range

Specifies the entry must fall within a numeric range. You may specify a minimum value, a maximum value, or both. The Application Generator generates code to enforce the range you specify, and adds the RANGE attribute to SPIN controls (see the *Language Reference).*

Lowest Check this box to set a minimum value, then enter the value in the corresponding spin box. Clear the box to specify no minimum value.

Highest Check this box to set a maximum value, then enter the value in the corresponding spin box. Clear the box to specify no maximum value.

By entering only a lowest, or only a highest value, you can specify an open ended range.

Must be True or False

Specifies a Boolean entry (yes/no, true/false, off/on). The Data Dictionary Window Control defaults to CHECK.

True Value Type the value to assign when the end user checks the CHECK control. The Application Generator adds the VALUE attribute to the CHECK control (see the *Language Reference*).

False Value Type the value to assign when the end user clears the CHECK control. The Application Generator adds the VALUE attribute to the CHECK control (see the *Language Reference*).

Note: The Application Generator does not generate code to enforce true/false entries because, in Clarion, all entries evaluate as either true or false. This selection affects the default window control in the Data Dictionary and applies the VALUE attribute if the control is a CHECK.

Must be in File

Specifies the value must match a field in a file. This option is enabled only if you previously related another file or files. See *Adding or Modifying Relationships* in this chapter.

File Label Select the lookup file from the list of related files. The Application Generator generates code to make sure the entered value is in the selected lookup file.

Tip: Use the FileDrop or FileDropCombo control template in your application to provide a same window pick list for the end user, or use the Actions tab for an ENTRY control to provide a separate window pick list.

Must be in List

Specifies the value must match one of the specified choices. The choices are displayed with a SPIN, LIST, COMBO, or RADIO control.

Choices Type the choices to display in the format "Choice1|Choice2|Choice3." Separate the choices with a pipe (|) character (usually SHIFT+\). The Application Generator adds the FROM attribute to SPIN, LIST, and COMBO controls, or adds text to RADIO controls (see the *Language Reference*).

Values Type the value to assign when the end user selects the corresponding choice. Type the values in the format "value1|value2|value3." Separate the values with a pipe (|) character (usually SHIFT+\). The Application Generator adds the VALUE attribute to RADIO controls (see the *Language Reference*).

Note: **The Application Generator does not generate code to enforce Must be in List entries. This selection affects the default window control in the Data Dictionary and applies the FROM and VALUE attributes.**

Window

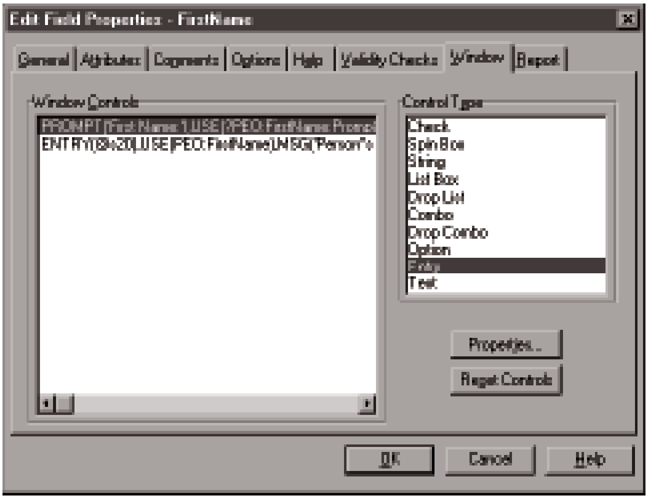
Use the **Window** tab in the **Field Properties** dialog to specify how your application presents a particular field to the user in the Windows environment. Remember, this specification is the *default* presentation method. By defining it here in the data dictionary, you establish a standard method of presenting the field which is used every time you place the field on a Clarion window. This means *you need only design your presentation method once*, no matter how many times your application makes use of this field, and no matter how many applications use this dictionary; but *you still retain the ability to modify this default presentation in any particular case*.

❑ To customize the default characteristics for prompts and entry controls for a field:

1. Select the **Window** tab in the **Field Properties** dialog.

Tip: **By setting the properties for the control here, you can save time later. Every application you generate from the dictionary, and every procedure in the application automatically formats the control according to the dictionary. If you don't format it here, and if the control requires custom formatting, you will have to custom format it for each use in a procedure and application later.**

2. In the **Control Type** list, highlight the type of control to use for the field. The **Window Controls** list reflects your choice.



Edit Field Properties - FirstName

Window Controls

PROMPT('First Name:'),USE(?PEO:FirstName:Prompt

ENTRY(@s20),USE(PEO:FirstName),MSG('Person's

Control Type

Check
Spin Box
String
List Box
Drop List
Combo
Drop Combo
Option
Entry
Text

3. In the **Window Controls** list, highlight a control (PROMPT, ENTRY, TEXT, SPIN, etc.), then press the **Properties** button. The respective control properties dialog opens. Use this dialog to set the control's position, size, color, font, text, mode, etc. See *Controls and Their Properties* for more information.
4. Optionally, press the **Reset Controls** button to change prompt and entry controls back to their default values.

> **Tip: To specify that no prompt at all is associated with this field, clear the Prompt Text field on the General tab, then press the Reset Controls button.**

Report

This tab works exactly like the **Window** tab. By defining your report control here in the data dictionary, you establish a standard method of displaying the field every time you place the field on a Clarion report.

Choosing a Datatype

Clarion provides a wide variety of datatypes to accomodate almost any programming need. This section provides some tips on choosing the correct datatype for your particular programming problem.

Dates and Times

LONG is generally the best data type for date and time values. This datatype lets you use Clarion standard Date and Time arithmetic, and provides a very efficient storage mechanism.

The DATE and TIME data types are useful for compatibility with Btrieve and SQL DATE and TIME datatypes, and they provide the most information to file utilities that rely on file header information. For Date arithmetic they are less efficient than LONG.

For xBase files, you generally use STRING data types for date storage, because STRING is the actual data storage that all the xBase file systems use. See the specific file driver's documentation in the User's Guide for more information on this issue, because your choice can be affected by whether the file already exists or your program needs to create it.

ZIP Codes

The DECIMAL data type is very good for zip codes because it's a packed decimal format—a 9 digit ZIP+4 in a DECIMAL is 5 bytes of storage while 9 digit zip in a STRING is 10 bytes. This kind of storage savings is a real consideration when you're setting up a large database. Since there is no math to perform on ZIP codes, storage is generally a larger consideration than performance.

Phone Numbers

The DECIMAL data type is also very good for non-international phone numbers, for the same reasons as ZIP codes. Since you're dealing only with phone numbers in your own country, you should be able to define the exact number of digits and format to display. For U.S. numbers, you can store the area code separately from the phone number—use a SHORT for the area code (3 digits in 2 bytes) and a LONG for the phone number (7 digits in 4 bytes)—and achieve the same storage as a single DECIMAL(10,0) (10 digits in 6 bytes).

If your program must deal with international phone numbers, the best data type is a STRING, because the most common method of indicating the country code is with a plus sign (+). For example, +44 (0)800 555 1212 indicates country code 44 (the United Kingdom). You should make the STRING at least 20 characters, since the number of digits in the number can vary from country to country, and even within separate sections of the same country.

"Customer" Numbers

"Customer" Number is defined for this discussion as any internal number in your program used primarily as the linking field between Parent and Child files.

LONG is the most common data type used for internal numbering for linking purposes. It is very efficient for both storage (4 bytes) and execution (it is one of the base data types used internally by the Clarion libraries see *Base Types* in the *Language Reference*). Any KEY based on a single LONG field is very efficient and small on the disk, since it requires fewer key node splits than a KEY based on a longer STRING (like the "customer" name).

Money

The best data type for any field that will store money values is DECIMAL. Using DECIMAL provides the most efficient storage, since it is a packed-decimal format. It also provides Binary Coded Decimal (BCD) math functionality, which means that calculations are executed in Base 10 instead of Binary (as it would if you use REAL). Using BCD math eliminates the rounding and significant digit problems that you can encounter when you use any of the floating point data types (REAL, SREAL, BFLOAT4, BFLOAT8). See *BCD Operations and Functions* in the *Language Reference* for more on BCD math.

Adding or Modifying Keys

Keys and indexes specify sort orders for a data file. Add and change keys and indexes for your database in the **Field/Key Definition** dialog. The Data Dictionary generates the correct FILE structure declaration based on the choices you specify here.

Keys

A key may reside within the data file or as a separate file, depending on the file system. See *Database Drivers* in the *Programmer's Guide* for more information.

Keys are automatically updated whenever records are added, changed, or deleted.

Static Indexes and Run-time Indexes

Indexes usually exist as separate files. Remember that a separate file handle is necessary for each separate key or index file. *Index files are not updated automatically*. The BUILD statement updates an index. See the *Language Reference* for more information on BUILD. BUILD behavior depends on the file system. See *Database Drivers* in the *Programmer's Guide* for more information.

A Static Index's component fields are specified in the Data Dictionary. The BUILD statement for a Static Index always rebuilds the index based on the component fields specified in the Data Dictionary.

A Run-time Index lets you declare an index file without specifying the key component fields in the Data Dictionary. The application must define the key component fields at run-time, as the second parameter of the BUILD statement. The application may rebuild the same index file with different key component fields!

Creating a key or index

1. Select a file from the list on the **Files** side of the **Dictionary** dialog and press the **Fields/Keys** button.
2. In the **Field / Key Definition** dialog, select the **Keys** tab to change focus to the **Keys** list.
3. Highlight a key (if one exists), then press the **Insert** button.

   This opens the **New Key Properties** dialog.
4. Type a valid Clarion label in the **Key Name** field.
5. Optionally type a **Description**.

The description displays in various dialog boxes, including the **File Definition** dialog.

6. In the **Type** group, select **Record Key** or **Static Index**.

   You may also select **Runtime Index**. If you do, you cannot specify component fields here in the Data Dictionary. Instead, you must specify the component fields with the BUILD statement.

7. Select the **Attributes** tab and check all boxes that are appropriate for the key.

   The *Setting Key Properties* section, below, describes the options in this dialog.

   Tip: If you imported the file definition based on existing data, these attributes are already set, and you should not change them.

8. Optionally type a valid file name in the **External Name** field, if the file system needs one.

   Clarion automatically adds the proper file extension.

9. Select the **Fields** tab, then press the **Insert** button.

   This opens **Insert Key Component** dialog. Key components are the fields used to index the data file. The component fields also determine the sort sequence of the key.

10. DOUBLE-CLICK a field in the list; this transfers its name to the **Fields** tab, which indicates the field is a component of the new key.

11. Repeat steps 9 and 10 for additional key component fields.

12. Press **OK** to close the **New Key Properties** dialog.

    The **New Key Properties** dialog reopens, ready to accept additional keys.

13. Repeat steps 4 through 12 to create additional keys for this file.

14. When you are finished adding keys, press **Cancel** to close the **New Key Properties** dialog and return to the **Field / Key Defintion** dialog.

At the end of the process, your keys appear on the **Keys** tab, with their field components arranged in hierarchical order.

Modifying a key or index

To modify a key or index, select it in the **Field/Key Definition** dialog, then press the **Properties** button. This opens the **Edit Key Properties** dialog. If you selected a key component, the **Fields** tab is on top. If you selected the key, the **General** tab is on top. The *Setting Key Properties* section, describes the options in this dialog.

Setting Key Properties

The following tabs and fields appear in the **New Key Properties** and **Edit Key Properties** dialogs. They set the attributes for the key.

General

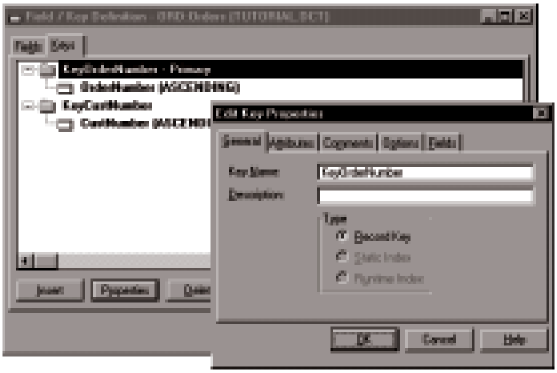
Key Name

Type a valid Clarion label in this field.

> **Tip:** **You cannot give a key the same name as one of the fields within the RECORD. One common convention is to use the field name plus the word "key," as in *LastNameKey*.**

Description

Type up to 40 characters in this field. *The description appears on Wizard generated tabs* and in dialogs such as the **File Definition** dialog. If you anticipate using many keys for your application, we recommend providing brief meaningful descriptions.



Key Name: KeyOrderNumber

Description:

Type

Specify a key or index. Choose from:

Record Key Keys are *automatically* updated whenever records are added, changed, or deleted. The Application Generator adds the KEY statement to the FILE declaration.

Static Index Static Indexes are *not* automatically updated, but require a BUILD statement. Static Index component fields are specified in the Data Dictionary. The Application Generator adds the INDEX statement to the FILE declaration.

Runtime Index Runtime Indexes are *not* automatically updated, but require a BUILD statement that specifies its component fields. Runtime Index component fields are *not* specified in the Data Dictionary. The Application Generator adds the INDEX statement to the FILE declaration.

> **Tip: The Static Index and Runtime Index options are *disabled* when the Require Unique Value check box is marked on the Attributes tab, because indexes always allow duplicates.**

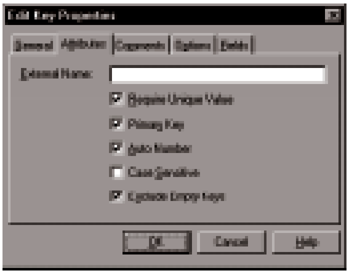
Attributes

External Name
Optionally, type a valid file name in this field. Clarion automatically adds the proper extension. The Application Generator adds the NAME attribute to the KEY or INDEX statement. Some file systems require an External Name. See *Database Drivers* in the *Programmer's Guide* for more information.

Require Unique Value
To disallow duplicate key values, check this box. This option is valid only for Record Keys and is disabled for indexes. The Application Generator adds the NO DUP attribute to the KEY statement.

Primary Key
To establish the current key as the Primary key, check this box. The Application Generator adds the PRIMARY attribute to the KEY statement. This may be required for certain file drivers. See *Database Drivers* in the *Programmer's Guide* for more information.

Auto Number
To tell the Clarion templates to generate code to manage record sequence numbers, check this box.



External Name:

☑ Require Unique Value

☑ Primary Key

☑ Auto Number

☐ Case Sensitive

☑ Exclude Empty Keys

Case Sensitive
To sort according to case, check this box. When creating or updating the key, capital letters precede lower case letters, as per their positions in the ASCII table. The Application Generator omits the NOCASE attribute from the KEY statement.

Exclude Empty Keys

To exclude records with a null or zero value in the key component fields from the key file, check this box. The Application Generator adds the OPT attribute to the KEY or INDEX statement.

> **Note: The primary key must be unique and must exclude nulls. Checking the primary key option has the same effect as checking both Require Unique Value and Exclude Empty Keys.**

Comments

Optionally select the **Comments** tab to enter up to 1000 characters of description.

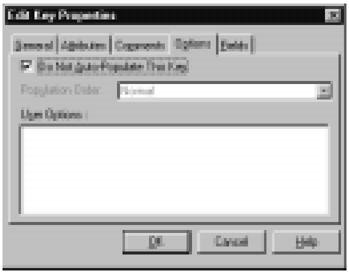
Options

Do Not Auto-Populate This Key

To cause Clarion's Wizards not to generate browses, forms, or reports based on this key, check this box.

Population Order

To specify where Clarion's Wizards will place this field on browses, forms, and reports, use the **Population Order** drop-down list. **Normal** populates fields in the same order they appear in the Data Dictionary. All **First** fields are placed before all **Normal** and **Last** fields. All **Last** fields are placed after all **First** and **Normal** fields.



Edit Key Properties

☑ Do Not Auto-Populate This Key

Population Order: Normal

User Options

To pass information to any Utility Templates that process this key, type the information here. The text typed into this field is available to any Utility Templates that process this key in the %KeyUserOptions symbol. The Utility Templates determine the proper syntax for these user options. See also TOOLOPTIONS and %KeyToolOptions in the *Programmer's Guide*.

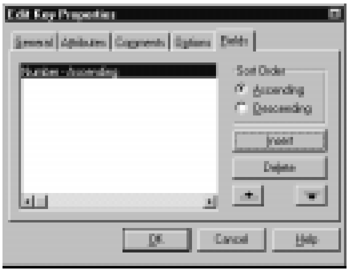
Key Component Fields

Specify the components of the key or index (the sort field or fields) with the **Fields** tab of the **Edit Key Properties** dialog. Key components are the fields used to index the data file. These fields determine the sort sequence of the key.

You may specify more than one field for a key and the fields may have different data types. You may also specify different sort directions—one field ascending, one field descending—when defining a key on multiple fields; however, mixing sort orders is file driver dependent. See *Database Drivers* in the *Programmer's Guide* for more information.

Key Fields List

To add fields, or components, to your key, press the **Insert** button. The **Insert Key Components** list then shows you the available fields. DOUBLE-CLICK on the name of a field in the list to add it to the **Key Fields List**.



Number - Ascending

Sort Order
● Ascending
○ Descending

Sort Order

To specify the sort sequence of your key component, choose either the **Ascending** or **Descending** radio button.

Component Order

You can change the order of the components of a key. To move a component up in the order, select it in the **Key Fields List**, then press the ▲ button. To move a component down in the order, select it in the **Key Fields List**, then press the ▼ button.

Adding or Modifying Relationships

Define relationships between files in the **New (or Edit) Relationship Properties** dialog. The relationships appear in the **Related Files** list on the right side of the **Dictionary** dialog for the currently selected file.

Creating a relationship

1. Select a file from the **Files** list on the left side of the **Dictionary** dialog.
2. Press the **Add Relation** button.

   This opens the **New Relationship Properties** dialog.
3. Select the relationship **Type** from the drop-down list.

   You may choose between a One-to-Many (**1:Many**) relationship or a Many-to-One relationship (**Many:1**). The 1:Many relationship defines a situation where *one* record in a file relates to *many* records in *another* file. For example, the Customer file contains only one record for customer Katy, but the Order file may contain many records for customer Katy, because Katy is a good customer that has ordered many items.

   In the above example, it doesn't matter which file you start with. If you selected the Customer file first, the type of relationship is **1:Many**, but if you selected the Order file first, you would specify a **Many:1** relationship.

Tip: **The label for the group box immediately below will change to Child or Parent, depending on your choice.**

4. Select the **Related File** from the drop-down list.

   The records in the two files, have one thing in common that relates them: the customer number. For example, the customer number for Katy might be 629, so the customer number for Katy's orders will also be 629. Thus the customer number is the "Key" to this file relationship.
5. Select the **Primary Key** or **Foreign Key** for the first file from the drop-down list at the top right of the dialog.

   Clarion automatically changes the label for the drop-down list (either **Primary Key** or **Foreign Key**) according to the relationship type.

   A **Primary Key** is *always* unique within the file for which it is primary. In our example there is exactly one customer number 629 in the customer file. So customer number is the **Primary Key** for the Customer file.

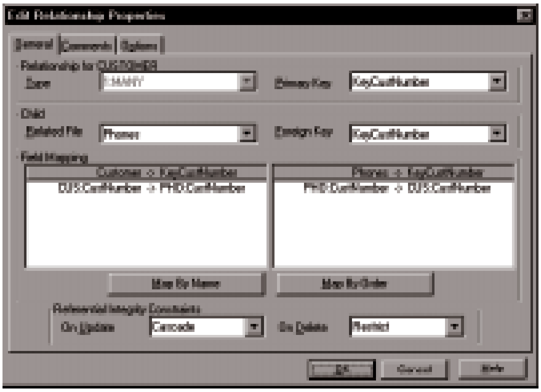
   A **Foreign Key** need not be unique, and it should match the primary key in another file. In our example, there is only one customer number 629 in our Customer file. However, the customer number 629 appears several times in the Order file. So customer number is the **Foreign Key** for the Order file.
6. Select the **Primary Key** or **Foreign Key** for the related file, from the drop-down list immediately below the first drop-down list.

For one-way or “lookup only” relationships, there is no Foreign Key. For example, the Customer file may contain a state code that is *not* a key component for the customer file. However, this same state code may be the primary key for the States file.

7. Press the **Map by Name** button to establish the link between the two keys by matching field names within the two keys.

   The **Field Mapping** lists show the actual links established between the two files.

   This mapping step is required because the keys in the two files are not always defined exactly the same way. For example, the Key_CustNumber in the Customer file might consist of CustNumber and LastName, while the Key_CustNumber in the Order file might consist of CustNumber only. Mapping ensures that keys with multiple component fields are handled correctly.



Edit Relationship Properties

Relationship for CUSTOMER

Type: 1:MANY

Primary Key: KeyCustNumber

Child

Related File: Phones

Foreign Key: KeyCustNumber

Field Mapping

Customer <- KeyCustNumber

CUS:CustNumber -> PHO:CustNumber

Phones -> KeyCustNumber

PHO:CustNumber -> CUS:CustNumber

Referential Integrity Constraints

On Update: Cascade

On Delete: Restrict

8. Optionally set Referential Integrity Constraints by choosing from the **On Update** and **On Delete** drop-down lists in the **Referential Integrity Constraints** group box.

   See the *Setting Referential Integrity Constraints* for more information.

9. Press the **OK** button to add the new relationship to the Data Dictionary.

Setting Referential Integrity Constraints

By setting Referential Integrity (RI) constraints in the data dictionary, you tell the Application Generator and the templates how to generate executable code for handling linked field updates and deletions when working with related data files.

Referential Integrity requires that a foreign key always has a match in the primary key. This raises potential problems when the end user wishes to change the value of a primary key or a foreign key, or delete a parent record.

Any of these changes could cause a mismatch between the primary and foreign keys.

The **Relationship Properties** dialog lets you specify how the generated code handles situations where only one of several related records is updated or deleted.

No Action

Tells the Application Generator to generate *no* code to maintain referential integrity.

Restrict

Tells the Application Generator to prevent the user from deleting or changing an entry, if the value is used in a foreign key. For example, if the user attempts to change a primary key value, the generated code checks for a related record with the same key value. If it finds a match, it will not allow the change.

Cascade

Tells the Application Generator to update or delete the foreign key record. For example, if the user changes a primary key value, the generated code changes any matching values in the foreign key. If the user deletes a parent record, the code deletes the children too.

Clear

Tells the Application Generator to change the value in the foreign key to null or zero.

Tip: The templates provide support for Referential Integrity for as many levels of relationships as are defined in the Data Dictionary.

Managing Your Dictionary

The Dictionary Editor provides several features to help you better manage your data dictionaries.

- You can copy and paste both file and field definitions from one dictionary file to another.
- The Dictionary Editor offers custom setup options which, for example, allow you to define the default database driver.
- The Dictionary Editor offers version control which lets you set checkpoints prior to major revisions, then roll back to previous checkpoints if necessary.

Tip: You can configure the information displayed in the Dictionary dialog and the Dictionary Toolbox. See *Configuring the Dictionary Editor* for more information.

Copying And Pasting

You can use the Copy and Paste commands to copy a file or field definition from one dictionary to another. To do so:

1. Open a dictionary file.
2. Select a file from the **Files** list in the **Dictionary** dialog.
3. Choose **Edit ➤ Copy**, or press CTRL+C.
4. Open a second data dictionary to the **Dictionary** dialog.
5. Choose **Edit ➤ Paste**, or press CTRL+V.

After pasting, the **New File Properties** dialog appears. You can modify the file definition as you wish. After you press the **OK** button, the file appears in the **Dictionary** dialog for the second dictionary.

Copying and pasting fields from one file to another works similarly, except that you must have the **Field / Key Defintion** dialogs open, rather than the **Dictionary** dialog. The target file must support the field type being copied.

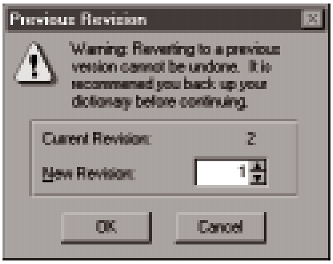
Tip: You can copy a Data Dictionary item, such as a file or a field, into the clipboard, then paste it into the Text Editor (and vice versa)!

Dictionary Revisions

A new dictionary automatically starts with version 1.0. You can see the version number/revision number on the caption bar of the **Dictionary** dialog. The **Dictionary Properties** dialog also displays the original creation date and time, and the last modified date and time.

You should increase the revision number whenever you make significant changes to a dictionary. From the **Dictionary** dialog, choose **Version ➤ Checkpoint**. A revision number (r#) is added to the caption bar. The revision number increases with each new "checkpoint."

To roll back to a previous revision, choose **Version ➤ Revert**. Choose the revision to revert to by selecting it with the spin control in the **Previous Revision** dialog.


Previous Revision

Warning: Reverting to a previous version cannot be undone. It is recommened you back up your dictionary before continuing.

Current Revision: 2

New Revision: 1

Configuring the Dictionary Editor

You can customize some of the default Dictionary Editor settings with the **Dictionary Options** dialog. These settings affect the appearance of the Dictionary Editor and the Dictionary Viewer. In addition, some settings affect the source code the Application Generator generates to manage data dictionary files and fields.

To customize the development environment's dictionary settings, choose **Setup ➤ Dictionary Options**.

File Options

Default Driver

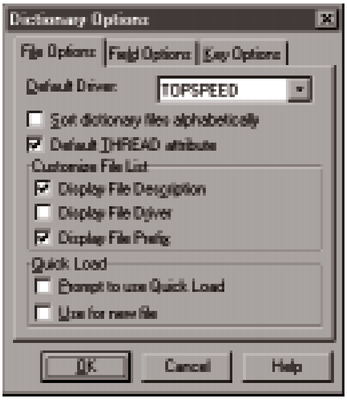
Select the default file system and database driver for all your dictionaries from the drop-down list. The default is TopSpeed—a solid, fast file system appropriate for large single user applications and small to medium sized multi-user applications. See *Database Drivers* in the *Programmer's Guide* for more information on drivers and file systems.

Sort dictionary files alphabetically

Check this box to show the files in alphabetical order in the Dictionary Editor.

Default THREAD Attribute

Check this box to add the THREAD attribute to each generated file definition. This provides a separate record buffer for each execution thread that uses the file so that multiple threads can access the file without conflict. Each application that uses the dictionary can override this setting for all files or for individual files.



Default Driver: TOPSPEED

☐ Sort dictionary files alphabetically

☑ Default THREAD attribute

Customize File List

☑ Display File Description

☐ Display File Driver

☑ Display File Prefix

Quick Load

☐ Prompt to use Quick Load

☐ Use for new file

Display File Description

Check this box to show file descriptions in the Dictionary Editor.

Display File Driver

Check this box to show file drivers in the Dictionary Editor.

Display File Prefix

Check this box to show the file prefix in the Dictionary Editor.

Prompt to use Quick Load

Check this box so the Dictionary Editor or Viewer prompts you to use Quick Load or not whenever you add a new file. See *Dictionary Editor—Quick Load* and see *Adding a File with Quick Load* in the*Getting Started* book.

Use for new file

Check this box to use Quick Load as the default method for adding new files to the dictionary. If you requested a prompt, this sets the prompt dialog default; if you requested no prompt, this sets the default new file action. See *Dictionary Editor—Quick Load* and see *Adding a File with Quick Load* in the*Getting Started* book.

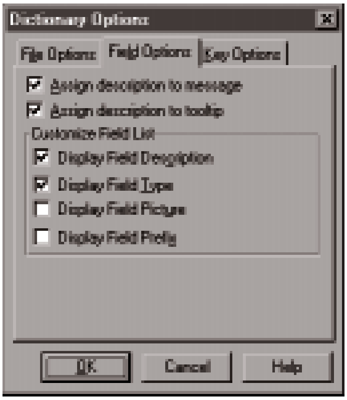
Field Options

Assign Description to Message

Check this box to display field descriptions on your application's status bar. You may override this default setting for individual fields in the dictionary and for the individual controls that display the fields. See *Dictionary Editor—Message* and see *MSG* in the *Language Reference*.

Assign Description to tooltip

Check this box to display field descriptions as your application's balloon help. You may override this default setting for individual fields in the dictionary, and for the individual controls that display the fields. See *Dictionary Editor—Tooltip* and see *TIP* in the *Language Reference*.



☑ Assign description to message
☑ Assign description to tooltip

Customize Field List

☑ Display Field Description
☑ Display Field Type
□ Display Field Picture
□ Display Field Prefix

Display Field Description

Check this box to show field descriptions in the Dictionary Editor.

Display Field Type
Check this box to show field types in the Dictionary Editor.

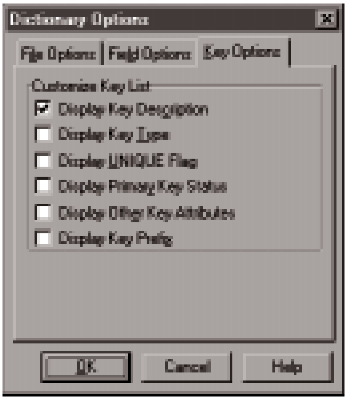
Display Field Picture
Check this box to show field pictures in the Dictionary Editor.

Display Field Prefix
Check this box to show field prefixes in the Dictionary Editor.

Key Options

Display Key Description
Check this box to show key descriptions in the Dictionary Editor.

Key Type
Check this box to show key types in the Dictionary Editor.



Customize Key List

☑ Display Key Description
□ Display Key Type
□ Display UNIQUE Flag
□ Display Primary Key Status
□ Display Other Key Attributes
□ Display Key Prefix

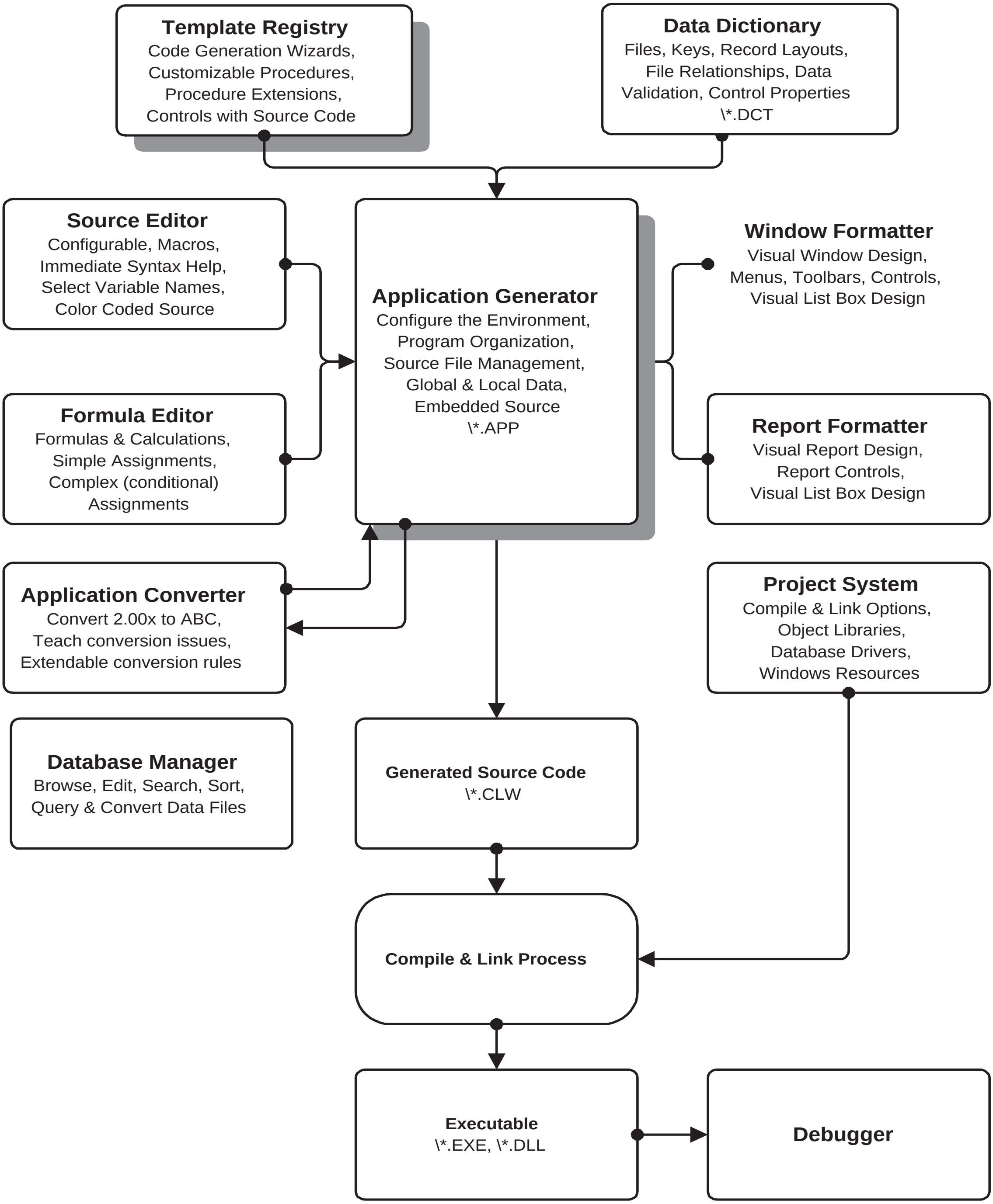
Display UNIQUE Flag
Check this box to identify unique keys in the Dictionary Editor.

Display Primary Key Status
Check this box to identify primary keys in the Dictionary Editor.

Display Other Key Attributes
Check this box to show other key attributes in the Dictionary Editor.

Display Key Prefix
Check this box to show key prefixes in the Dictionary Editor.

3 - APPLICATION GENERATOR



About This Chapter

When you use the Application Generator, you define procedures for the major tasks you want your application to do. Then you describe how the procedures accomplish the tasks, and how their windows, dialogs and reports appear to the end user. The Application Generator draws from the Template Registry, the Data Dictionary, and the information you provide, to write the source code for the application and its procedures.

This chapter describes:

- How to begin a new application by creating an application (.APP) file.
- How to add procedures to the application.
- How to fully customize your procedures.
- How to import and export procedures.
- How to configure the Application Generator.
- How templates “drive” the Application Generator.
- How to configure the Template Registry.

Creating the Application (.APP) File

The first step in creating a new application (after creating a Data Dictionary) is to create an application (.APP) file. The application file holds the procedure specifications, data declarations, and other properties you define for your application—it contains everything necessary to generate source code, then make an executable program.

> **Tip: You may want to create a new directory for each application you develop because whenever you open an application (.APP) file, Clarion uses the directory in which the file resides as the working directory.**

1. Optionally, in File Manager or Windows Explorer, choose **File ➤ Create Directory** or **File ➤ New ➤ Folder,** type a subdirectory or folder name then press **OK**.

   This creates a working directory for your application.

2. Start Clarion and choose **File ➤ New ➤ Application**.

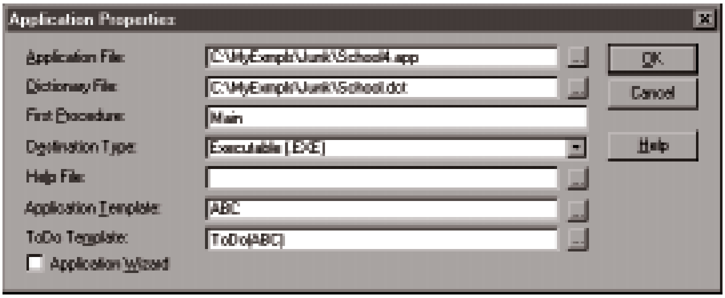
   This opens the **New** dialog.

3. Clear the **Use Quick Start** box by CLICKING on it.

   See *Quick Start Tutorial* in the *Getting Started* manual for more information on using **Quick Start**.

4. Use the **Save in** drop box to navigate to your application directory and the **File name** box to name the .app file, then press the **Save** button.

   This opens the **Application Properties** dialog where you define the basic files and properties for the application.



Application File: C:\MyExampl\Junk\School4.app

Dictionary File: C:\MyExampl\Junk\School.dct

First Procedure: Main

Destination Type: Executable (.EXE)

Help File:

Application Template: ABC

ToDo Template: ToDo(ABC)

□ Application Wizard

5. In the **Application File** field, optionally press the ellipsis (**...**) button to redefine the pathname for your .APP file.

6. Type a name for the .DCT file in the **Dictionary File** field, or press the ellipsis (**...**) button to select the dictionary file from the **Select Dictionary** dialog.

   See the previous chapter for information on creating your application's data dictionary.

> **Tip:** **The Application Generator does not require a data dictionary to generate an application if you *clear* the Require a dictionary box in the Application Options dialog. See *Configuring the Environment* for more information.**

7. In the **First Procedure** field, name application's "supervisor" procedure.

8. Choose the **Destination Type** from the drop-down list.

   This defines the target file for your application. Choose from *Executable (.EXE), Library (.LIB),* or *Dynamic Link Library (.DLL)*.

   Choosing *Dynamic Link Library (.DLL)* enables the **Export Procedure** prompt in the **Procedure Properties** dialog (see *Setting Procedure Properties*) and the **Export all file declarations** prompt in the **Global Properties** dialog (see *Template Overview—File Control Tab Options* the *Application Handbook*).

> **Tip:** **Using LIBs or DLLs to modularize and organize larger applications can provide substantial savings in development and maintenance costs: compiling and linking only a portion of a large application saves development time, and calling a set of common functions from a single source means maintaining only one set of code.**

   You may want to develop a portion of an application as an .EXE, then remake it as a .DLL when complete. See *Development and Deployment Strategies* for more information.

> **Tip:** **Setting the Project's Target Type is equivalent to setting the Application's Destination Type and vice versa. See *Project System* for more information.**

9. Type a name for the application's .HLP file in the **Help File** field, or use the ellipsis (**...**) button to select one from the **Select Help File** dialog.

   If you specify a help file in the current directory, the application looks for the help file in the current directory, then the system directory, then the system path. The full path is not stored.

   However, if you specify a help file in another directory, a full path is established and the application looks for the help file by the full pathname.

   You are responsible for creating a Windows Standard .HLP file that contains the context strings and keywords that you enter as HLP attributes for the application's various controls and dialogs. There are many third party products that help you do this.

10. Choose the **Application Template** type.

Accept the default (ABC), or press the ellipsis (**...**) button to select from another template set. The Application template controls source code generation.

11. Choose the **ToDo Template** type.

    Accept the default (ABC), or press the ellipsis (**...**) button to select from a third party template set. The ToDo template controls source code generation.

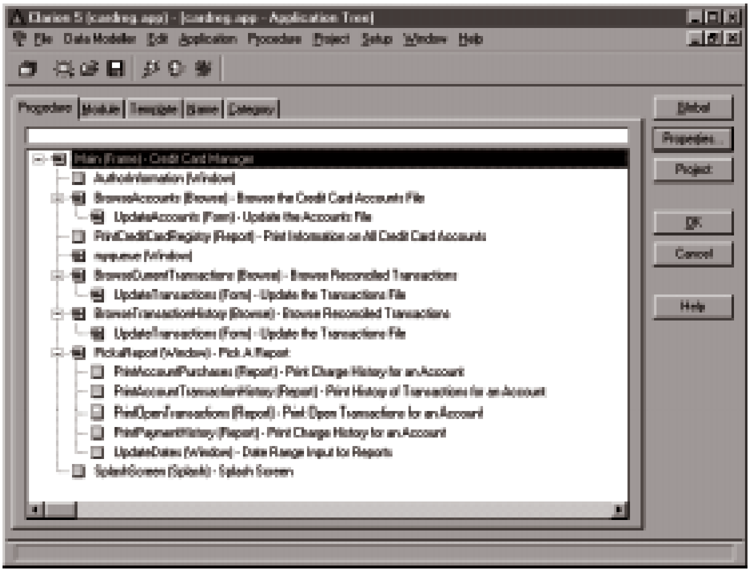
12. Clear the **Use Application Wizard** box by CLICKING on it.

    Checking this box causes the Application Generator to create an entire working application based on the data dictionary you selected. In this chapter, we will build an application without using the **Application Wizard**.

13. Press the **OK** button.

    Clarion creates the .APP file, then displays the **Application Tree** dialog for your new application.

    The **Application Tree** dialog provides five different views of your application. The **Procedure** view displays all application procedures in hierarchical order, nesting each procedure under its calling procedure.



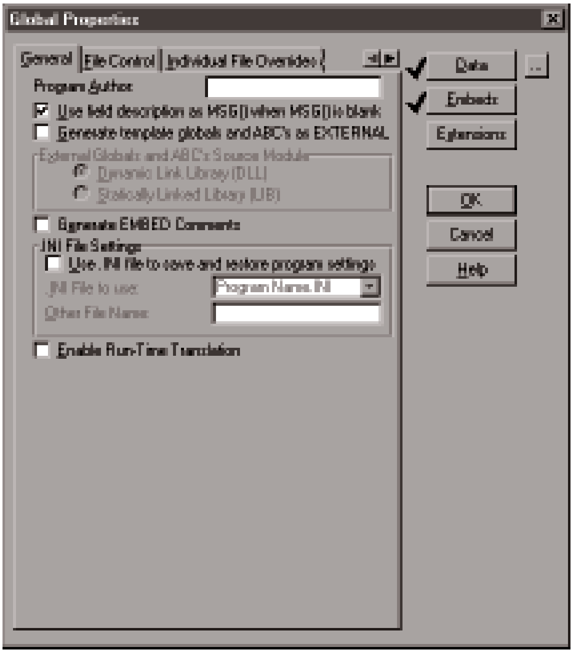
```
Main (Frame) - Credit Card Manager
  AuthorInformation (Window)
  BrowseAccounts (Browse) - Browse the Credit Card Accounts File
    UpdateAccounts (Form) - Update the Accounts File
  PrintCreditCardRegistry (Report) - Print Information on All Credit Card Accounts
  nqueue (Window)
  BrowseCurrentTransactions (Browse) - Browse Reconciled Transactions
    UpdateTransactions (Form) - Update the Transactions File
  BrowseTransactionHistory (Browse) - Browse Reconciled Transactions
    UpdateTransactions (Form) - Update the Transactions File
  PickaReport (Window) - Pick A Report
    PrintAccountPurchases (Report) - Print Charge History for an Account
    PrintAccountTransactionHistory (Report) - Print History of Transactions for an Account
    PrintOpenTransactions (Report) - Print Open Transactions for an Account
    PrintPaymentHistory (Report) - Print Charge History for an Account
    UpdateDates (Window) - Date Range Input for Reports
  SplashScreen (Splash) - Splash Screen
```

Global Application Settings

You can specify a number of settings that apply to your entire application, including file handling defaults, use of .INI files, global variables, and embedded source code. These "global" settings are done primarily through the **Global Properties** dialog.

Global Template Settings

The prompts on the **Global Properties** tabs are provided by the Application Template. See *Global Application Settings* in the *Application Handbook* for more information on these prompts.



Program Author

☑ Use field description as MSG() when MSG() is blank

□ Generate template globals and ABC's as EXTERNAL

External Globals and ABC's Source Module

● Dynamic Link Library (DLL)

○ Statically Linked Library (LIB)

□ Generate EMBED Comments

.INI File Settings

□ Use .INI file to save and restore program settings

.INI File to use: Program Name.INI

Other File Name:

□ Enable Run-Time Translation

The buttons in the **Global Properties** dialog (**Data, Embeds**, and **Extensions**) are provided by the Application Generator, and are described in this section.

Global Data and Variables

Global data must be declared before the CODE statement in your PROGRAM module (see the *Language Reference* for more information). There are several ways to accomplish this with the Clarion environment. You can declare global data in the data dictionary (see *Dictionary Editor—File Properties*); you can declare global data with the **Data** button in the **Global**

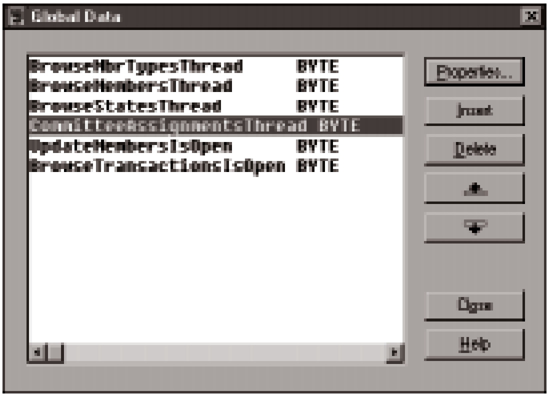
Properties dialog; and you can declare global data with the **Embeds** button in the **Global Properties** dialog (embed data declarations in a data section embed point—see *Embedded Source Code*).

data dictionary global data
: declares global data that can be shared by several applications. Because it is declared with the **Field Properties** dialog, you can specify controls and properties to apply to the data each time you populate them on your application's windows and reports.

Global Properties dialog **Data** button
: declares global data for a single application. Because it is declared with the **Field Properties** dialog, you can specify controls and properties to apply to the data each time you populate them on your application's windows and reports.

Global Properties dialog **Embeds** button
: declares global data for a single application with free form source code.

To access the **Global Properties** dialog, go to the **Application Tree** dialog and press the **Global** button. To add global variables, press the **Data** button in the **Global Properties** dialog.



```
BrowseMbrTypesThread      BYTE
BrowseMembersThread       BYTE
BrowseStatesThread        BYTE
CommitteeAssignmentsThread BYTE
UpdateMembersIsOpen       BYTE
BrowseTransactionsIsOpen  BYTE
```

To add a new variable to the list

1. Press the **Insert** button.
2. Fill in the **New Field Properties** dialog.

   The **New Field Properties** dialog is the same dialog used to add a field to the Data Dictionary. You can set all the characteristics of the variable, including the data type, length, label, etc. in this dialog. See *Dictionary Editor—Adding or Modifying Fields*.
3. Press the **OK** button to close the **New Field Properties** dialog.
4. Press the **Close** button to close the **Global Data** dialog.

To change the data type or label of a global variable

1. Highlight the variable in the **Global Data** dialog list.
2. Press the **Properties** button.
3. Make any changes necessary in the **Edit Field Properties** dialog then press **OK**.
4. Press the **Close** button to close the **Global Data** dialog.

To reposition a global variable

1. To move a variable up in the list, highlight it, then press the ⬆ button.
2. To move a variable down in the list, highlight it, then press the ⬇ button.

Global Embed Points

The global embed points are provided by the Application Template. See *Global Embed Points* in the *Application Handbook* for more information on these embed points.

To access these embed points, press the **Embeds** button in the **Global Properties** dialog. As with any embed point, you can write your own custom code, call a procedure, or use a code template. The Application Generator, when generating code, places your code or calls your procedure at the next source code line following the point you pick from the **Embedded Source** dialog. See *Embedded Source Code* for more information on adding embedded source code to your application.

Global Extensions

The **Extensions** button in the **Global Properties** dialog lets you add Extension templates to your application. Extension templates generate a variety of task oriented source code statements at one or more preset locations as needed to accomplish the extension task.

The ABC Templates include no application Extension templates. However, TopSpeed's Internet Connect uses an application extension template to make the application a hybrid Web/Windows application that can be run with an Internet Browser. See #EMPTYEMBED in the *Programmer's Guide* for another example of a small application Extension template.

Overview: Developing Your Application

Once the .APP file exists, you develop your application through a series of dialogs. When you create your application's menus and toolbars, they call procedures that you name. The Application Generator adds these "ToDo" procedures to the application tree. You define the functionality of the "ToDo" procedures by picking from a set of Procedure templates.

Remember, templates are code generation scripts that prompt you for information on how to customize the generated code. Use the Window and Report Formatters to supply information to the templates about how your application looks to the end user.

Following is an overview illustrating the tasks which you normally complete when building an application with the Application Generator. The tutorials in the *Getting Started* and *Learning Clarion* books provide a more detailed description.

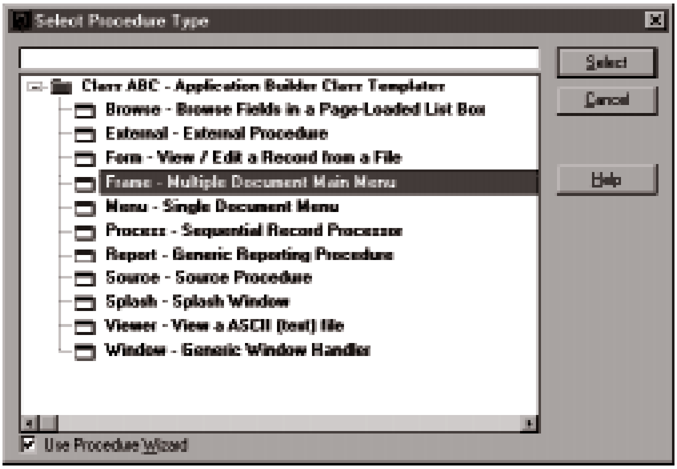
- ❏ Define the Main procedure.
    - Add menu commands and their "ToDo" procedures.
- ❏ Define the "ToDo" procedures.
    - Choose the appropriate template to generate each procedure.
    - Use the **Procedure Properties** dialog to identify procedure files.
    - Use the **Window Formatter** to design your windows.
    - Use the **Report Formatter** to design your reports.
    - Use the **Procedure Properties** dialog to add local variables as needed.
- ❏ Make the application (generate source code, compile, and link).
- ❏ Incrementally test the application (run it).

Define the Main Procedure.

In the dialog, highlight the *Main* "ToDo" procedure, then press the **Properties** button to access the **Select Procedure Type** dialog. This lists the Procedure templates available in the Template Registry.

Select the *Frame* procedure type for *Main* from the **Select Procedure Type** dialog then press the **Select** button.

The *Frame* procedure template is usually the best starting point for a typical application which employs different MDI child windows to present data in different views and forms. The *Frame* procedure template contains an MDI application frame, which already includes fully functional standard windows menus like **File, Edit** and **Help**. See *Procedure Templates* in the *Application Handbook* for more information.



Class ABC - Application Builder Class Templates
- Browse - Browse Fields in a Page-Loaded List Box
- External - External Procedure
- Form - View / Edit a Record from a File
- Frame - Multiple Document Main Menu
- Menu - Single Document Menu
- Process - Sequential Record Processor
- Report - Generic Reporting Procedure
- Source - Source Procedure
- Splash - Splash Window
- Viewer - View a ASCII (text) file
- Window - Generic Window Handler

☑ Use Procedure Wizard

After you make your selection, the **Procedure Properties** dialog appears. Each Procedure template contains defaults or starting points for such elements as the window, a basic menu structure, reports and more. These defaults are designed with real world uses in mind, such as update forms (a window that displays a single record) for updating a database record. When developing an application, you can customize these procedures to fit your needs.

Add Menu Commands and Their "ToDo" Procedures

In the **Procedure Properties** dialog, access the **Window Formatter** by pressing the **Window** button. When the **Window Formatter** appears, go directly to the **Menu Editor**: choose **Menu ➤ Menu Editor.** The **Menu Editor** dialog appears. See *Creating Menus and Toolbars* for details on editing the menu.

Typically, you add a menu item by pressing the **Item** button. Then, select the **Actions** tab to specify the procedure or program to execute when the end user chooses that menu item. Once you type in the procedure name, the Application Generator adds the procedure to the Application Tree as a "To Do."

When creating a Multiple Document Interface (MDI) application, check the **Initiate Thread** box when prompted.

Press the **Close** button to close the **Menu Editor**, saving your changes. Press **Exit!** to exit the **Window Formatter** and save your changes.

Define the "ToDo" Procedures

Select the first "ToDo" procedure in the Application Tree and press the **Properties** button. The "ToDo" items are the procedure or procedures you named with the **Menu Editor**.

Choose the Appropriate Template to Generate each Procedure

Select a Procedure type from the **Select Procedure Type** dialog, then press **Select**. At this point, you might choose, for example, a Browse template, which displays records in a list box. See *Procedure Templates* in the *Application Handbook* for more information on the available Procedure templates.

If you check the **Use Procedure Wizard** box, the **Browse Wizard, Form Wizard,** or **Report Wizard** prompts you for the information needed to complete your procedure (or procedures). See *Wizards and Utility Templates* in the *Application Handbook* for more information on the Procedure Wizards.

Choose the Files that the Procedure Uses

From the **Procedure Properties** dialog, press the **Files** button to open the **File Schematic Definition** dialog, then choose the files and keys the procedure uses. See the *Procedure Files* section below for detailed instructions. By adding files to the schematic, you allow the procedure to access them.

Add local variables

Press the **Data** button in the **Procedure Properties** dialog. The *Procedure Data* section, below, describes this process in detail. Basically, you declare each variable same way you define a field in a data file.

Use the Window Formatter to design your windows

In the **Procedure Properties** dialog, press the **Window** button. The **Window Formatter** displays a sample window. See *Window Formatter* for more information. Depending on the Procedure template you chose, the window may already contain some predefined controls.

Everything that appears in the window is a control, including buttons, list boxes, check boxes, spin boxes, data entry fields, etc. Select a control, then choose **Edit ➤ Properties** and **Edit ➤ Actions** to specify the appearance and behavior of the control.

Use control templates (**Populate ➤ Control Template**) to place “prefabricated” controls—fully functional controls with associated source code. See *Control Templates* in the *Application Handbook* for more information. For example, a BrowseBox control template generates a list control with associated source code that loads and scrolls the list.

Use dictionary fields (**Populate ➤ Field**) to place “some assembly required” controls, that is, entry controls that are automatically loaded with a data dictionary field or memory variable values.

Use simple controls (**Control** menu) to place “do it yourself” controls, that is, controls with no associated source code. See *Controls and Their Properties* for more information.

Make the Application

Press the ⚡ button on the toolbar to generate source code, compile, and link the application. The Application Generator automatically maintains the compile and link information for the application.

Test the Application

Press the 🏃 button on the toolbar or press the **Run** button in the **Compile Results** dialog.

After testing your first procedure, you can add more procedures, embed custom source code, and otherwise add functionality to your application.

Adding a Procedure to Your Application

A *procedure* is a series of Clarion language statements (source code) which perform a task. A *Procedure template* is an *interactive tool* that (with the help of Clarion's development environment) requests information from you, the developer, then generates a custom procedure for just the task you specify.

A Procedure as stored in a Clarion application (.app) file, is really a specification that the development environment uses to generate the procedure source code. The specification includes the Procedure template, your answers to its prompts, the WINDOW definition, the REPORT definition, local data declarations, embedded source code, etc.

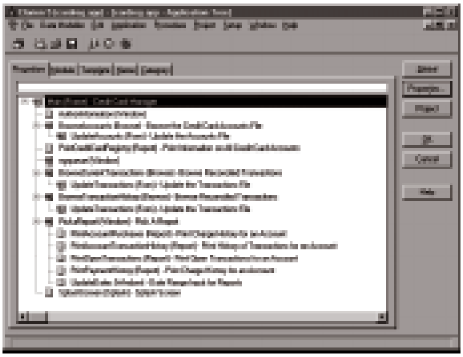
Your application's supervisor procedure is called "Main" by default. You can name this procedure anything you want, but this chapter refers to it as "Main." All other procedures branch from "Main"—one procedure can call another.

Application Tree

The hierarchical tree controls (or outline controls) in the **Application Tree** dialog illustrate how the procedures branch from "Main" and from each other. This provides a schematic diagram of your program's logical structure. See *Maintaining Your Application* for more information.

> **Tip: An icon means the procedure contains embedded source code.**

The Application Generator adds a procedure to the Application Tree whenever you press the INSERT key, or add a menu item, a toolbar command, or a code template that calls a procedure. Each new procedure is marked "To Do." When you "fill in" its functionality, the Application Generator replaces the "To Do" with your description.

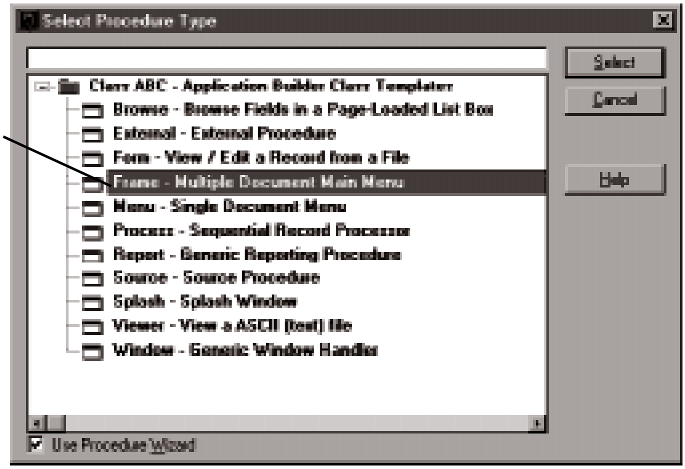


Defining the Procedure Type

Once you add a "ToDo" procedure to the Application Tree, the next step is to define its type from the choices available in the **Select Procedure Type** dialog. The choices available correspond directly to the Procedure templates in your template registry. See *Procedure Templates* in the *Application Handbook* for more information on the Procedure templates in this package.

❑ To open the **Select Procedure Type** dialog, select any "ToDo" procedure in the **Application Tree** dialog, then press the **Properties** button, or choose **Edit ➤ Properties**. You can also DOUBLE-CLICK on the "ToDo" procedure.

❑ To define the procedure type for your application's procedures, highlight a Procedure template from the **Select Procedure Type** dialog list, then press the **Select** button. You can also DOUBLE-CLICK on a procedure type. If you select a Browse, Form, or Report, and you check the **Use Procedure Wizard** box, the **Wizard** guides you through each step of the procedure properties definition. If you do not check the **Use Procedure Wizard** box, the **Procedure Properties** dialog appears.



Class ABC - Application Builder Class Templates
Browse - Browse Fields in a Page-Loaded List Box
External - External Procedure
Form - View / Edit a Record from a File
Frame - Multiple Document Main Menu
Menu - Single Document Menu
Process - Sequential Record Processor
Report - Generic Reporting Procedure
Source - Source Procedure
Splash - Splash Window
Viewer - View a ASCII (text) file
Window - Generic Window Handler

☑ Use Procedure Wizard

Select the procedure type from a list of available procedure templates.

If you must change the procedure *type* later, go to the Application Tree, highlight the procedure, then choose **Procedure ➤ Change Template Type**. The **Select Procedure Type** dialog appears so you can select another procedure type. If the new procedure type doesn't support some of the structures—such as menus—that you defined in the previous procedure type, you may "orphan" the previously defined structures. Therefore, be cautious when changing procedure type.

Setting Procedure Properties

After you choose the procedure type, you can define the procedure's properties—these properties include:

- a *description* of the procedure
- the procedure *prototype*
- the *module* containing the generated source code
- whether to *export* the procedure
- whether to *declare* the procedure *globally*
- *parameters* passed to the procedure[T]
- *return values* from the procedure[T]
- *INI file settings* used by the procedure[T]
- *files* accessed by the procedure
- the *WINDOW* displayed by the procedure, including its size, shape, appearance and functionality
- the *REPORT* generated by the procedure
- *data* items (fields and variables) used by the procedure
- *procedures* called by the procedure
- custom source code *embed*ded within the procedure
- *formulas* used by the procedure
- template generated *extensions* to the procedure

[T] These prompts are provided by the Procedure templates, therefore their presence or absence depends on the particular template that generates the procedure. See *Procedure Templates* in the *Application Handbook*.

You need not define every property for every procedure. In many cases, the default property definition is appropriate. When a default property is already established, a green check mark appears beside its command button. For example, the Browse procedure template contains a predefined window; therefore, a green check mark appears next to the **Window** button for procedures with this template.

For the properties you do define, you may use a **Wizard** and you may use the **Procedure Properties** dialog and its subordinate dialogs and formatters to set them. This section is primarily concerned with the **Procedure Properties** dialogs (see the *Application Handbook—Wizards and Utility Templates* for more information on Procedure Wizards).

Application Generator Properties (Prompts)

The properties discussed in this section are common to *all* **Procedure Properties** dialogs because they are managed by the Application Generator. You define these properties by completing the entry boxes and using the command buttons on the **Procedure Properties** dialogs.

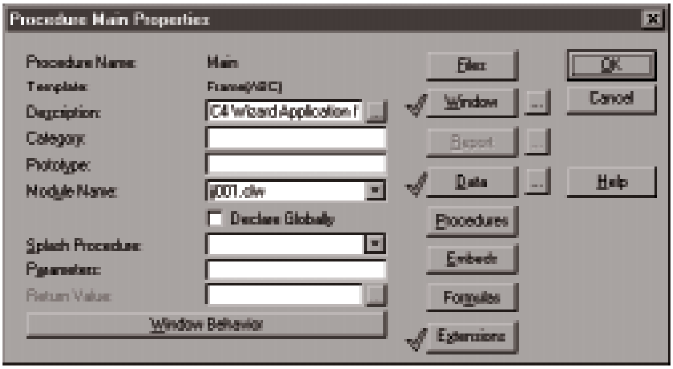
Template Properties (Prompts)

In addition to the Application Generator properties, the **Procedure Properties** dialog displays and manages the template prompts for the templates in the procedure. See *Procedure Templates* in the *Application Handbook* for more information on the template prompts.

To set the procedure properties

From the Application Tree, select a procedure then press the **Properties** button to access the **Procedure Properties** dialog. Alternatively, DOUBLE-CLICK on the procedure, or RIGHT-CLICK the procedure, then choose **Properties** from the popup menu.

This opens the **Procedure Properties** dialog which displays the following prompts.



Procedure Main Properties

Procedure Name: Main
Template: Frame(ABC)
Description: C4 Wizard Application f
Category:
Prototype:
Module Name: j001.clw
☐ Declare Globally
Splash Procedure:
Parameters:
Return Value:

Description

A short text description for the procedure, which appears next to the procedure name in the **Application Tree** dialog.

Press the ellipsis (...) button to edit a longer (up to 1000 characters) description.

Prototype

Optionally specify a custom procedure prototype which the Application Generator places in the MAP section. If you specify nothing, the Application Generator provides the correct prototype for the selected procedure template. See the *Procedure Prototyping* in the *Language Reference* and *Prototyping and Parameter Passing* below for more information. For example:

```
(SHORT ID,STRING Name)
```

Module Name

Specify which module (.CLW) file contains the *source code* for the procedure by selecting from the drop-down list. By default, the Application Generator names modules by taking the first five characters of the .APP file name, then adding a three digit number for each module. You may specify your own module names by choosing **Application ➤ Insert Module** from the menu.

Export procedure

Check this box to add the procedure to the application's automatically generated export (.EXP) file, so the procedure can be called by other executables. See *Module Definition Files* in the *Programmer's Guide* and see *Development and Deployment Strategies* for more information. This prompt is only available when you specify *Dynamic Link Library (.DLL)* as the **Destination Type** in the **Application Properties** dialog.

Declare Globally

Check this box to generate the procedure's prototype into the PROGRAM's MAP, rather than the MODULE's MAP. This makes the procedure callable from any other procedure, but it also forces a recompile of all program modules whenever you change the prototype.

Procedure Files

By default, file data (data stored in your application files) are available to any procedure within the entire application; however, you must tell the Application Generator which files are used by the procedure so it can provide source code for reading (and writing) the files.

❏ Press the **Files** button in the **Procedures Properties** dialog.

or

❏ RIGHT-CLICK the procedure in the Application Tree, then choose **Files** from the popup menu.

This opens the **File Schematic Definition** dialog.

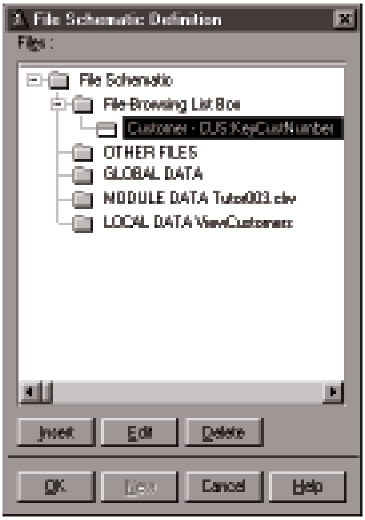
To add a file to the file schematic, select

a control template <To Do> item,
a control template file,
or **OTHER FILES**,

then press the **Insert** button. Next, choose a file from the **Insert File** dialog.

When you select a <To Do> itemfor the control template, you add a "primary" file from among all the files in your data dictionary. When you select an existing control template file, you add a "secondary" or "child" file from a list of related files only. Clarion's templates *automatically* generate all the code needed to open, read, and close both the primary and any secondary files. The templates also generate any code needed to update the primary file.

Select **OTHER FILES** when you want to access a file that is already open and positioned to the appropriate record, or when you want to hand code the file access. The templates automatically generate code to open (if not already open) and close **OTHER FILES**, but any other processing is up to you.



To delete an item from the file schematic, highlight it and press the **Delete** key.

Sorting

To specify the sort sequence of a file, select it in the **Files** list, then press the **Edit** button. Choose a key from the **Change Access Key** dialog.

Views and Joins

To specify an "inner join" or a custom join instead of the default "left outer join" for the generated JOIN structure, select the secondary (child) file, then press the **Edit** button to open the **File Relationship** dialog.

Dictionary

Choose this to generate a VIEW that reflects the file relationship as defined in the Data Dictionary. For example:

```
VIEW(Customer)
  PROJECT(CUS:LastName)
  PROJECT(CUS:FirstName)
  PROJECT(CUS:ID)
  JOIN(PH:CustomerIDKey,CUS:ID) !default JOIN
  END
END
```

Custom

Choose this to enter a custom JOIN expression for the generated VIEW. See *JOIN* in the *Language Reference* for more information. For example:

```
VIEW(Customer)
  PROJECT(CUS:LastName)
  PROJECT(CUS:FirstName)
  PROJECT(CUS:ID)
  JOIN(Phone,'PH:CustomerID,CUS:ID')  !custom JOIN
  END
END
```

Inner

Check this box so that only those primary file records with related secondary file records are retrieved. Inner joins are normally more efficient than outer joins. See *INNER* in the *Language Reference* for more information. For example:

```
VIEW(Customer)
  PROJECT(CUS:LastName)
  PROJECT(CUS:FirstName)
  PROJECT(CUS:ID)
  JOIN(Phone,'PH:CustomerID,CUS:ID'),INNER  !custom inner JOIN
  END
END
```

Procedure Windows

❑ Press the **Window** button in the **Procedures Properties** dialog.

or

❑ RIGHT-CLICK the procedure in the Application Tree, then choose **Window** from the popup menu.

The **Window Formatter** lets you visually design the size, shape, menus, controls and functionality for the window in this procedure. See the *Window Formatter* chapter for details on how to define your window.

❑ Press the **Window's** ellipsis (...) button in the **Procedures Properties** dialog to edit the source code that declares the window.

Tip: The ellipsis (...) button next to the Window button lets you edit the source code declaring the WINDOW structure. You may notice some non-language keywords such as #FREEZE, #ORIG, or #SEQ in this source code. Do not remove or change these keywords. The Application Generator uses them to manage source code generation, but does not include them in the generated source. The #LINK keyword ties an input control to its associated PROMPT control and can be safely deleted.

Procedure Reports

- ❑ Press the **Report** button in the **Procedures Properties** dialog.

  or

- ❑ RIGHT-CLICK the procedure in the Application Tree, then choose **Report** from the popup menu.

The **Report Formatter** lets you visually design the size, shape, content, layout, and functionality for the report in this procedure. See *Report Formatter* for more information.

- ❑ Press the **Report's** ellipsis (...) button in the **Procedures Properties** dialog to edit the source code that declares the report.

Tip: The ellipsis (...) button next to the Report button lets you edit the source code declaring the REPORT structure. You may notice some non-language keywords such as #LINK or #ORIG in this source code. Do not remove or change these keywords. The Application Generator uses them to manage source code generation, but does not include them in the generated source.

Procedure Data

Procedures may access several classes of data. These include file or field data (see *Procedure Files*), GLOBAL data (see *Global Variables*), MODULE data, and LOCAL data.

LOCAL data are defined in the data section of a procedure, and may only be accessed by the procedure that defines them. MODULE data are defined in the data section of a module. A module is simply a source file that may contain several procedures. Module data may be accessed by any procedures contained in the source file where the module data are defined. GLOBAL data may be accessed by any procedure in the entire application. See the *Language Reference* section on *Data Declarations and Memory Allocation* for more information.

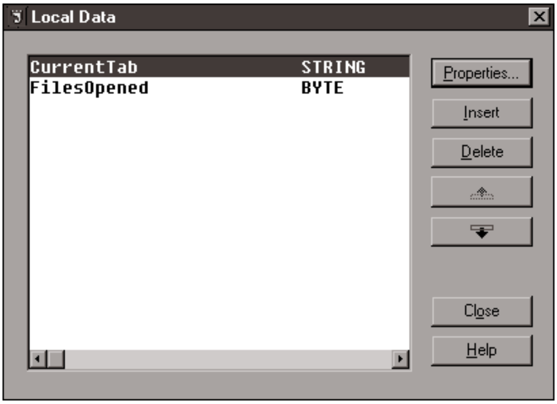
LOCAL Data

- ❑ Press the **Data** button in the **Procedures Properties** dialog.

  or

- ❑ RIGHT-CLICK the procedure in the Application Tree, then choose **Data** from the popup menu.

This opens the **Local Data** dialog. If any local variables already exist, they appear in the list.



```
CurrentTab                STRING
FilesOpened               BYTE
```

To define a new data item, press the **Insert** button.

This opens the **New Field Properties** dialog. Type in the variable name, choose the variable type, and set any additional attributes, including screen attributes. See *The DictionaryEditor—Adding or Modifying Fields* for more information on this dialog.

❏ Press the **Data's** ellipsis (...) button in the **Procedures Properties** dialog to edit the TXA code that declares the data. See the *Programmer's Guide* for more information on TXA format.

Tip: You may declare new data items here with normal Clarion language syntax. You do not need to supply the TXA code.

MODULE Data

❏ Press the **Data** button in the **Module Properties** dialog.

or

❏ RIGHT-CLICK the module in the Application Tree, then choose **Data** from the popup menu.

Defining Module data is exactly like defining Local data with one exception—you must select a module rather than a procedure. To define MODULE data (memory variables available to several procedures in a single source file):

1. From the **Application Tree** dialog, select the **Module** tab.
2. Highlight a module (folder), not a procedure.
3. Press the **Properties** button to display the **Module Properties** dialog, or RIGHT-CLICK then choose **Data** from the popup menu.
4. Press the **Data** button.

This opens the **Module Data** dialog.

5. Press the **Insert** button.

   This opens the **Field Properties** dialog. This is the *same* dialog used to set field properties in the data dictionary. For details, see *Dictionary Editor—Adding or Modifying Fields*.

❑ Press the **Data's** ellipsis (...) button in the **Module Properties** dialog to edit the TXA code that declares the data. See the *Programmer's Guide* for more information on TXA format.

Tip: You may declare new data items here with normal Clarion language syntax. You do not need to supply the TXA code.

Tip: Unlike LOCAL data, MODULE data is not automatically cleared when a procedure closes. You, the developer, must take care to initialize the MODULE data as required by your various procedures.

Calls to Other Procedures

❑ Press the **Procedures** button in the **Procedures Properties** dialog to identify procedures called within embedded source code.

Procedures may call other procedures. Procedure calls specified with template prompts are automatically added to the Application Tree; however, the Application Generator cannot "see" procedures called from embedded source code.

Identifying these embedded procedure calls is important to project management and to source code generation. When you use the **Procedures** button to identify procedures called within embedded source code, the Application Generator can properly display the procedures within the Application Tree hierarchy, *and* the Application Generator can generate the appropriate local MAP structure for the source module.

To identify procedures called from embedded source, press the **Procedures** button to open the **Called Procedures** dialog. This dialog lists all procedures in the application. Mark the called procedure by CLICKING on its name.

Embedded Source Code

❑ RIGHT-CLICK the procedure in the Application Tree, then choose **Embeds** from the popup menu (or press the **Embeds** button in the **Procedures Properties** dialog) to open the **Embedded Source** dialog to embed source code using alphabetically or logically ordered named embed points.

or

❑ RIGHT-CLICK the procedure in the Application Tree, then choose **Source** from the popup menu to open the Embeditor to embed source code within the context of surrounding generated code.

or

❑ RIGHT-CLICK on a control in the Window Formatter, then choose **Embeds** from the popup menu to access the embed points for a single control.

Clarion's templates let you add your own customized code to many predefined points inside the standard code that the templates generate. It's a very efficient way to achieve maximum code reusability and flexibility. The point at which your code is inserted is called an *Embed Point*. Embed points are available at all the standard events for the window, the window controls, and for many other logical positions within the generated code. The embed points are determined by the templates. You can even add your own embed points if needed. See #EMBED in the *Programmer's Guide* or the on-line Template Language help.

Embedding source code in a procedure lets you fully customize the procedure. The Application Generator saves the embedded source in the .app file and integrates it into the template generated source code each time you generate source code.

You can write your own embedded source code or use Code templates to generate the code for you. Once you embed source code in a procedure, the procedure is flagged with an "S" in the Application Tree.

In order to effectively embed code, you should understand the surrounding template generated code. See *Learning Clarion* and the *Application Handbook* for moreinformation on the Clarion and ABC Templates and the code they generate.

Several ways to Embed Source Code

Clarion provides several powerful methods for embedding source code. There are advantages to each of these methods as noted below:

- The Embeditor (choose **Source** from the popup menu) lets you see the embedded source code *within the context of the surrounding generated code* and gives you the full power of the Text Editor, including text search and replace, copy and paste, the Populate Fields toolbox, and File Import.

- The **Embedded Source** dialog (choose **Embeds** from the popup menu) lets you see *only the embed points and their code, without the surrounding code.* It gives you the full power of the Text Editor, plus a locator to find embed points, plus tools for moving and copying entire embed points with multiple blocks of embedded code, and for generating embedded code with Code templates.
- The **Embeds** button for a control (choose **Embeds** from the Window Formatter's popup menu) gives you the power of the **Embedded Source** dialog focused on *the embed points for a single control*.

Source code embedded with the Embeditor is fully accessible with the **Embedded Source** dialog and vice versa, with the exception of Code templates, which are only modifiable with the **Embedded Source** dialog.

Embeditor

1. From the Application Tree, RIGHT-CLICK the procedure, then choose **Source** from the popup menu.

   The Embeditor generates a temporary source file with shading and optional comments to identify all the embed points for the selected procedure. You may insert source code into the embed points simply by typing the new source statements into the unshaded or white area.

   Tip: You may configure the Embeditor's temporary source file with the Application and Editor tabs of the Application Options dialog. Choose Setup ➤ Application Options. See *Configuring the Environment—Action for Legacy embeds* and *Editor*.

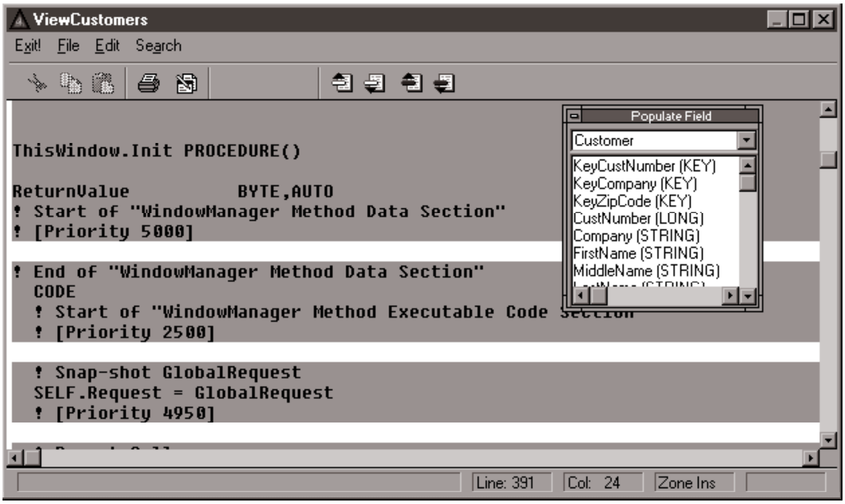
   The Embeditor is the Text Editor opened in a special mode which allows you to not only edit all the embed points in your procedure, but to edit them within the context of template-generated code. The Embeditor displays *all possible* embed points for the procedure within the context of *all* the *possible* code that *may* be generated for the procedure. Notice the distinction here—**Embeditor does not show you the code that *will* be generated, but all the code which *could be* generated**, if you placed code into every available embed point.

2. Press ⬇ to scroll to the next embed point.

   ⬆ scrolls to the previous embed point; ⬇ scrolls to the next filled embed point; ⬆ scrolls to the previous filled embed point.

3. Place the insertion point in the unshaded area, then type your source code.

   The full power of the Text Editor is at your disposal. See *Text Editor* for more information.



```
ThisWindow.Init PROCEDURE()

ReturnValue          BYTE,AUTO
! Start of "WindowManager Method Data Section"
! [Priority 5000]

! End of "WindowManager Method Data Section"
  CODE
  ! Start of "WindowManager Method Executable Code Section
  ! [Priority 2500]

  ! Snap-shot GlobalRequest
  SELF.Request = GlobalRequest
  ! [Priority 4950]
```

Note: The Embeditor automatically indents your source code at least as far as the embed point comments. You may indent farther (to the right), but you may not indent less (to the left).

4. Choose **Exit!** from the menu, then save when prompted.

   The Embeditor automatically puts your source into the appropriate embed point and sets the priority for the embedded code.

Embedded Source Dialog

1. From the Application Tree, RIGHT-CLICK the procedure, then choose **Embeds** from the popup menu.

   This opens the **Embedded Source** dialog, providing access to all the embed points in the procedure. You can also get here from the **Embeds** button on the **Procedure Properties** window, but the popup menu is quicker.

Tip: You may sort the embed points in alphabetical order or in logical order with the Application tab of the Application Options dialog. Choose Setup ➤ Application Options from the menu.

2. Filter the embed points by choosing from the **View** menu or by pressing buttons on the toolbar. Choose from:

   Expand All
   Fully expand the embeds list.

   Expand Filled
   Expand only the filled embeds.

   Contract All
   Fully contract the embeds list.

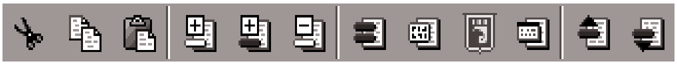
Show Filled Only
Show only filled embeds.

Show Priority Labels
Show template generated embed point labels so you can precisely interleave your code with template generated code.

Include Legacy Embeds
Show Clarion 2.x embed points.

Tip: **You may set the default for legacy embed points with the Application tab of the Application Options dialog. Choose Setup ➤ Application Options. See *Configuring the Environment—Action for Legacy embeds*.**

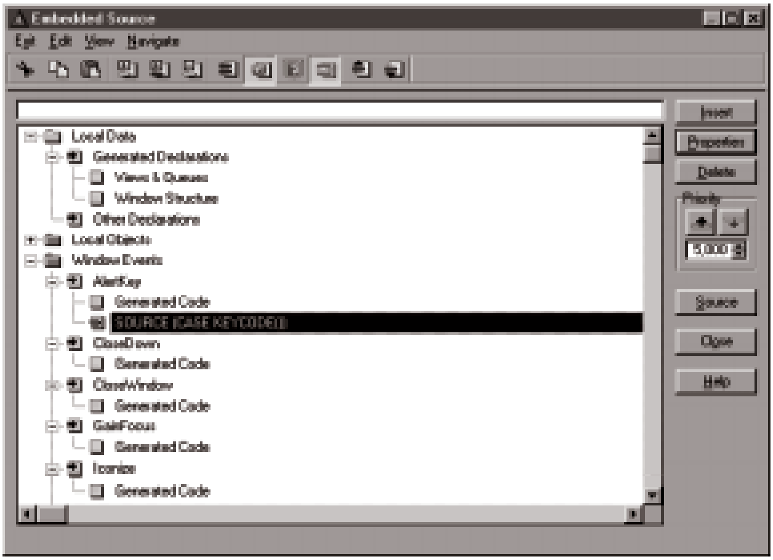
Display All
Available only when editing embeds for a control, this button allows you to expand the view to show embeds for the window.



3. Locate an embed point by typing its name in the locator field near the top of the dialog, or by choosing from **Navigate** menu or by pressing buttons on the toolbar. Choose from:

Previous Filled
Scroll to the next filled embed point.

Next Filled
Scroll to the next filled embed point.



Local Data
Generated Declarations
Views & Queues
Window Structure
Other Declarations
Local Objects
Window Events
AlertKey
Generated Code
SOURCE (CASE KEYCODE())
CloseDown
Generated Code
CloseWindow
Generated Code
GainFocus
Generated Code
Iconize
Generated Code

> **Tip:** **To embed code associated with a specific control, open the Window Formatter, RIGHT-CLICK the control and choose Embeds from the popup menu. Only those embed points associated with the selected control are listed.**

4. Select an embed point then press the **Insert** button.

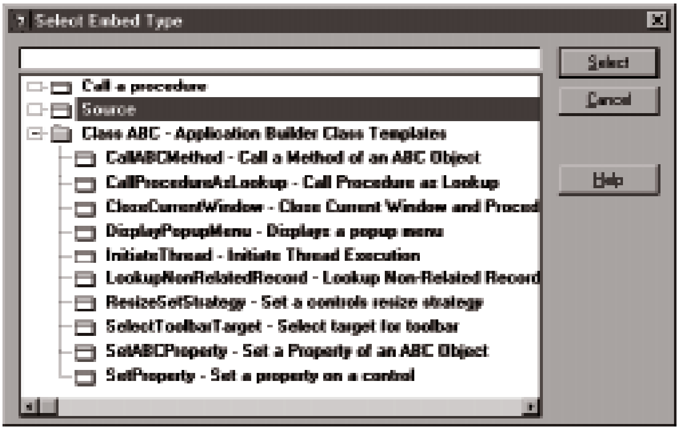
   This opens the **Select Embed Type** dialog. There are three ways to create the embedded source code: hand-coding with the text editor, calling another procedure, or embedding a Code template.

   You may combine one or more of these three methods at a single embed point—that is, a single embed point accepts multiple "blocks" of embedded code. You can control the execution sequence of each block of code relative to any other code in the embed point by setting its priority. Lower priority numbers execute before higher priority numbers.

   The **Embedded Source** dialog displays the embedded source in the order it generates and executes.

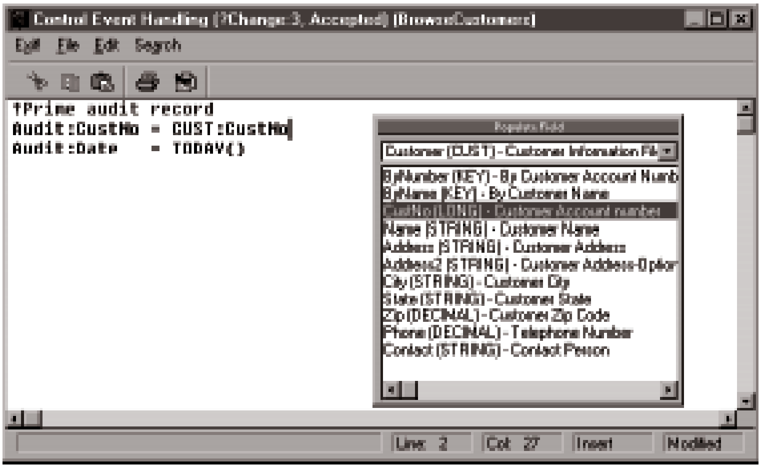
To "hand-code" embedded source with the Text Editor

1. Select SOURCE in the **Select Embed Type** dialog.


Select Embed Type
Select
Cancel
Help
Call a procedure
Source
Class ABC - Application Builder Class Templates
CallABCMethod - Call a Method of an ABC Object
CallProcedureAsLookup - Call Procedure as Lookup
CloseCurrentWindow - Close Current Window and Proced
DisplayPopupMenu - Displays a popup menu
InitiateThread - Initiate Thread Execution
LookupNonRelatedRecord - Lookup Non-Related Record
ResizeSetStrategy - Set a controls resize strategy
SelectToolbarTarget - Select target for toolbar
SetABCProperty - Set a Property of an ABC Object
SetProperty - Set a property on a control

2. Press the **Select** button to start the Text Editor with a blank source code window.

   This opens the Text Editor (see *Text Editor* for more information). The display includes a Populate Field toolbox from which you can select variable names and field names. Simply DOUBLE-CLICK on an item in the toolbox to insert its fully qualified name at the insertion point.



Control Event Handling (?Change:3, Accepted) [BrowseCustomers]
Exit
File
Edit
Search

```
!Prime audit record
Audit:CustNo = CUST:CustNo
Audit:Date   = TODAY()
```

Customer (CUST) - Customer Information Fil

ByNumber (KEY) - By Customer Account Numb
ByName (KEY) - By Customer Name
CustNo (LONG) - Customer Account number
Name (STRING) - Customer Name
Address (STRING) - Customer Address
Address2 (STRING) - Customer Address-Optior
City (STRING) - Customer City
State (STRING) - Customer State
Zip (DECIMAL) - Customer Zip Code
Phone (DECIMAL) - Telephone Number
Contact (STRING) - Contact Person

Line: 2 | Col: 27 | Insert | Modified

3. Write your custom code in the source code window.

> **Tip: Don’t forget to use the on-line help for explanations and examples of Clarion Language syntax and techniques. Copy and paste directly from the help examples!**

4. Choose **Exit!**.

5. Choose **Yes** when prompted to save the embedded source.

6. Optionally set the **Priority** for the embedded source.

   The **Priority** of each block within an embed point controls the execution sequence of the code relative to any other code in the same embed point. Lower priority numbers execute before higher priority numbers.

Configure the Embedded Source Dialog

From the environment main menu choose **Setup ➤ Application Options**, then select the **Embed Tree** tab. This tab lets you control the appearance of the *Local Objects* section of the embed tree. This tab only appears if you have the ABC Templates registered. Options specified here only affect the embed tree of an application using the ABC templates.

Show Procedure Keyword
Check this box to enable display of the PROCEDURE keyword in the Embed tree. This is a matter of personal preference.

Show Virtual Keyword
Check this box to enable display of the VIRTUAL keyword in the Embed tree. This allows you to identify embed points in virtual methods. You can also use color to identify virtuals.

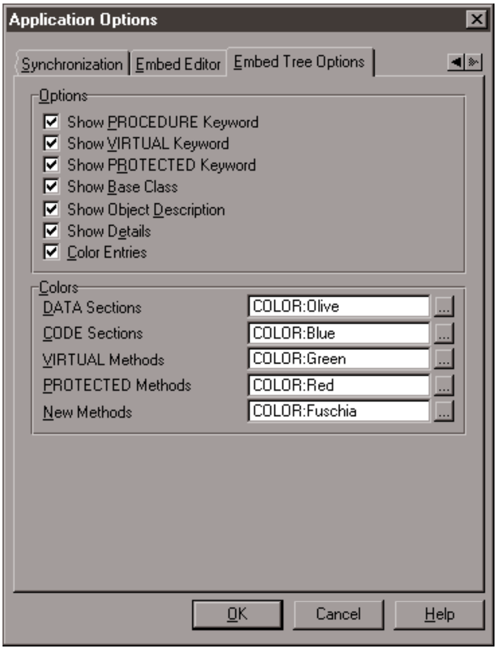
Show Protected Keyword
Check this box to enable display of the PROTECTED keyword in the Embed tree. This allows you to identify embed points in protected methods. You can also use color to identify protected methods.

Show Base Class

Check this box to display the name of the Local Object's base class. Use the base class to find object methods and properties in the *Application Handbook* or the on-line help.

Show Object Description

Check this box to display the Local Object's description. This shows information about the object. For example, a BrowseClass object's description displays the primary file it uses.



Options

- ☑ Show PROCEDURE Keyword
- ☑ Show VIRTUAL Keyword
- ☑ Show PROTECTED Keyword
- ☑ Show Base Class
- ☑ Show Object Description
- ☑ Show Details
- ☑ Color Entries

Colors

DATA Sections	COLOR:Olive
CODE Sections	COLOR:Blue
VIRTUAL Methods	COLOR:Green
PROTECTED Methods	COLOR:Red
New Methods	COLOR:Fuschia

Show Details

Check this box to display more detail about the Local Object. For example, display the condition that invokes a BrowseClass object's locator or step object.

Color Entries

Check this box to enable the Colors group so you can set different colors for different types of embed points. Choose from the following embed types.

DATA Sections

The color displayed for DATA sections in the embed tree. Enter a valid color equate, hexadecimal color value or press the ellipsis (...) button to select a color from the Color dialog. See \LIBSRC\EQUATES.CLW for a list of valid color equates.

CODE Sections

The color displayed for CODE sections in the embed tree. Enter a valid color equate, hexadecimal color value or press the ellipsis (...) button to select a color from the Color dialog. See \LIBSRC\EQUATES.CLW for a list of valid color equates.

VIRTUAL Methods

The color displayed for VIRTUAL methods in the embed tree. Enter a valid color equate, hexadecimal color value or press the ellipsis (...) button to select a color from the Color dialog. See \LIBSRC\EQUATES.CLW for a list of valid color equates.

PROTECTED Methods

The color displayed for PROTECTED methods in the embed tree. Enter a valid color equate, hexadecimal color value or press the ellipsis (...) button to select a color from the Color dialog. See \LIBSRC\EQUATES.CLW for a list of valid color equates.

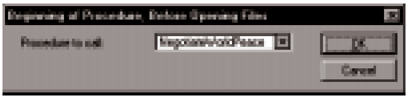
New Methods

The color displayed for New methods in the embed tree. Enter a valid color equate, hexadecimal color value or press the ellipsis (...) button to select a color from the Color dialog. See \LIBSRC\EQUATES.CLW for a list of valid color equates.

Call a Procedure

1. Select *Call a Procedure* in the **Select Embed Type** dialog.

   A dialog named for the embed point opens to accept the name of the procedure to call.



Procedure to call: NegotiateWorldPeace

2. In the **Procedure to Call** field, type a name for the procedure or choose an existing procedure from the drop-down list.

   Typing a new name tells the Application Generator to add the procedure to the Application Tree as a "To Do" item. If another procedure with the same name already exists, the Application Generator generates code to call it.

   You define the functionality of the other procedure separately. See *Defining Procedure Properties*.

3. Press the **OK** button to close the dialog.

Use a Code template to generate the embedded code

1. In the **Select Embed Type** dialog, select a Code template then press the **Select** button.

   Code templates are the items indented beneath the Class folders. See *Code and Extension Templates* in the *Application Handbook* for descriptions of the Code templates included with this package.

   This displays a **Prompts for**... dialog box.

2. Read the instructions and explanations in the dialog.

   Each code template includes explanatory text on its proper use and how to fill in the necessary options.

3. Fill in or choose from the options in the **Prompts for**... dialog.
4. Press the **OK** button to close the dialog.

Managing Embedded Source

The **Embedded Source** dialog contains several tools that let you control the sequence in which embedded source is listed and executed. The and **Priority** prompt and the ▲ and ▼ buttons change the order of one or more embedded source items; execution occurs in the order they are listed.

There are also **Delete** and **Properties** buttons, plus Cut, Copy, and Paste buttons for maintenance. To Cut and Paste (or Copy and Paste) embedded source from one embed point to another:

1. In the **Embedded Source** dialog, highlight a line in the tree diagram.

   Highlighting an embed point line (folder icon) selects *all* the embedded source at this embed point for subsequent cut and paste operations. Highlighting a single embed source item selects only that item.
2. Press the ✂ to cut, or press the ⧉ button to copy.
3. Again, highlight a line in the tree diagram.
4. Press the 📋 button to paste.

Copying Embedded Source Between Procedures

Occasionally you will create two or more procedures that are very similar and that require lots of embedded source code. Rather than retype the embedded source in the similar procedures, you can copy the embedded source as follows:

1. Develop and test the embedded code in your first procedure.
2. Choose **File ➤ Selective Export**.
3. Specify a .TXA file to receive the exported procedures, then press **OK**.
4. Select all the similar procedures for export, then press **OK**.

   Selected procedures are identified with a check mark.
5. With your favorite text editor, open the .TXA file and copy the embed definitions from the finished procedure to the other similar procedures, then save the .TXA file.

   The embed definitions commence with the [EMBED] line. See the *Programmer's Guide* for more information on TXA file format.
6. In Clarion Choose **File ➤ Import Text**.
7. Specify the .TXA file, then press **OK**.

   The import replaces the procedures in the the .APP with the procedures from the .TXA, with the embedded source code intact.

Procedure Formulas

❑ Press the **Formulas** button in the **Procedures Properties** dialog.

This opens the Formula Editor. The Formula Editor helps you create simple assignment statements and complex conditional structures. See the *Formula Editor* chapter for details on how to use the Formula Editor.

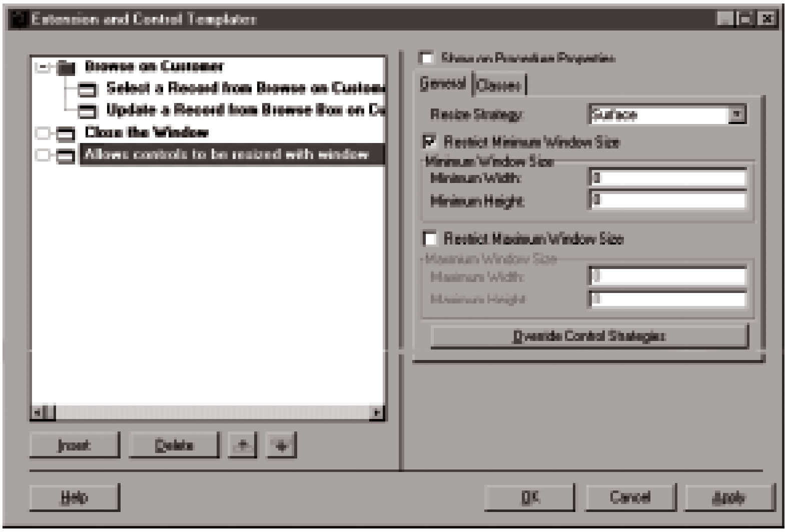
Procedure Extensions

❑ Press the **Extensions** button in the **Procedures Properties** dialog.

The **Extensions** button opens the **Extension and Control templates** dialog. This dialog lets you add, change, and delete Extension templates, and it lets you change existing Control template properties. To add and delete Control templates, use the Window Formatter.

Procedure extensions include both Extension templates and Control templates. These templates provide additional functionality to the basic Procedure templates. Control templates give your procedure the ability to display *and manage* a specific control. For example a browse box that displays, scrolls, and locates file data may be added with the BrowseBox control template.

Extension templates give your procedure additional functionality not associated with specific controls. For example, date and time displays may be added using an extension template. See the *Application Handbook* for specific information on each template.



Extension and Control Templates

- Browse on Customer
  - Select a Record from Browse on Custom
  - Update a Record from Browse Box on Cu
- Close the Window
- Allows controls to be resized with window

☐ Show on Procedure Properties

General | Classes

Resize Strategy: Surface

☑ Restrict Minimum Window Size

Minimum Window Size
Minimum Width: 0
Minimum Height: 0

☐ Restrict Maximum Window Size

Maximum Window Size
Maximum Width: 0
Maximum Height: 0

Prototyping and Parameter Passing

When calling procedures, you may want to pass parameters, return values, or both. You can define (prototype) parameters and return values with the **Procedure Properties** dialog.

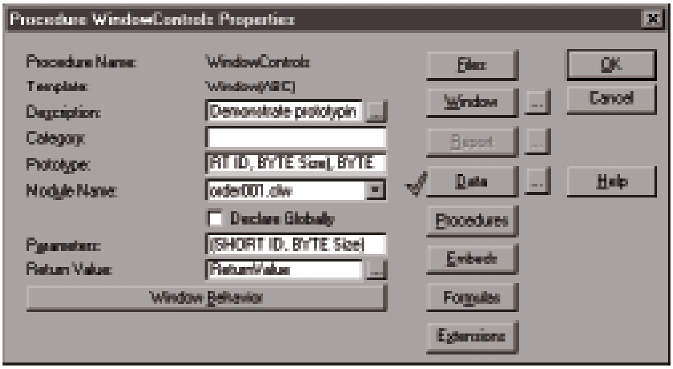
To pass parameters to a procedure, you must do the following.

1. Add the parameters' datatypes to the prototype in the MAP.
2. Add the parameters' names to the PROCEDURE statement.
3. Pass the parameters in the procedure call.

Adding Parameters to the Prototype

Use the **Prototype** field in the **Procedure Properties** dialog to redefine the prototype generated in the program or module MAP. The prototype declares everything the calling program needs to know to call the PROCEDURE, including the data types of any parameters. See *Procedure Prototyping* in the *Language Reference* for more information.

For example, type *(SHORT ID,BYTE Size),BYTE* in the **Prototype** field


Procedure WindowControls Properties

Procedure Name: WindowControls
Template: Window(ABC)
Description: Demonstrate prototypin
Category:
Prototype: RT ID, BYTE Size), BYTE
Module Name: order001.clw
□ Declare Globally
Parameters: (SHORT ID, BYTE Size)
Return Value: ReturnValue

to generate the following prototype statement in the module's MAP:

```
              MAP
                ...
  WindowsControls PROCEDURE(SHORT ID,BYTE Size),BYTE
                ...
              END
```

Notice the entire text from the **Prototype** field, including the parentheses, is appended to the prototype for the procedure. The words inside the parentheses are the datatypes and labels of the parameters passed to the procedure. The word following the parentheses is the datatype of the value returned by the procedure. See *Procedure Prototyping* in the *Language Reference* for more information.

Adding Parameters to the PROCEDURE Statement

Type *(SHORT ID, BYTE Size)* in the **Parameters** field in the **Procedure Properties** dialog to generate the following code for the procedure:

```
WindowsControls PROCEDURE(SHORT ID,BYTE Size)
  ...
```

Again, the entire text from the **Parameter** field, including the parentheses, is appended to the PROCEDURE statement.

In the **Return Value** field in the **Procedure Properties** dialog, press the ellipsis button (...) to select or define a return variable for a procedure, and to generate the following code. Notice the generated procedure now RETURNs the value of the return variable you specified in the **Return Value** field: *ReturnValue*.

```
WindowsControls PROCEDURE(ControlX,ControlY)
...
  CODE
  GlobalResponse = ThisWindow.Run()
  RETURN(ReturnValue)
```

Tip: You should add embedded code to assign the appropriate value to the returned variable.

Passing Parameters in the Procedure Call

Use the **Actions** tab of the calling control (menu item, button, etc.) to pass parameters to a procedure.

Passing Parameters to Procedures

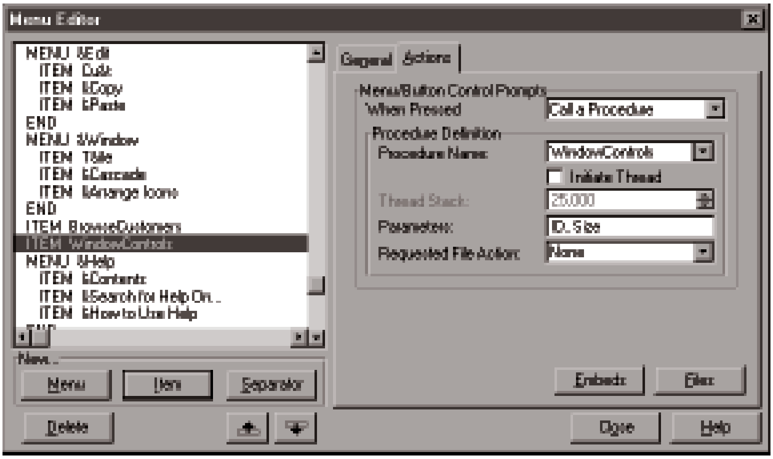
1. Go to the **Actions** tab for the control.
2. In the **When Pressed** drop-down list, choose *Call a Procedure*.
3. In the **Procedure Name** drop-down list, choose the name of the procedure to call.

   If you have not yet defined the procedure, type it's name. You can define the new procedure later.

Tip: The templates do not support parameters for the START statement, therefore, do not check the Initiate Thread box if you want to pass parameters. Checking the Initiate Thread box generates a START statement to start a new thread.

4. In the **Parameters** field, type the parameters to pass, separated by commas.

The parameters may be literal values, expressions, or variable names.



```
MENU &Edit
  ITEM Cu&t
  ITEM &Copy
  ITEM &Paste
END
MENU &Window
  ITEM T&ile
  ITEM &Cascade
  ITEM &Arrange Icons
END
ITEM BrowseCustomers
ITEM WindowControls
MENU &Help
  ITEM &Contents
  ITEM &Search for Help On...
  ITEM &How to Use Help
END
```

When Pressed: Call a Procedure

Procedure Name: WindowControls

☐ Initiate Thread

Thread Stack: 25,000

Parameters: ID, Size

Requested File Action: None

Receiving Return Values from Procedures

Although you may call a procedure with *Call a Procedure* from the **Actions** tab, this method does not allow you to receive return values. Therefore, you should use embedded source to receive a return value from a procedure. Following is one way to call a procedure from a control, however, you may call a procedure in many ways.

1. Go to the **Actions** tab for the control.
2. In the **When Pressed** drop-down list, choose *No Special Action*.
3. Press the **Embeds** button.
4. In the **Embedded Source** dialog, choose the Accepted embed point for Control Event Handling, then press **Insert**.
5. In the **Select Embed Type** dialog, select SOURCE.
6. In the Text Editor, type your procedure call, for example:

   ```
   IF WindowControls(0,75) THEN RETURN.
   ```

   This source statement passes an ID of 0 and a Size of 75 to the WindowControls procedure. It uses the IF statement to evaluate the return value.
7. Choose **Exit!** from the menu, then save your embedded source when prompted.

Maintaining Your Application

The **Application Tree** dialog provides five different views of your application. Each view has its own tab: **Procedure, Module, Template, Name**, and **Category**. You maintain the application with several environment menus: the **Edit** menu, the **Application** menu, the **Procedure** menu, and the **Popup** menu.

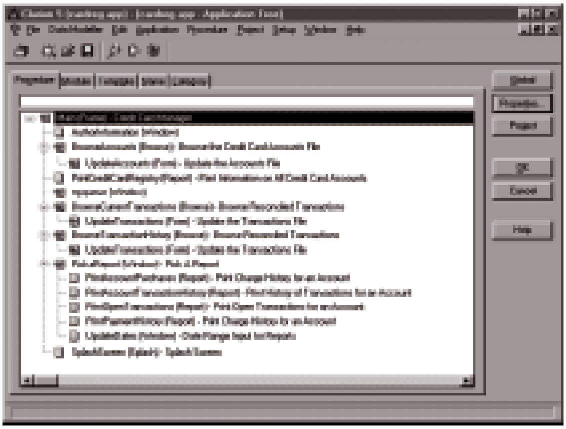
Application Tree Views/Tabs

The **Application Tree** dialog shows your procedures in five different views or arrangements. Change the view by selecting the corresponding tab.

Tip: For each hierarchical view (Procedure, Module, Template, and Category), CLICK **on the plus (+) sign to expand a tree branch;** CLICK **on the minus (-) sign to contract a branch.**

Procedure

Displays procedures in hierarchical order, nesting each procedure under its calling procedure. This is the default view, and combined with the "ToDo" legends for undefined procedures, is the best view for determining what remains to be done to complete the application.


File
DataModeller
Edit
Application
Procedure
Project
Setup
Window
Help
Procedure
Module
Template
Name
Category
Global
Properties...
Project
OK
Cancel
Help
AuthorInformation (Window)
BrowseAccounts (Browse) - Browse the Credit Card Accounts File
UpdateAccounts (Form) - Update the Accounts File
PrintCreditCardRegistry (Report) - Print Information on All Credit Card Accounts
UpdateTransactions (Form) - Update the Transactions File
PickaReport (Window) - Pick A Report
SplashScreen (Splash) - Splash Screen

Module

Each procedure appears nested under the name of the source file (module) to which its source code generates.

Tip: This tab is the only entry point to the Module level embed points and to the Default Program Properties dialog.

Template

Each procedure appears nested under its template type. This view is useful for making a single change to several similar procedures.

Name
Lists procedures alphabetically by name. This is sometimes convenient for large projects.

Category
Each procedure appears nested under its template category. This view is useful for making a single change to several similar procedures.

Locator

The **Application Tree** dialog provides a locator field—the wide entry box near the top of the dialog. Type the first letter of the procedure name, module name, or template type to select the first matching item. The locator is incremental: typing additional letters searches for a more specific match anywhere within the procedure list. Type CTLR+ENTER to advance to the next matching item.

Edit Menu—Edit Procedure Properties

The **Application Tree** dialog provides an Edit menu that lets you “edit” the properties of the selected procedure. With the **Application Tree** dialog active, you can execute the following commands from the **Edit** menu:

Properties
Calls the selected procedure’s **Procedure Properties** dialog. Equivalent to the **Properties** button.

Window
Calls the **Window Formatter** for the selected procedure. Equivalent to the **Window** button in the **Procedures Properties** dialog.

Report
Calls the **Report Formatter** for the selected procedure. Equivalent to the **Report** button in the **Procedures Properties** dialog.

Data
Calls the **Local Data** dialog for the selected procedure. Equivalent to the **Data** button in the **Procedures Properties** dialog.

Embeds
Calls the **Embedded Source** dialog for the selected procedure. Equivalent to the **Embeds** button in the **Procedures Properties** dialog.

Extensions
Calls the **Extension and Control Templates** dialog for the selected procedure, where you can add Extension templates, or you can

edit either Control or Extension templates. Equivalent to the **Extensions** button in the **Procedures Properties** dialog. See the *Control Templates* and *Code and Extension Templates* in the *Application Handbook*.

Source

Opens the Embeditor for the selected procedure. See *Embedded Source Code* for more information.

Find

Searches for a procedure by name. This can be very useful in a large application with dozens of procedures. Type a string to search for in the **Search for Procedure** dialog.

Find Next

Searches for another procedure using the same search string as the previous search. If there was no previous search, this invokes the **Find** command.

Edit by Name

Lets you type the name of a procedure in the **Edit Procedure by Name** dialog, then opens the **Procedure Properties** dialog of the procedure you named. This is useful in a large application with many procedures.

Delete

Deletes the selected procedure code, properties, etc. The procedure remains as a ToDo item in the Application Tree if it is called by another procedure. To remove the procedure from the Application Tree, you must remove all references to the procedure. See *Calls to Other Procedures*.

Application Menu—Edit Application Properties

With the **Application Tree** dialog active, you can execute the following commands from the **Application** menu:

Properties

Displays the **Application Properties** dialog for specifying fundamental changes to the application. See *Creating the Application (.APP) File* for more information on this dialog.

Global Properties

Displays the **Global Properties** dialog for specifying template based changes to the application. See the *Application Handbook—ABC Templates* for more information on this dialog. This is quivalent to using the **Global** button in the **Application Tree** dialog.

Change Dictionary

Sets a new data dictionary for the application. Type a file name in the **Select New Dictionary** dialog, or press the ellipsis (...) button to choose a new dictionary file from the **Open File** dialog.

If your procedures already reference fields in one dictionary, the Application Generator can only match fields from the new dictionary if both the FILE structure prefix and the RECORD fields are exactly the same. The **Select New Dictionary** dialog provides a warning message.

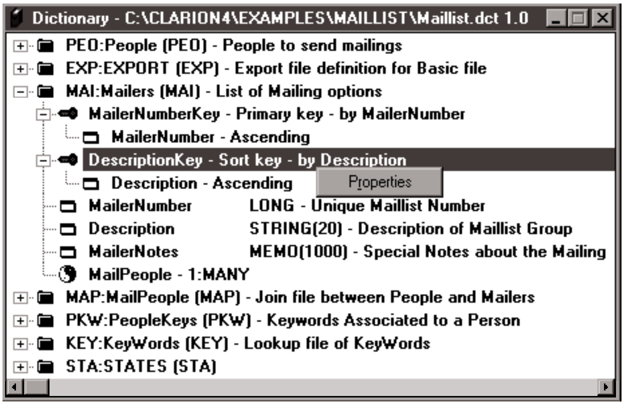
Select New Dictionary

Warning This is only guaranteed to work if the dictionaries are the same. It is not sufficient to have the same files and fields.

New Dictionary: [illegible]

View Dictionary

Opens the Dictionary Toolbox for the application's data dictionary. The Dictionary toolbox provides a hierarchical list representing your database. This list shows database files, keys, key components, fields, and relationships in an expanding hierarchical tree. The Dictionary toolbox can update all the *existing* items in the application's data dictionary except the dictionary properties.



Dictionary - C:\CLARION4\EXAMPLES\MAILLIST\Maillist.dct 1.0

```
PEO:People (PEO) - People to send mailings
EXP:EXPORT (EXP) - Export file definition for Basic file
MAI:Mailers (MAI) - List of Mailing options
   MailerNumberKey - Primary key - by MailerNumber
      MailerNumber - Ascending
   DescriptionKey - Sort key - by Description
      Description - Ascending
   MailerNumber      LONG - Unique Maillist Number
   Description       STRING(20) - Description of Maillist Group
   MailerNotes       MEMO(1000) - Special Notes about the Mailing
   MailPeople - 1:MANY
MAP:MailPeople (MAP) - Join file between People and Mailers
PKW:PeopleKeys (PKW) - Keywords Associated to a Person
KEY:KeyWords (KEY) - Lookup file of KeyWords
STA:STATES (STA)
```

Insert Module

Opens the **Select Module Type** dialog where you specify a new MODULE for your application. Highlight an item from the list, then press the Select button.

ExternalDLL	External Dynamic Link Module--Use this choice when the module to add is a DLL.
ExternalLib	External Library Module--Use this choice when the module to add is a .LIB (without a .DLL).
ExternalObj	External Object Module--Use this choice when the module to add is an .OBJ file.
ExternalSource	Use this choice when the module to add is Source Code that will not be maintained by the Application Generator. The Application Generator reads and compiles this source file but does not write to it.

GENERATED Use this choice when the module to add is source code which is generated by the Application Generator. This allows you to control the name of the Module.

Template Utility

Lets you run any utility templates that are registered You may write your own utilities or acquire utilities from third party vendors. Use this command to start any of Clarion's Wizards. See *Wizards and Utility Templates* in the *Application Handbook*.

Synchronize

Applies the Data Dictionary's respective default control specifications, including Screen Picture, Prompt, Heading, Case, Typing Mode, Flags, Justification, Initial Value, Help IDs, Messages, Tool Tips, Validity Checks, etc. to all the controls in the application, with the following exceptions.

The control's type and position do not change. The control's height and width may change if they are set to "Default." Blank dictionary attributes do not replace filled control attributes (e.g.. Tool Tips). Controls placed by a Control template retain those attributes required by the template. Controls that are explicitly frozen do not change. See *Controls and Their Properties—Common Control Properties* for information on freezing controls.

Redistribute Procedures

Redistributes the procedures among the modules *in the order in which they already occur*. The number of procedures contained in each module is determined by the **Procedures Per Module** you specify. This utility immediately affects the Application Tree, but generated source modules are not affected until the next time source is generated.

Repopulate Modules

Redistributes the procedures among the modules *trying to keep procedures that call each other in the same module*. The number of procedures contained in each module is determined by the **Procedures Per Module** you specify. This utility immediately affects the Application Tree, but generated source modules are not affected until the next time source is generated.

Renumber Modules

Renumbers modules. This is useful when empty modules have been deleted. This utility immediately affects the Application Tree, but generated source modules are not affected until the next time source is generated.

Delete Empty Modules

Removes modules from the Application Tree that have become empty as a result of application changes or deletions. This menu option immediately affects the Application Tree but it does not delete module files (.CLW) on your disk drive.

Delete Empty Libraries

Removes libraries from the Application Tree that have become empty as a result of application changes or deletions. This menu option immediately affects the Application Tree but it does not delete library files on your disk drive.

Procedure Menu—Edit Procedure Properties

With the **Application Tree** dialog active, you can execute the following commands from the **Procedure** menu:

New

Adds a new procedure not connected to the procedure tree. The INSERT key does the same thing.

Rename

Changes the name of the selected procedure. Type a new name in the **Rename** dialog box.

Copy

Copies the selected procedure. Type a new name in the **New Procedure** dialog box.

Synchronize

Applies the Data Dictionary's respective default control specifications, including Screen Picture, Prompt, Heading, Case, Typing Mode, Flags, Justification, Initial Value, Help IDs, Messages, Tool Tips, Validity Checks, etc. to all the controls in the procedure, with the following exceptions.

The control's type and position do not change. The control's height and width may change if they are set to "Default." Blank dictionary attributes do not replace filled control attributes (e.g.. Tool Tips). Controls placed by a Control template retain those attributes required by the template. Controls that are explicitly frozen do not change. See *Controls and Their Properties—Common Control Properties* for information on freezing controls.

Change Module

Lets you move the currently selected procedure from one source module to another. Select the destination in the **Select Destination Module** dialog.

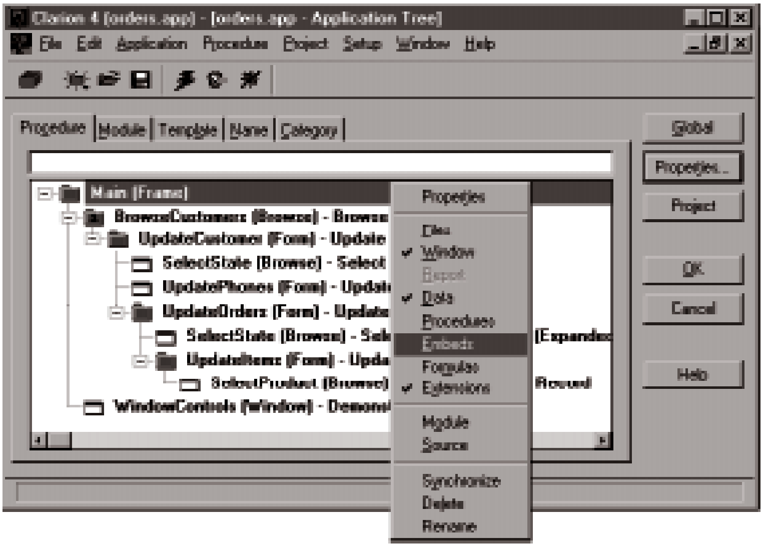
Your application may execute slightly faster if you group procedures which commonly execute together in the same module.

Change Template

Lets you change the procedure type for the currently selected procedure. Select a new procedure template in the **Select Procedure Type** dialog. If the new procedure type doesn't support some of the structures—such as menus—that you defined in the previous procedure type, you may “orphan” the previously defined structures. Therefore, be cautious when changing procedure type.

Popup Menu—Edit Procedure Properties

With the **Application Tree** dialog active, right-click on any procedure to access the following commands. This popup menu serves as a shortcut to the procedures’ properties.


Clarion 4 (orders.app) - [orders.app - Application Tree]
File
Edit
Application
Procedure
Project
Setup
Window
Help
Procedure
Module
Template
Name
Category
Main (Frame)
BrowseCustomers (Browse) - Browse
UpdateCustomer (Form) - Update
SelectState (Browse) - Select
UpdatePhones (Form) - Updat
UpdateOrders (Form) - Update
SelectState (Browse) - Sel
UpdateItems (Form) - Upda
SelectProduct (Browse)
WindowControls (Window) - Demons
[Expande
Round
Properties
Files
✓ Window
Report
✓ Data
Procedures
Embeds
Formulas
✓ Extensions
Module
Source
Synchronize
Delete
Rename
Global
Properties...
Project
OK
Cancel
Help

Properties

Opens the currently selected procedure’s **Procedure Properties** dialog. Equivalent to the **Properties** button.

Files

Opens the selected procedure’s **File Schematic Definition** dialog. Equivalent to the **Files** button in the **Procedures Properties** dialog.

Window

Opens the **Window Formatter** for the selected procedure. Equivalent to the **Window** button in the **Procedures Properties** dialog.

Report

Opens the **Report Formatter** for the selected procedure—same as the **Report** button in the **Procedures Properties** dialog.

Data

Opens the **Local Data** dialog for the selected procedure—same as the **Data** button in the **Procedures Properties** dialog.

Procedures

Opens the **Called Procedures** dialog for the selected procedure—same as the **Procedures** button in the **Procedures Properties** dialog.

Embeds

Opens the **Embedded Source** dialog for the selected procedure—same as the **Embeds** button in the **Procedures Properties** dialog. See *Embedded Source Code* above for more information.

Formulas

Opens the **Formula Editor** for the selected procedure—same as the **Formulas** button in the **Procedures Properties** dialog.

Extensions

Opens the **Extension and Control Templates** dialog for the selected procedure, where you can add Extension templates, or you can edit either Control or Extension templates—same as the **Extensions** button in the **Procedures Properties** dialog. See the *Control Templates* and *Code and Extension Templates* in the *Application Handbook* for more information.

Module

Opens the source file for the selected procedure, provided the source has been generated. Any changes you make to the generated source are overwritten next time the source is generated. This lets you experiment before permanently making changes with other Application Generator tools.

Tip: The Text Editor displays the source code that was last generated. Thus, if you have changed the application *after* the last code generation, your changes are not reflected in the Module display. Choose Project ➤ Generate to update your source code.

Source

Opens the Embeditor which lets you embed your own source code within the context of the surrounding generated code. See *Embedded Source Code* for more information.

Synchronize

Applies the Data Dictionary’s respective default control specifications, including Screen Picture, Prompt, Heading, Case, Typing Mode, Flags, Justification, Initial Value, Help IDs, Messages, Tool Tips, Validity Checks, etc. to all the controls in the procedure, with the following exceptions.

The control’s type and position do not change. The control’s height and width may change if they are set to “Default.” Blank dictionary attributes do not replace filled control attributes (e.g., Tool Tips). Controls placed by a Control template retain those attributes required by the template. Controls that are explicitly frozen do not change. See *Controls and Their Properties—Common Control Properties* regarding freezing controls.

Delete

Deletes the selected procedure from the application (but does not remove the source module from the disk).

Rename

Renames the selected procedure.

File Menu—Application Import/Export Commands

When the Application Generator is active, the **File** menu contains commands for importing and exporting application procedures. This import and export process is managed through the use of a special file format (.TXA) designed to help incorporate procedures written with other versions of Clarion. See *TXA File Format* in the *Programmer's Guide* for more information.

The following commands are available on the **File** menu:

Import from Application

Lets you select an .APP file from the **Select Application to Import From** dialog. After you make the selection and press the **OK** button, the Application Generator adds all the procedures from the selected .APP file to the Application Tree.

Name conflicts (same procedure name in both applications) are resolved according to the **Application Options** settings. See *Configuring the Application Generator—Import name clash action* for complete information on these settings.

Warning! Be sure to back up both source and target before importing. As part of the import process, Clarion converts the source file to the new version format. Once converted to the new format, earlier versions of Clarion cannot read or use the files.

Import Text

Imports the procedures defined in a .TXA file (an ASCII version of an .APP file), created with the **Export Text** (see below) command. You will be prompted to rename or replace procedures with name conflicts. Name conflicts (same procedure name in .APP and .TXA) are resolved according to the **Application Options** settings. See *Configuring the Application Generator—Import name clash action* for complete information on these settings.

Export Text

Creates a .TXA file (an ASCII version of your .APP file) containing all the application's procedures and properties.

Selective Export

Creates a .TXA file (an ASCII version of your .APP file) containing only the procedures you specify.

Configuring the Application Generator

The **Application Options** dialog lets you specify default settings for each *new* application you create as well as for the active application. To access the dialog, choose **Setup ➤ Application Options**. The dialog is divided into five sections or tabs: **Application**, **Registry**, **Generation**, **Synchronization**, **Editor.**

Application

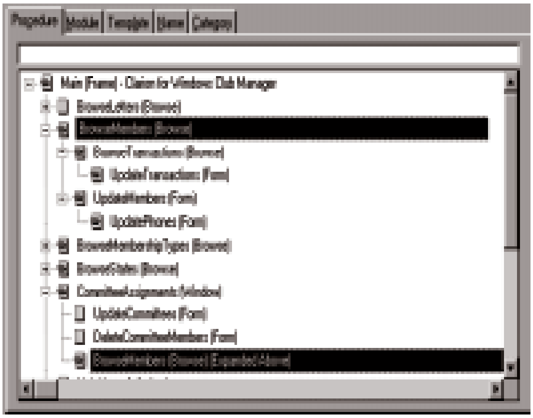
This tab lets you specify several miscellaneous defaults pertaining to the application.

Require a dictionary
Each new application *must* have a data dictionary.

Default dictionary
Specifies the default data dictionary filename to use when you creaate a new application. You may override the default.

Tip: **A single data dictionary may be used to support multiple applications.**

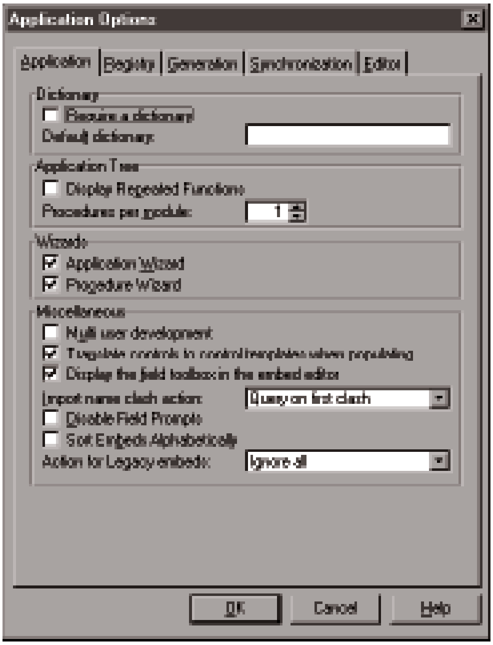
Display Repeated Functions
Check this box to have the Application Generator (Procedure tab only) provide full expansion of procedures called from more than one place in the Application Tree. Clear the box to provide full expansion of only the first instance—every other instance is marked “Expanded Above.”



```
Main (Frame) - Clarion for Windows Club Manager
  BrowseLetters (Browse)
  BrowseMembers (Browse)
    BrowseTransactions (Browse)
      UpdateTransactions (Form)
    UpdateMembers (Form)
      UpdatePhones (Form)
  BrowseMembershipTypes (Browse)
  BrowseStates (Browse)
  CommitteeAssignments (Window)
    UpdateCommittees (Form)
    DeleteCommitteeMembers (Form)
    BrowseMembers (Browse) (Expanded Above)
```

Procedures per module
Specifies the number of procedures the Application Generator writes to each source code module. This affects compile times when the **Conditional Generation** switch is turned on. It can also have a small effect on runtime performance.

Specifying one procedure per module causes Clarion to compile only those procedures changed since the last compile. However, this requires more disk space and more file accesses. We recommend 3-5 procedures per module for the Clarion Templates and 15-20 procedures per module for the ABC Templates.



Dictionary
- □ Require a dictionary
- Default dictionary:

Application Tree
- □ Display Repeated Functions
- Procedures per module: 1

Wizards
- ☑ Application Wizard
- ☑ Procedure Wizard

Miscellaneous
- □ Multi user development
- ☑ Translate controls to control templates when populating
- ☑ Display the field toolbox in the embed editor
- Import name clash action: Query on first clash
- □ Disable Field Prompts
- □ Sort Embeds Alphabetically
- Action for Legacy embeds: Ignore all

Application Wizard

Sets the default value of the **Application Wizard** check box on the **Application Properties** dialog when creating a new application. The **Application Wizard** builds an entire application based on your data dictionary. See *Wizards and Utility Templates* in the *Application Handbook* for moreinformation.

Procedure Wizard

Sets the default value of the **Procedure Wizard** check box on the **Select Procedure Type** dialog when creating a new procedure. Checking this box will invoke a **Wizard** to help you define your Browse, Form, and Report procedures. See *Wizards and Utility Templates* in the *Application Handbook* for moreinformation.

Multi user development

Opens the Data Dictionary and the Template Registry in read only mode, so many developers can work with the same dictionary. See *Development and Deployment Strategies* for more information.

Translate controls to control templates when populating

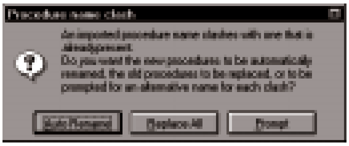
Check this box to have the Window Formatter prompt you with a list of control templates whenever you place a control. We recommend this setting for new Clarion users to encourage the use of templates whenever possible. Generally it is to your advantage to use Control templates rather than simple controls. Control templates define a control and the source code to manage it.

Import name clash action

Specifies how the Application Generator handles procedure names from an imported application file which clash with procedure names already resident. Choose from the following:

Query on first clash When the first clash is encountered, the Application Generator prompts for specific instructions on how to handle this clash and each subsequent clash. Choose from:

Auto Rename Renames all imported procedures with name clashes by appending a sequence number to the imported procedure name.



Procedure name clash

An imported procedure name clashes with one that is already present.
Do you want the new procedures to be automatically renamed, the old procedures to be replaced, or to be prompted for an alternative name for each clash?

Replace All Replaces all resident procedures with imported procedures of the same name.

Prompt Asks for specific instructions for each clash encountered. Choose from:

Replace Replaces the resident procedure with the imported procedure of the same name.

Rename Prompts you to rename the imported procedure.

Ask for alternative For all procedures with the same name, the Application Generator prompts you to rename each imported procedure.

Auto Rename For all procedures with the same name, the Application Generator renames the imported procedure by appending a sequence number to the name.

Replace previous For all procedures with the same name, the Application Generator replaces the resident procedures with the imported procedures.

Disable Field Prompts

Check this box to suppress template generated control-specific prompts on **Action** tabs (affects the properties dialog for CHECK, BUTTON, ITEM, and ENTRY controls). This does not disable Control template prompts. This can reduce "clutter" when you are using Control templates rather than simple controls.

Sort Embeds Alphabetically

Check this box to show embed points in alphabetical order in the **Embedded Source** dialog. Clear the box to show the embed points in "logical" order (order of execution). See *Embedded Source Code* for more information.

Tip: The logical sort sequence is most useful with the Clarion template chain. The ABC Template embed points (object oriented) do not lend themselves to logical ordering.

Action for Legacy embeds

Specify how the Application Generator handles legacy embed points. Legacy embed points are generally provided for backward compatibility among template chains. They allow newer template chains to conditionally support embed points from older template chains. See *LEGACY* in the *Programmers Guide*. Choose from:

Ignore all
Application Generator neither displays Legacy embed points at design time nor generates any code embedded therein. We recommend this setting to reduce "clutter" when developing new applications.

Show all and generate
Application Generator displays all Legacy embed points at design time and generates any code embedded therein. We generally do not recommend this setting; however, it can be useful for developers that are very comfortable with a particular template chain and its embed points.

Show filled and generate
Application Generator displays only filled Legacy embed points at design time and generates any code embedded therein. We recommend this setting for applications ported to a new template chain.

Registry

This tab lets you specify application options concerning maintenance and use of the Template Registry. See *Configuring the Template Registry*.

Generation

This tab lets you specify application options concerning source code generation.

Conditional Generation

Specifies that only source code modules *changed* since the last make should be compiled.

Debug Generation

Turns debug logging on and off and specifies a **Debug Filename** for the Application Generator to log events to. In case of a fatal error by the Application Generator, this log provides a trace for TopSpeed Technical Support to identify where the problem occurred.

Generation Message

Specifies what and how many messages are displayed during source code generation. Choose from

No Messages,
Module Names only (#MESSAGE line 1),
Module and Procedure Names (#MESSAGE lines 1-2), and
All Messages (#MESSAGE lines 1-3).

See *#MESSAGE* in the *Programmers Guide* for more information.

Enable #ASSERT checking

Check this box to enforce heightened error checking during source code generation. This allows the Application Generator to identify certain template execution problems and notify you during source code generation. This slows the code generation process slightly. Clear the box for faster, but riskier source code generation. See *#ASSERT* in the *Programmers Guide* for more information.

Create local maps

Check this box to generate a MAP structure for each source module that prototypes only the procedures referenced in the module. This results in faster compiles whenever you add new procedures or change procedure prototypes, because only the affected modules are recompiled. To generate accurate local maps, you must use the **Procedures** button in the **Procedure Properties** dialog to identify any procedures referenced in embedded source code.

Clear this box to generate a single MAP for the PROGRAM module that prototypes all the program’s procedures globally. This results in slower compile times whenever you add new procedures or change procedure prototypes, because the change to the PROGRAM module forces a recompile of all application source modules.

Enable embed commenting

Check this box to optimize automatic comment generation specified by the application templates. See T*emplate Overview—General Tab Options—Generate EMBED Comments* in the *Application Handbook* for more information.

Synchronization

This tab lets you specify how and when control attributes defined in your data dictionary are applied to your application's procedures and controls. See also *Application Menu—Synchronize*, and *Controls and Their Properties—Common Control Attributes*.

Synchronize Application when opened

Check this box to reapply data dictionary attributes each time your application is opened. Clear the box to apply the attributes only on your explicit command. See *Application Menu—Synchronize*.

Synchronize Window definitions

Check this box to apply data dictionary attributes to WINDOW structures during application-wide synchronization. Clear the box to ignore WINDOW structures.

Synchronize Report definitions

Check this box to apply data dictionary attributes to REPORT structures during application-wide synchronization. Clear the box to ignore REPORT structures.

Update controls for variables

Check this box to apply memory variable control attributes to their associated controls. Clear the box to ignore controls associated with memory variables.

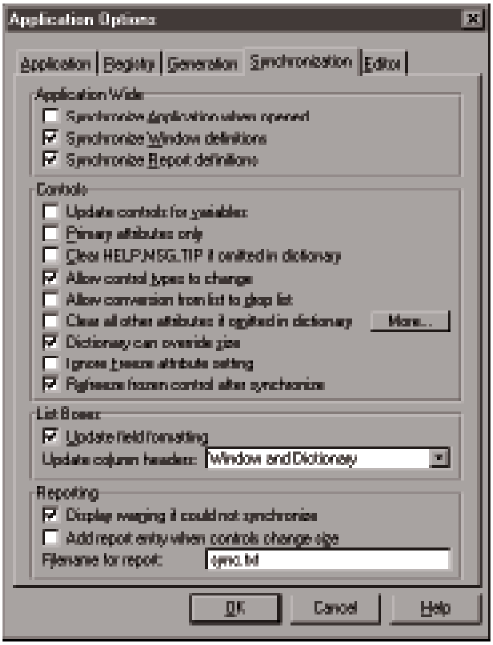
Primary attributes only

Check this box to apply only the primary control attributes. Primary attributes are those attributes set on the **General, Attributes, Help** and **Validity Checks** tabs of the Dictionary Editor's **Field Properties** dialog. They include Field Name, Characters (length) Screen Picture, Prompt Text, Column Heading, Case (UPR, CAP), Typing Mode (INS, OVER), Flags (IMM, PASSWORD, READONLY), Justification, Initial Value, Help IDs (HLP), Messages (MSG), Tool Tips (TIP), and Validity Checks. Secondary attributes are those attributes set on the **Window** and **Report** tabs of the Dictionary Editor's **Field Properties** dialog.

Tip: Check the Primary attributes only box to speed up the synchronization process, especially if you synchronize each time you open the application.

Clear HLP, MSG, TIP if omitted in dictionary
Check this box to override control specific help attributes (set from the **Window Formatter**) with blank help attributes from the data dictionary. Clear the box to retain control specific help attributes despite blank help attributes in the dictionary.

Allow control types to change
Check this box to apply new control types. For example, a SPIN may replace an ENTRY.



Application Options

Application | Registry | Generation | Synchronization | Editor

Application Wide
- □ Synchronize Application when opened
- ☑ Synchronize Window definitions
- ☑ Synchronize Report definitions

Controls
- □ Update controls for variables
- □ Primary attributes only
- □ Clear HELP,MSG,TIP if omitted in dictionary
- ☑ Allow control types to change
- □ Allow conversion from list to drop list
- □ Clear all other attributes if omitted in dictionary
- ☑ Dictionary can override size
- □ Ignore freeze attribute setting
- ☑ Refreeze frozen control after synchronize

List Boxes
- ☑ Update field formatting
- Update column headers: Window and Dictionary

Reporting
- ☑ Display warning if could not synchronize
- □ Add report entry when controls change size
- Filename for report: sync.txt

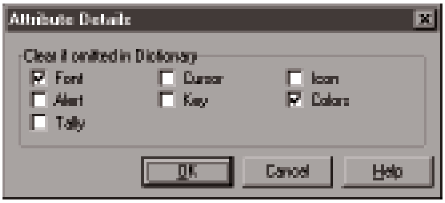
Note: **Changing a control's type can result in "orphaned" embed code. For example a SPIN supports a NewSelection embed point, but an ENTRY does not. Orphaned embed code should be manually moved to an appropriate place.**

Allow conversion from list to drop list
Check this box to allow a drop list to replace a list.

Clear all other attributes if omitted in dictionary
Check this box to override all control specific attributes (attributes set from the **Window Formatter**) except HELP, TIP, and MSG with blank attributes from the dictionary. Clear the box to retain control specific attributes despite blank attributes in the dictionary. Clearing the box also enables the **More** button so you can set each attribute individually.

More
Press this button to elect, for each individual attribute, whether to override the control specific (**Window Formatter**) attribute with

a blank attribute from the dictionary or whether to keep the attribute despite a blank attribute in the dictionary. Attributes are **Font, Alert, Tally, Cursor, Key, Icon,** and **Colors.**



Attribute Details

Clear if omitted in Dictionary

☑ Font ☐ Cursor ☐ Icon
☐ Alert ☐ Key ☑ Colors
☐ Tally

Dictionary can override size

Check this box to let data dictionary size attributes prevail over **Window Formatter** size attributes. Control sizes can change when the height or width value is default and the control's text changes, or when an explicit height or width value in the dictionary varies from the control specific (**Window Formatter**) height or width values.

Ignore Freeze attribute setting

Check this box to apply data dictionary attributes to controls with the #Freeze attribute. Clear the box to leave frozen controls alone. See *Controls and Their Properties—Common Control Attributes—Setting Control Modes*.

Refreeze frozen control after synchronize

Check this box to" refreeze" the control after synchronizing it. Clear the box to "unfreeze" the control after synchronizing.

Update field formatting

Check this box to apply dictionary attributes to LIST FORMAT strings (i.e. justification). In other words, format list box fields according to data dictionary attributes. Clear this box to leave LISTs alone.

Update column headers

Select from the drop-down list to specify when List Box column headers are applied from the data dictionary. Choose from:

Always The Application Generator always applies the dictionary column header, even if it is blank.

If present in dictionary
The Application Generator applies the dictionary column header only if it is non-blank, otherwise, the LISTs column header prevails.

Window and Dictionary
The Application Generator applies the dictionary column header only if *both* are non-blank, otherwise, the LISTs column header prevails.

Never The Application Generator never applies the dictionary column header. The LISTs column header always prevails.

Display warning if could not synchronize

Check this box to display a warning dialog if warnings occur during synchronization. Warnings occur when controls change size, when there is a different number of radio buttons in the dictionary than on the window or when an OPTION is replaced with a different control (e.g. a drop-down list).

Add report entry when controls change size

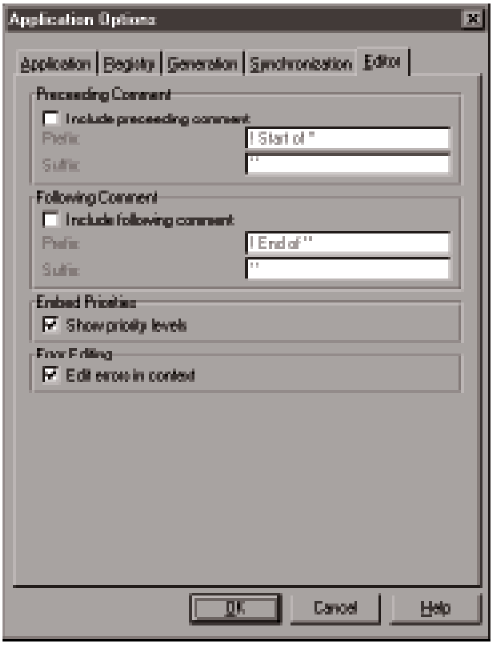
Check this box to generate warnings when controls change size. Clear the box to suppress size change warnings.

Filename for report

Specify the file to hold the warning report. Clearing this box suppresses the report.

Editor

This tab lets control how the Application Generator generates the temporary source file for the Embeditor. You can specify the text that delimits the embed points within the temporary source file. By customizing the text, you can make it easy to identify the embed points you want to edit. See *Embedded Source Code* and see *Text Editor* for more information.



Preceding Comment

□ Include preceding comment

Prefix: ! Start of "

Suffix: "

Following Comment

□ Include following comment

Prefix: ! End of ''

Suffix: ''

Embed Priorities

☑ Show priority levels

Error Editing

☑ Edit errors in context

Preceeding Comment

Specify the text that marks the beginning of each embed point.

Include preceeding comment

Check this box to generate a preceeding comment. Clear the box to omit the preceeding comment.

Prefix Set the text generated before the embed point name.

Suffix Set the text following the embed point name.

Following Comment

Specify the text that marks the end of each embed point.

Include following comment

Check this box to generate a trailing comment. Clear the box to omit any trailing comment.

Prefix Set the text generated before the embed point name.

Suffix Set the text following the embed point name.

Show priority levels

Check this box to show the embedded source priority within the Embeditor. The priority determines the sequence in which the Application Generator places multiple blocks of embedded code within a single embed point.

Edit errors in context

This box controls which edit mode to invoke when you edit embedded source code from the **Make Status** dialog (see *Clarion's Development Environment—Project Commands* for more information). Check this box to open the Embeditor (equal to **Edit ➤ Source**) to edit embedded source code. Clear the box to open the non-contextual embed editor (equal to **Edit ➤ Embeds**).

Templates and the Template Registry

Template files (*.TPL) drive the Application Generator. Each Procedure template contains some source code, as well as prompts for additional information needed to complete the procedure. The templates are *interactive*—they process the information you specify when you design the application. Clarion evaluates the template file twice:

- Before creating an application, Clarion preprocesses registered template classes and stores the information in the ..\TEMPLATE\REGISTRY.TRF file. Preprocessing occurs only when the Application Generator detects a new or changed template.

  When it preprocesses the template set, the Application Generator stores a list of all the information you must provide to each template. It also determines the points where you can embed source code to customize a procedure (see *Application Generator—Embedded Source Code*).

- At code generation time, the Application Generator uses the information you provided in the **Procedure Properties** dialogs, information from the Data Dictionary, the .APP file, the template language statements, and symbols in the REGISTRY.TRF file to generate your source code.

Each template class can contain multiple templates which you use to create the procedures in your application. Before you can use a template it must be in the Template Registry. See *Registering Templates*.

Configuring the Template Registry

To set Template Registry options, choose **Setup ➤ Application Options**. This opens the **Application Options** dialog. Select the **Registry** tab.

Template language code can be stored among many files, typically .TPW and .TPL files. Clarion uses these files to produce one logical template class (..\TEMPLATE\REGISTRY.TRF). Think of the .TPW and .TPL files as the source or backup of your templates, and the REGISTRY.TRF file as your working copy.

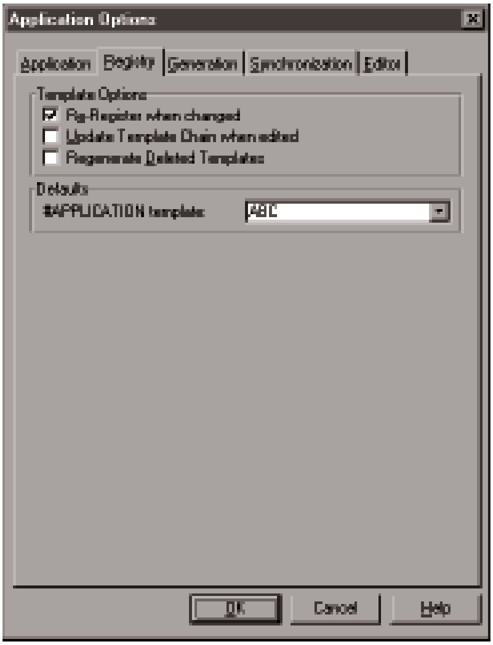
The **Registry** tab lets you specify how the template language files and the one logical template class are managed for your applications. These options are mainly for developers who produce their own template files or make modifications to the default templates.

Re-Register when changed

Check this box to automatically reregister your templates when the Application Generator detects a change—that is, when a .TPL or .TPW file changes, Application Generator copies the change to the REGISTRY.TRF file, .

Update Template Chain when edited

Check this box to automatically update the template files (.TPL and .TPW) when you use the **Template Registry** to edit the REGISTRY.TRF file.



Template Options

☑ Re-Register when changed

☐ Update Template Chain when edited

☐ Regenerate Deleted Templates

Defaults

#APPLICATION template: ABC

Regenerate Deleted Templates

Check this box to specify the Application Generator should replace any deleted .TPL or .TPW files from the registered templates (\TEMPLATE\REGISTRY.TRF).

#APPLICATION template

Select the APPLICATION template from the drop-down list. The APPLICATION template controls source code generation. See *Template Overview* in the *Application Handbook* for more information.

Registering Templates

The Template Registry stores a list of all the templates available to you when building an application. To create an application, you must have at least one template class registered.

The default template class is preregistered. However, if you need to reregister it, or if you wish to register a third party template class, this is how to do it:

1. Choose **Setup ➤ Template Registry**.

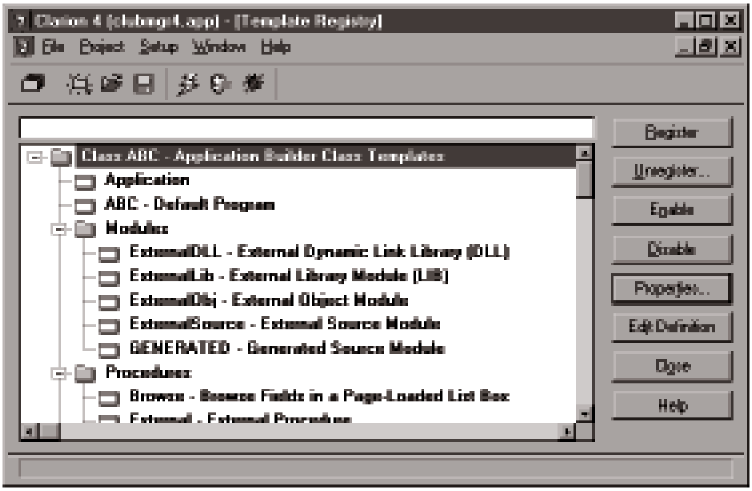
This opens the **Template Registry** dialog which provides a hierarchical list of the templates, plus some command buttons to maintain them.

2. Press the **Register** button.

   This opens the **Template File** dialog which lets you choose the templates to register. The **List Files of Type** specification is *.TPL. The default template subdirectory or folder is ..\Clarion5\TEMPLATE.

3. Select the .TPL files to register, then press the **Open** button.

   This registers (preprocesses) the template sets (classes), making them available for use in your applications.



Class ABC - Application Builder Class Templates
Application
ABC - Default Program
Modules
ExternalDLL - External Dynamic Link Library (DLL)
ExternalLib - External Library Module (LIB)
ExternalObj - External Object Module
ExternalSource - External Source Module
GENERATED - Generated Source Module
Procedures
Browse - Browse Fields in a Page-Loaded List Box
External - External Procedure

Template Registry Maintenance

The **Template Registry** dialog also provides other command buttons for other file maintenance options for the registry:

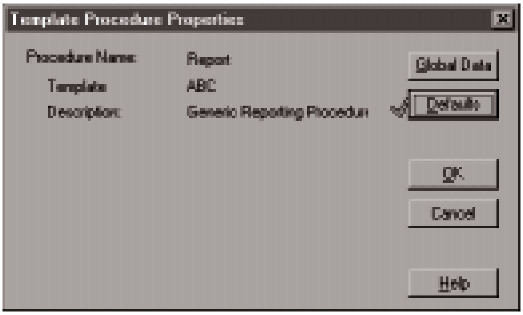
Unregister
This button deletes the highlighted template class from REGISTRY.TRF, making it unavailable for use in your applications.

Enable
This button enables the highlighted template class or template, making it available for use in your applications.

Disable
This button disables the highlighted template class or template, making it unavailable for use in your applications.

Properties
This button opens the **Template Procedure Properties** dialog to modify Procedure templates. Press the **Global Data** button to edit

default global data generated by this Procedure template. Press the **Defaults** button to edit default structures (windows, list boxes, etc.) contained in this template.



Template Procedure Properties

Procedure Name: Report
Template: ABC
Description: Generic Reporting Procedun

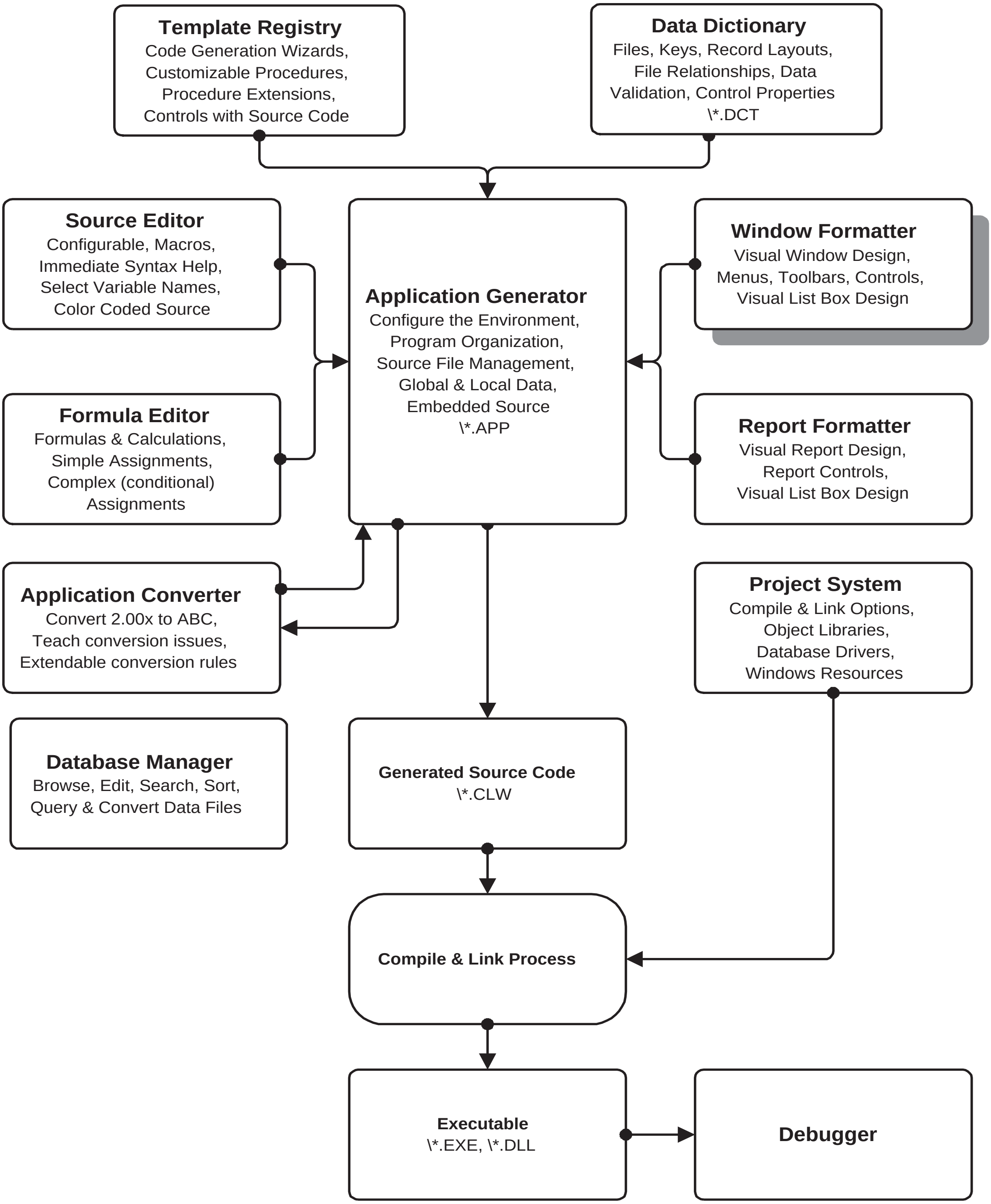
Edit Definition

This button opens the highlighted Template source file (.TPL or .TPW) with the Text Editor.

If the highlighted item in the Template Registry is a module, the Text Editor opens to the first line of template language code for the #MODULE. If the highlighted item in the Template Registry is a procedure, it opens to the first line of template language code for the #PROCEDURE. See the *Programmer's Guide* for more information on template language format and syntax.

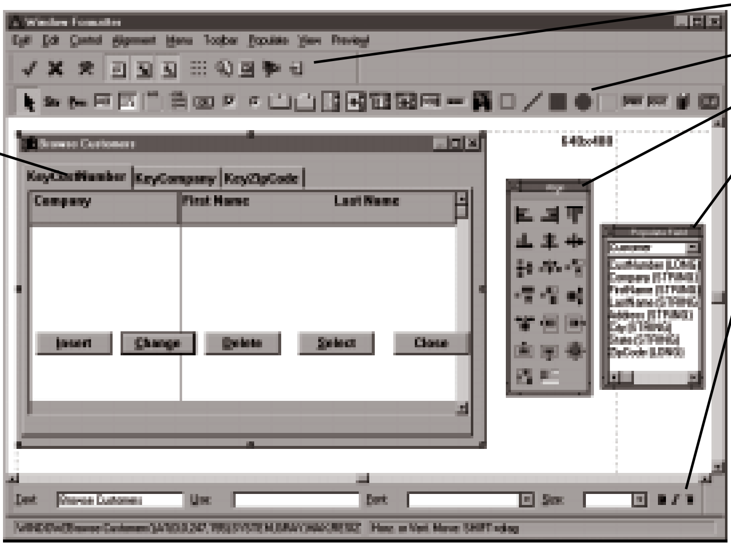
The next time you open an application, any changes to the Template source are registered or not, according to the Registry settings in the **Application Options** dialog.

4 - WINDOW FORMATTER



About This Chapter

Use the **Window Formatter** to visually design window elements—windows and controls—on screen. The **Window Formatter** generates the Clarion language source code that describes the window, then the Application Generator places the generated source code at the appropriate point in your application.



Text: Browse Customers

The Window Formatter displays a sample window showing the controls you place in it. You can resize or reposition any control by dragging its handles.

Commands

Controls

Alignment

Data Fields

Properties

This chapter:

- Tells you how to use the **Window Formatter** to create a new WINDOW structure or edit an existing one.
- Tells you how to use the **Window Formatter** to create a new TOOLBAR structure or edit an existing one.
- Tells you how to use the **Menu Editor** to create a new MENUBAR structure or edit an existing one.
- Tells you how to use the **Window Properties** dialog to set window properties.
- Tells you how to configure the **Window Formatter** to work the way you prefer.
- Tells you how to use the **List box Formatter** to format your LIST and COMBO controls.

Window Creation Overview

Most likely, your application will use a number of windows to display instructions, accept input, and provide data or other information to the user. In general, this is what you will do to put such a window on the screen:

1. Select or create the procedure that manages the window.

   See the *Application Generator* chapter for more information.

2. From the **Application Tree** dialog, RIGHT-CLICK the procedure name and select **Window** from the popup menu.

   If no default window is defined, select a window type from the **New Structure** dialog. See *Choosing a Window Type*.

   If a default window is already defined, the **Window Formatter** opens.

Tip: You can access the Window Formatter from the Text Editor! To create a new window from the Text Editor, place the cursor on a blank line, then choose Edit ➤ Format Structure or press CTRL+F. To edit an existing window, place the cursor anywhere within the source code structure that defines the window, then choose Edit ➤ Format Structure or press CTRL+F.

3. Customize the window by setting its size and properties.

   See *Defining Your Application's Windows*.

4. Optionally, place a menu in the window with the **Menu Editor**.

   See *Creating Your Application's Menus* for more information on this process.

5. Place *controls* in the window—these might include *entry boxes* for editing fields from the database, *command buttons* for initiating or cancelling actions, *text, strings,* or *prompts* containing instructions for the user, and other controls to enhance the appearance and ease of use of the window.

   See *Placing Controls in a Window*.

6. Set the *control* properties.

   ALT+DOUBLE-CLICK accesses the control properties. See the *Controls and Their Properties* chapter.

7. Return to the **Procedure Properties** dialog.

Choosing a Window Type

Clarion's Procedure templates usually provide an appropriate default window for you. So if you create your procedure with a code generation Wizard or with a Frame, Browse, Form, Viewer, or Splash Procedure template, then you need not choose a window type, although you can change the default if you want to.

However, if you use the Window - Generic Window Handler Procedure template, or if you start the **Window Formatter** from within the Text Editor (CTRL+F) the Application Generator opens the **New Structure** dialog so you can choose from a list of default window definitions. Following are some guidelines to help you choose the right window for the job at hand.

STARTing Modeless Windows (new thread)

When you START a procedure on its own thread, the procedure and its window operate independently of other threads in the same program; that is, the end user can switch focus between each execution thread at will. This is true regardless of whether the windows on each thread are MDI or non-MDI. These are "**modeless**" windows. See *START* in the *Language Reference*.

MDI and Non-MDI Windows (same thread)

If you start a procedure on an existing thread (call a procedure without START), program behavior depends on whether or not the procedure's window has the MDI attribute.

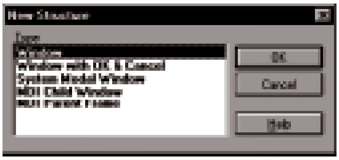
A non-MDI window on the same thread as its parent blocks access to its parent window, blocks access to all other threads in the program, and prevents subsequent opening of non-MDI windows on the same thread. This is an "**application modal**" window. When the application modal window closes, the other execution threads are available again.

An MDI window on the same thread as its parent blocks access only to its parent window. When the MDI child window closes, its parent window regains focus.

Default Window Structures

Some of the types of windows you can create with Clarion appear in the **New Structure** dialog. The items in the **New Structure** dialog represent Clarion language data structures.

You may see window structures, report structures, or both, depending on how you access the dialog. A window structure is a group of Clarion language statements that define all the attributes of a window. You may want to think of a window structure as the definition of the window. See *WINDOW* in the *Language Reference*.



This section discusses only the default window structures supplied with this release; however, you may modify these default windows, and you may even add your own default window structures by editing the \LIBSRC\DEFAULTS.CLW file. If you edit the DEFAULTS.CLW file, be sure to precede each new structure with the following line:

```
!!> title
```

where “title” is the structure name that appears in the **New Structure** dialog.

Following is a description of the default window structures provided with this package.

Window

To create a general purpose document window or dialog box, choose **Window** from the **New Structure** dialog. The **Window Formatter** generates a non-MDI WINDOW structure with no controls. That is, a bare or empty window.

This window accepts any controls (listboxes, entry boxes, buttons, etc.) you want to add. Because the window is non-MDI, it can move “outside” its application window. See *Windows Design Issues* for more information.

Window with OK & Cancel

This window is exactly like *Window* described above, except it contains **OK** and **Cancel** buttons. There are no actions associated with the buttons, you must add any needed functionality. If you want buttons with functionality already attached, see the *Application Handbook—Control Templates—CancelButton, CloseButton,* etc.

MDI Child Window

To create a document window which appears only inside an application frame, choose **MDI Child Window**. The **Window Formatter** generates a WINDOW structure with the MDI attribute.

The child window typically appears as a normal window, with frame, system menu, maximize and minimize buttons, and icon. The user should be able to manipulate it like any other window—except that the child window *cannot* move outside the main application window. See *Windows Design Issues* for more information.

All MDI windows must reside in separate procedures and execution threads from the APPLICATION window (see *MDI Parent Frame* below). This means you must initiate a thread (use START) when you start this window's procedure from the APPLICATION frame.

Tip: Any menus and toolbars you create for an MDI window will automatically merge with the APPLICATION's menu and toolbar when the MDI child is the active window!

MDI Parent Frame

To create the APPLICATION frame, or main window, for an MDI application, choose **MDI Parent Frame**. This provides the "outside" frame in which the MDI child windows appear. See *Windows Design Issues* for more information.

Tip: Typically, the APPLICATION window should have a resizable frame, plus a system menu, maximize and minimize buttons, and a menu. The File menu should provide commands to open the MDI child windows, and the Window menu should provide commands for managing the separate child windows. The Frame template provides all these features automatically!

The APPLICATION window may only have controls on its toolbar. MDI child windows contain all other controls in an MDI application. In other words, the APPLICATION window should hold only its child windows, and optionally, its toolbar.

The APPLICATION window and its MDI children must *not* reside in the same procedure. You must START the MDI child's procedure so that the MDI child window is in a separate thread from the APPLICATION. Multiple MDI windows may run in the same thread, but not on the same thread as the APPLICATION window.

System Modal Window

A system modal window prevents the user from doing anything else—even switching to another application—until the window closes. The **Window Formatter** generates a WINDOW structure with the MODAL attribute.

Tip*:* Only use a system modal window to signal a critical error. Unless your application has a very compelling reason to halt *all* system activity—for example, a severe file error which might result in lost data unless corrected at once—do not use this type of window.

Configuring the Window Formatter

The **Window Formatter Options** dialog sets the default position and size values applied when auto-populating controls, or when aligning controls with the alignment tools. To access the dialog, choose **Setup ➤ Window Formatter Options** from the environment menu, or choose **View ➤ Options** from the Window Formatter menu. The dialog is divided into four sections or tabs: Grid, Populate Defaults, Margin Defaults, and Spread Defaults.

The **Reset Default Values** button affects all four tabs.

Reset Default Values
Press this button to reset the values on all tabs to their Windows Standard values.

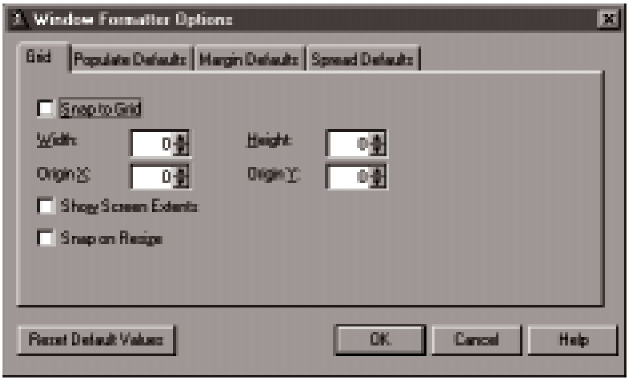
Grid

This tab turns “grid snap” on or off, and sets the starting point and offsets of the window grid. It also lets you show or hide the screen boundaries (extents) for the most common video resolutions (640x480, 800x600, 1024x768, etc.).

You can use the grid to force the boundaries of your window controls to fall only on certain x / y values (axes, latitude, longitude). By enforcing the grid axes, your controls are easier to position and align.

Snap to Grid
Check this box turn grid snap on; clear the box to turn it off. Grid snap displays a dot grid of valid positioning coordinates and forces the upper left corner of new controls to align with the dot grid. The end user does not see the grid at run time; it is a design tool only.

Tip: **You can also choose View ➤ Show Grid from the menu, or press the ⋮⋮⋮ button to toggle grid snap on and off.**



□ Snap to Grid

Width: 0 Height: 0

Origin X: 0 Origin Y: 0

□ Show Screen Extents

□ Snap on Resize

Width

Enter the horizontal distance between the grid dots (x axis). This is the minimum horizontal distance you can move a control when grid snap is on.

Height

Enter the vertical distance between the grid dots (y axis). This is the minimum vertical distance you can move a control when grid snap is on.

Origin X

Enter the horizontal coordinate at which to begin placing the grid dots. This is the left-most position at which controls will align or auto-populate when grid snap is on.

Origin Y

Enter the vertical coordinate at which to begin placing the grid dots. This is the top-most position at which controls will align or auto-populate when grid snap is on.

Show Screen Extents

Check this box to show video screen boundaries within the Window Formatter for the most common video resolutions. Clear the box to suppress the boundaries.

Snap on Resize

Check this box to force controls to snap to the nearest grid point grid when resizing from the right or bottom edges. This constrains a control's width and height to the grid. Resizing a control using the top and left edges always snaps to the grid.

Populate Defaults

This tab sets the default width and height for a variety of window controls. The Window Formatter applies the default sizes whenever you use it to add a control to the window.

Control Type

Choose the type of control for which to set the default size.

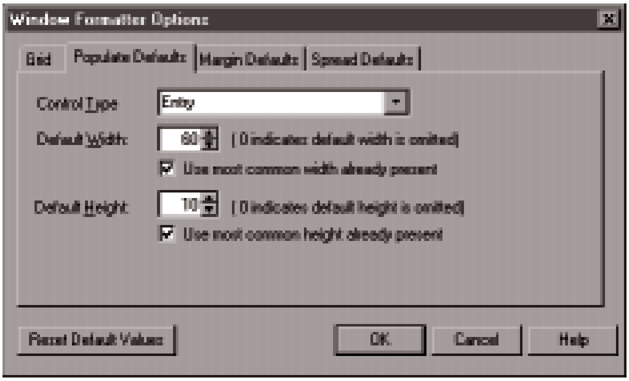
Tip: The default sizes are specified in dialog units—a unit of measure based on the current system font. See the Glossary for a complete definition.

Default Width

Set the default width for the specified control type. A value of zero (0) specifes no width—the control expands to the size of the data it displays. See *AT* in the *Language Reference* for more information.

Use most common width already present

Check this box to specify a dynamic default based on width of any controls of the same type that are already present on the window. For example, if there are three ENTRY controls and two of the controls are 50 units wide, then 50 becomes the default width for ENTRY controls. Clear this box to always apply the Default Width value, even if other controls of the same type are present.



Control Type: Entry

Default Width: 60 [0 indicates default width is omitted]

☑ Use most common width already present

Default Height: 10 [0 indicates default height is omitted]

☑ Use most common height already present

Default Height

Set the default height for the specified control type. A value of zero (0) specifes no height—the control expands to the size of the data it displays. See *AT* in the *Language Reference* for more information.

Use most common height already present

Check this box to specify a dynamic default based on height of any controls of the same type that are already present on the window. For example, if there are three ENTRY controls and two of the controls are 10 units wide, then 10 becomes the default height for ENTRY controls. Clear this box to always apply the Default Height value, even if other controls of the same type are present.

Margin Defaults

This tab sets the margins applied by the margin alignment tool. For more information on these alignment tools, see *Window Formatter Tools*. The margin is simply the distance between the closest edges of two controls (or of a control and the window). The Window Formatter applies the margins whenever you use the margin alignment tools.

Different types of controls require different margins to accomodate their unique characteristics. For example, TAB controls and GROUP controls need extra space to allow for their text.

Margin Relative to

Choose the type of control for which to set the margins. Choose from:

Other Set the default margins.

Group Set the margins to apply for controls inside a GROUP control, abutting the GROUP control.

Option Set the margins to apply for controls inside an OPTION control, abutting the OPTION control.

Tab Set the margins to apply for controls inside a TAB control, abutting the TAB control.

Top Margin

The distance between the top edge of the selected control and the nearest horizontal edge of a bounding control or window.

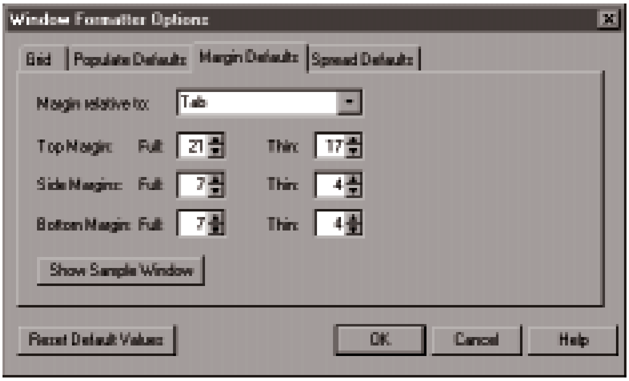
Side Margins

The distance between the vertical edges of the selected control and the nearest vertical edge of a bounding control or window.

Bottom Margin

The distance between the bottom edge of the selected control and the nearest horizontal edge of a bounding control or window.

Tip: The two settings (Full and Thin) provide alternative margins applied by the margin alignment tools. For more information on these alignment tools, see *Window Formatter Tools*.



Margin relative to: Tab

Top Margin: Full: 21 Thin: 17

Side Margins: Full: 7 Thin: 4

Bottom Margin: Full: 7 Thin: 4

Tip: The default sizes are specified in dialog units—a unit of measure based on the current system font. See the Glossary for a complete definition.

Show Sample Window

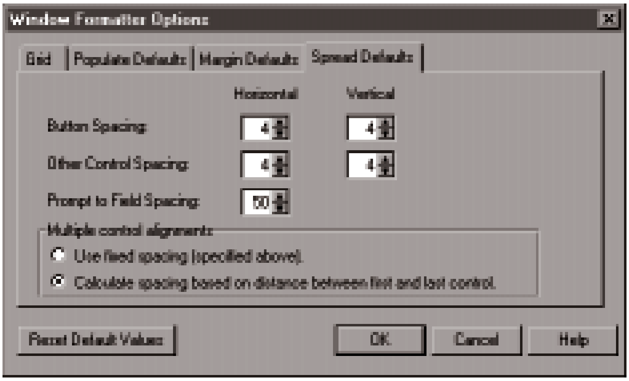
Press this button to see a sample window that applies the current margin settings.

Spread Defaults

This tab sets the default spacing between auto-populated window controls and between controls positioned by the Spread Alignment tools. The Window Formatter applies the default spacing when you auto-populate fields from the Fields Toolbox and when you use the Spread Alignment tools. For more information on these alignment tools, see *Window Formatter Tools*.

Button Spacing

Set the default distance between the edges of a button control and the nearest control.



	Horizontal	Vertical
Button Spacing:	4	4
Other Control Spacing:	4	4
Prompt to Field Spacing:	50	

Multiple control alignments:
○ Use fixed spacing (specified above).
● Calculate spacing based on distance between first and last control.

Other Control Spacing

Set the default distance between the edges of two adjacent controls.

Prompt to Field Spacing

Set the default distance between the right edge of a PROMPT control and the left edge of its (visually) associated (ENTRY, SPIN, TEXT, etc.) control.

Tip: The default sizes are specified in dialog units—a unit of measure based on the current system font. See the Glossary for a complete definition.

Multiple control alignments

This selection applies only to the Spread Alignment tools. For more information on these alignment tools, see *Window Formatter Tools*.

Use fixed spacing

Apply the static values specified above when spacing (spreading) multiple controls.

Calculate spacing

Calculate spacing based on distance between first and last control, so there is an equal distance between each control.

Using the Window Formatter

Choosing the window type is just the beginning. The **Window Formatter** provides a rich assortment of visual tools and menus to help you create and edit your window.

The **Window Formatter** lets you directly manipulate the window and the controls inside it. The sample window, for example, contains 'handles'—tiny boxes located at the corners and sides of the window. By selecting a handle and dragging the mouse, you may resize the sample window. The window the user sees when your application runs is the same size as the window you create by dragging.

When the **Window Formatter** generates the source code for the window, it places the data determining the size and position of the window (as you specified by dragging the mouse) in the AT attribute of the statement declaring the window.

Similarly, the **Window Formatter** supplies the other attributes by presenting you with options, check boxes and fields in which you specify your design preferences.

Typical Window Design Process

Here is the typical process for customizing a new window with the **Window Formatter**:

1. Set the size of the window by dragging its handles so that the sample window is the size you wish.
2. Set other window attributes by using the **Window Properties** dialog.

   RIGHT-CLICK the window then choose **Properties** from the popup menu, or select the window then choose **Edit ➤ Properties.**

   Window attributes include the window caption, whether the window is resizable, whether the window is scrollable, icons associated with the window, messages, help files, and cursor types associated with the window, and many others. See *Window Properties Dialog*.
3. Close the **Window Properties** dialog.
4. Place controls in the window.

   See *Placing Controls in a Window*. Also see *Controls and Their Properties*.
5. Preview the window by choosing **Preview!** from the menu. Press ESC to return to the **Window Formatter**, then make any additional adjustments.
6. Choose **Exit!** to return to the Application Generator or Text Editor.

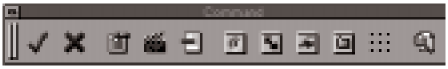
Window Formatter Tools

Sample Window

The **Window Formatter** is a *visual* design tool. You always *see* a sample of the window you're working on, *as* you work on it. In addition, you can see the window, *exactly* as it will appear to the end user by choosing **Preview!** from the menu.

Command Toolbox

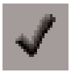
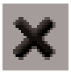
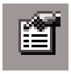
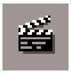
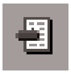
The **Window Formatter** contains a dockable **Command** toolbox. The toolbox lets you quickly execute a variety of **Window Formatter** functions at the touch of a button.



All the commands in the **Command** toolbox are also available from the menu (**Exit!, Edit, View, Preview!**).

Display or hide the **Command** toolbox by choosing **View ➤ Show Commandbox.** Resize the **Command** toolbox by placing the cursor on the border of the box. When the cursor changes to a double headed arrow, CLICK and DRAG. Dock the toolbox by dragging the handle (double verticle lines) to any edge of the Window Formatter frame (dragging the titlebar repositions the toolbox *without* docking).

> **Tip:** **Position the cursor over any tool and wait for half a second. A tool tip shows you the type of control this tool creates.**

	Exit the Window Formatter and save changes.
	Exit the Window Formatter and abandon changes.
	Edit the properties of the selected (red handles) control or window.
	Edit the actions of the selected (red handles) control or window.
	Edit the embedded source for the selected (red handles) control or window.

Hide or display the Controls Toolbox.

Hide or display the Align Toolbox.

Hide or display the Property Toolbox.

Hide or display the Fields Toolbox.

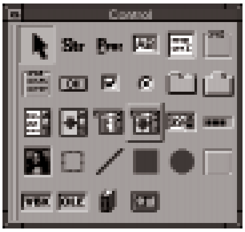
Toggle the alignment grid on or off.

Preview the window.

Controls Toolbox

The **Window Formatter** contains a dockable **Controls** toolbox, similar to those found in many draw or paintbrush programs. Simply choose a control from the toolbox (CLICK on it), then CLICK in the sample window to place the control in the window. By default, the control takes on the size of the other controls of that type already in the window. If there are no like controls in the window, the control is the default size.

> **Tip:** **You can control the initial height and width of a control with the Window Formatter Options dialog. See the *Configuring the Window Formatter* for more information.**



All the controls in the toolbox are also available from the **Controls** menu with the exception of the Control Template and the Dictionary Field, which are available from the **Populate** menu.

Display or hide the **Controls** toolbox by choosing **View ➤ Show Toolbox** or press . Resize the **Controls** toolbox by placing the cursor on the border of the box. When the cursor changes to a double headed arrow, CLICK and DRAG. Dock the toolbox by dragging the handle (double verticle lines) to any edge of the Window Formatter frame (dragging the titlebar repositions the toolbox *without* docking).

Tip: Position the cursor over any tool and wait for half a second. A tool tip shows you the type of control this tool creates.

Places a Control Template on the window under construction. See the *Application Handbook—Control Templates* chapter.

Tip: Generally, it is to your advantage to use a Control template rather than a simple control.

Control templates generate source code to declare controls *and* manage their associated data. For example, the BrowseBox Control template not only generates source code to display a listbox, it also generates code to load data from a file into a QUEUE, then display the data in the listbox with complete scrolling and mouse-click selection capability.

Drops the selected control tool.

Places a STRING control on the window under construction. See *Controls and Their Properties—String Properties*.

Places a PROMPT control on the window under construction. See *Controls and Their Properties—Prompt Properties*.

Places an ENTRY control on the window under construction. See *Controls and Their Properties—Entry Properties*.

Places a TEXT control on the window under construction. See *Controls and Their Properties—Text Properties*.

Places a GROUP control (group box) on the window under construction. See *Controls and Their Properties—Group Box Properties*.

Places an OPTION control (OPTION structure, which appears as a group box with radio buttons) on the window under construction. See *Controls and Their Properties—Radio Button Properties*.

Places a BUTTON control on the window under construction. See *Controls and Their Properties—Button Properties*.

Places a CHECK control on the window under construction. See *Controls and Their Properties—Check Box Properties*.

Places a RADIO control on the window under construction. See *Controls and Their Properties—Radio Button Properties*.

Places a SHEET control on the window under construction. Sheet controls contain Tab controls. See *Controls and Their Properties—Property Sheet Properties*.

Places a TAB control on the window under construction. Tab controls may contain any other control types. See *Controls and Their Properties—Tab Properties*.

Places a LIST control (listbox, or drop-down listbox) on the window under construction. See *Controls and Their Properties—Creating Listboxes*.

Places a Drop LIST control on the window under construction. See *Controls and Their Properties—Creating Listboxes*.

Places a COMBO control (combo box, or drop combo box) on the window under construction. See *Controls and Their Properties—Combo Box Properties*.

Places a Drop COMBO control on the window under construction. See *Controls and Their Properties—Combo Box Properties*.

Places a SPIN control on the window under construction. See *Controls and Their Properties—Spin Box Properties*.

Places a PROGRESS control on the window under construction. See *Controls and Their Properties—Progress Bar Properties*.

Places an IMAGE control (graphic image) on the window under construction. See *Controls and Their Properties—Image Properties*.

Places a REGION control on the window under construction. See *Controls and Their Properties—Region Properties*.

Places a LINE control on the window under construction. See *Controls and Their Properties—Line Properties*.

Places a BOX control on the window under construction. See *Controls and Their Properties—Box Properties*.

Places an ELLIPSE control on the window under construction. See *Controls and Their Properties—Ellipse Properties*.

Places a PANEL control on the window under construction. See *Controls and Their Properties—Panel Properties*.

Places a VBX control (Visual Basic) on the window under construction. See *Custom Controls—VBX Control Properties*.

Places a OLE/OCX Container control on the window under construction. See *Custom Controls—OLE Control Properties*. Available only in Professional and Enterprise Editions.

Lets you select a field defined in the Data Dictionary, and places the control specified in the data dictionary, plus an associated PROMPT, on the window under construction.

Display or hide the **Controls** toolbox by choosing **Options ➤ Toolbox.** All the controls in the toolbox are also available from the **Controls** menu. See *Placing Controls in a Window*. Also see *Controls and Their Properties*.

Populate Field Toolbox

The **Window Formatter** contains a floating **Populate Field** toolbox. This toolbox lets you quickly "populate" a window with controls for data dictionary fields and memory variables. First, choose a file or variable scope from the drop-down list, then DOUBLE-CLICK the field or variable you want to show on the window. This places a prompt and a control for the selected field. The type of control (entry box, check box, radio button, etc.) is determined by the settings for this particular field in the Data Dictionary. The field is automatically aligned.

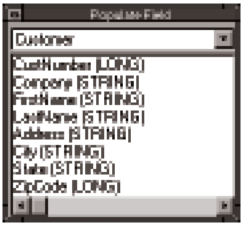
If the Data Dictionary specifies no size for the control, it takes on the size of the other controls of that type already in the window. If there are no like controls in the window, the control is the default size.

Tip: **You can control the initial height and width of a control with the Window Formatter Options dialog. See the *Configuring the Window Formatter* for more information.**

Tip: **Select a control, then press a directional arrow key to nudge the control's position one dialog unit at a time.**

Select a control, then press and hold the SHIFT key while pressing a directional arrow key to nudge the control's size one dialog unit at a time.

Display or hide the **Populate Field** toolbox by choosing **View ➤ Show Fieldbox** or press ▣. Resize the **Populate Field** toolbox by placing the cursor on the border of the box. When the cursor changes to a double headed arrow, CLICK and DRAG.

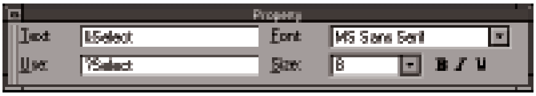


CustNumber (LONG)
Company (STRING)
FirstName (STRING)
LastName (STRING)
Address (STRING)
City (STRING)
State (STRING)
ZipCode (LONG)

You may also populate a window with entry controls for fields in your data files by using the **Populate** menu, or by using the Dictionary Fields tool in the **Controls** toolbox.

Property Toolbox

The **Window Formatter's Property** toolbox lets you quickly specify the appearance and content of the *text* on each control within the window and on the window title bar. Control the font, size, style, and content of all your text, using standard word processor buttons and drop-down lists.



Text: &Select

Use: ?Select

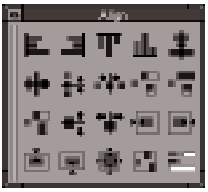
Font: MS Sans Serif

Size: 8

Display or hide the **Property** toolbox by choosing **View ➤ Show Propertybox** or press **.** Resize the **Property** toolbox by placing the cursor on the border of the box. When the cursor changes to a double headed arrow, CLICK and DRAG. Dock the toolbox by dragging the handle (double verticle lines) to any edge of the Window Formatter frame (dragging the titlebar repositions the toolbox *without* docking).

Align Toolbox

The **Window Formatter's Align** toolbox lets you quickly, professionally, and precisely align the controls in your window. Select the controls to align (CTRL+CLICK lets you select multiple controls, or you can "lasso" multiple controls with CTRL+DRAG), then click on the appropriate alignment tool. All the toolbox alignment actions (and more) are also available from the **Align** menu.



Display or hide the **Align** toolbox by choosing **View ➤ Show Alignbox** or press **.** Resize the **Align** toolbox by placing the cursor on the border of the box. When the cursor changes to a double headed arrow, CLICK and DRAG. Dock the toolbox by dragging the handle (double verticle lines) to any edge of the Window Formatter frame (dragging the titlebar repositions the toolbox *without* docking).

> **Tip:** **For most alignment functions, the first control(s) selected (blue handles) are aligned with the *last* control selected (red handles). That is, the last control selected is the anchor control. It doesn't move, the others do.**

Align Left: Aligns the left borders of the selected controls with the left border of the last control selected (red handles).

Align Right: Aligns the right borders of the selected controls with the right border of the last control selected (red handles).

Align Top: Aligns the top borders of the selected controls with the top border of the last control selected (red handles).

Align Bottom: Aligns the bottom borders of the selected controls with the bottom border of the last control selected (red handles).

Align Vertically: Along a vertical axis, aligns the centers of the selected controls with the center of the last control selected (red handles).

Align Horizontally: Along a horizontal axis, aligns the centers of the selected controls with the center of the last control selected (red handles).

Spread Vertically: Equalizes the vertical space between the selected controls.

Spread Horizontally: Equalizes the horizontal space between the selected controls.

Same Size: Makes all selected controls the same height and width as the last control selected (red handles).

Same Height: Makes all selected controls the same height as the last control selected (red handles).

Same Width: Makes all selected controls the same width as the last control selected (red handles).

Center Vertically: As a group (relative positions of selected controls don't change), centers the selected controls vertically within the window.

Tip: **Position the cursor over any tool and wait for half a second. A tool tip shows you the type of alignment this tool accomplishes.**

Center Horizontally: As a group (relative positions of selected controls don’t change), centers the selected controls horizontally within the window.

Set Left Margin: Repositions (or SHIFT+▣ resizes) the selected controls so the left edge of each control is a specified distance from the neareast bounding control or window. The margin distance toggles between two standard left margins. See *Configuring the Window Formatter—Margin Defaults*.

Set Right Margin: Repositions (or SHIFT+▣ resizes) the selected controls so the right edge of each control is a specified distance from the neareast bounding control or window. The margin distance toggles between two standard margins. See *Configuring the Window Formatter—Margin Defaults*.

Set Top Margin: Repositions (or SHIFT+▣ resizes) the selected controls so the top edge of each control is a specified distance from the neareast bounding control or window. The margin distance toggles between two standard top margins. See *Configuring the Window Formatter—Margin Defaults*.

Set Bottom Margin: Repositions (or SHIFT+▣ resizes) the selected controls so the bottom edge of each control is a specified distance from the neareast bounding control or window. The margin distance toggles between two standard margins. See *Configuring the Window Formatter—Margin Defaults*.

Set All Margins: Repositions (or SHIFT+▣ resizes) the selected controls so all edges of each control are a specified distance from the neareast bounding control or window. The margin distance toggles between two standard margins. See *Configuring the Window Formatter—Margin Defaults*.

Tip: Use the Set All Margins tool to position and size SHEET, GROUP, and LIST controls within their respective containers.

Shrink Wrap: If only a container control is selected, this repositions and resizes the container to the minimum dimensions necessary to surround its containees. If multiple controls are selected, this creates a new UBOXED (hidden) GROUP control minimally surrounding the selected controls. This lets you position the contained controls with the Margin Alignment tools.

Align Prompts: For selected prompt/field pairs, this aligns the prompt with its (visually) associated (ENTRY, SPIN, COMBO, TEXT, etc.) field.

Constrained Dragging

Press and hold the SHIFT key while dragging a control to limit the control's movement to a single axis. That is, SHIFT + DRAG moves a control either horizontally or vertically, but not both.

Fine Sizing and Positioning

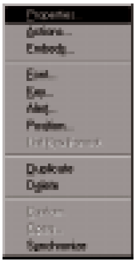
Select a control, then press a directional arrow key to nudge the control's position one dialog unit at a time.

Select a control, then press and hold the SHIFT key while pressing a directional arrow key to nudge the control's size one dialog unit at a time.

Window Formatter Menus

Popup Menu

Access the popup menu by RIGHT-CLICKING a window or a control. The popup menu on the **Window Formatter** lets you manipulate and customize the window *and* the controls on the window, depending on whether the window or a control is selected.



❏ To select a *window*, place the cursor in the sample window title bar and RIGHT-CLICK.

- ❑ To select a *control*, place the cursor on the control and RIGHT-CLICK.
- ❑ To select a *property sheet* control, place the cursor anywhere on the sheet, but *not* on other controls, and *not* on a tab, then RIGHT-CLICK.
- ❑ To select a *tab* control, place the cursor on the corresponding tab and RIGHT-CLICK.

Tip: All of the popup menu commands, and more, are available on the Window Formatter Edit menu.

Following is a description of the popup menu choices.

Properties

To edit control or window properties, RIGHT-CLICK a control or the window, then choose **Properties**. See *Window Properties Dialog*, or see *Controls and Their Properties*.

Actions

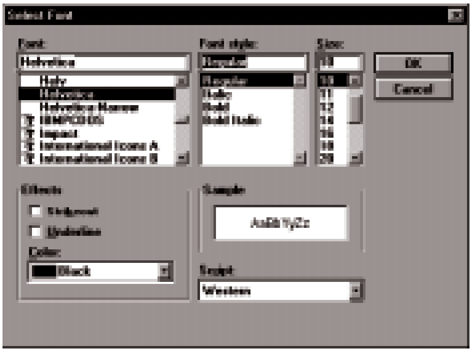
To answer the template prompts associated with a control or the window, RIGHT-CLICK it then choose **Actions**. See the *Controls and Their Properties* chapter for more information.

Embeds

To add or edit embedded source associated with a control or window, RIGHT-CLICK it then choose **Embeds**. See *Application Generator—Embedded Source Code*.

Font

To control the appearance of the text displayed in a control or window, RIGHT-CLICK the control or window then choose **Font.** Specify font, size, style, script, and color from drop-down listboxes. Toggle Strikeout and Underline on and off with check boxes. The **Select Font** dialog shows you a sample of the text design you have chosen.

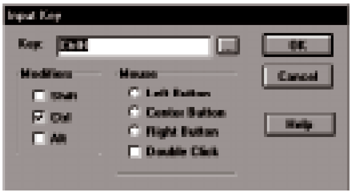

Select Font

Font: Helvetica
Font style: Regular
Size: 10
Effects
□ Strikeout
□ Underline
Color: Black
Sample: AaBbYyZz
Script: Western

Key

To specify a "hot" key for a control, RIGHT-CLICK the control then choose **Key** (the KEY attribute is not applicable to windows, nor to some controls). Use the **Input Key** dialog to add the KEY attribute to your control. The KEY attribute specifies a "hot" key, or key combination, which, when pressed by the user, gives immediate focus to the control, or, for an action control such as a command button, initiates the action.

From the **Input Key** dialog, specify the hot key or key combination by pressing the desired key or key combination. The keys you press appear in the **Key** field, and are supplied as parameters to the KEY attribute for this control. See *KEY* in the *Language Reference*.



Input Key

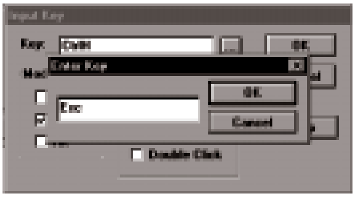
Key:

Modifiers
□ Shift
☑ Ctrl
□ Alt

Mouse
○ Left Button
○ Center Button
○ Right Button
□ Double Click

Mouse clicks may be used as hot keys; however, mouse clicks *cannot* be specified by clicking the mouse. For mouse clicks, check the corresponding check box(es). For example, to give focus to a control when the user double-clicks, check the **Left Button** box *and* the **Double Click** box.

Optionally, add a modifier or modifiers to create a multiple-key hot key sequence (for example, CTRL+H, or ALT+RIGHT-CLICK), by checking **Ctrl**, **Alt,** or **Shift**, or any combination of the three.



Input Key

Key: CtrlH

Enter Key

Esc

□ Double Click

The ESC, ENTER, and TAB keys *cannot* be specified by pressing them. For these keys, press the ellipsis (...) button then type "esc," "enter," or "tab."

The following controls receive focus from the KEY attribute:

Combo Box
Entry Field
Group Box
Listbox
Option Box
Prompt

Property Sheet
Spin Box
Tab
Text Field

The following controls both receive focus and immediate execution from the KEY attribute:

Button
Check Box
Radio Button
OLE Control
VBX Control

The KEY attribute is not applicable to the following controls:

String
Progress Bar
Image
Region
Line
Box
Ellipse
Panel

Alert

To specify an Alert key for a window or a control, select the window or control then choose **Alert**. Use the **Alert Keys** dialog and the **Input Key** dialog to add the ALRT attribute to your window or control. When the ALRT attribute is set, the window generates an EVENT:AlertKey if the user presses the key(s) you specify in these dialogs. You may specify more than one Alert key for a window or a control.

See *Key* above for a discussion on how to specify keys using the **Input Key** dialog.

Position

To specify the position of a control or a window, select it then choose **Position**. See *Position* in *Window Properties Dialog* section for a discussion of positioning windows.

To position controls, you will normally CLICK and DRAG the controls and use the **Align** tools, or both. However, you may use the **Position** tab of the various control properties dialogs. See *Controls and Their Properties* for more information. Also see *Grid Settings* in *The Options Menu* section.

Listbox Format

To specify the appearance and functionality of a listbox control, select the listbox then choose **Listbox Format.** See *List box Formatter* for more information.

Duplicate

To place a copy of a control in the same window, select the original then choose **Duplicate**. The copy appears beside the original.

Tip: You may duplicate *multiple* controls by selecting multiple controls before invoking the Duplicate command. Use CTRL+CLICK to select multiple controls, or lasso multiple controls using CTRL+CLICK+DRAG.

Delete

To delete a control, select it then choose **Delete**, or select it then press the DELETE key.

Custom

To open the Property Sheet for an OCX associated with an OLE control, select the control and choose the **Custom** command.

Open

To open the OLE Server associated with an OLE control, select the control and choose the **Open** command.

Synchronize

Applies the control attributes specified in the Data Dictionary to the selected control, or if the window is selected, to all the controls in the window. The attributes are applied as specified in the **Synchronization** tab of the **Application Options** dialog. See *Application Generator—Configuring the Application Generator*.

Edit Menu

The **Edit** menu in the **Window Formatter** lets you manipulate and customize the window, *and* the controls in the window, depending on whether the window or a control is selected.

❏ To select a *window*, place the cursor in the sample window title bar and CLICK.

❏ To select a *control*, place the cursor on the control and CLICK.

❏ To select a **Property Sheet** control, place the cursor anywhere on the sheet, but *not* on other controls, and *not* on a tab, then CLICK.

❏ To select a **Tab** control, place the cursor on the corresponding tab then CLICK.

Tip: Many of the Edit menu commands are also available on the popup menu that you access by RIGHT-CLICKING on the control or the window.

Following is a description of the **Edit** menu choices *not* described in the *Popup Menu* section:

Undo

To reverse the last editing action, choose **Undo**. All **Window Formatter** actions may be reversed, except deleting a control.

Tip: To undo several actions, including deleting controls, Exit the Window Formatter, but do not save the changes.

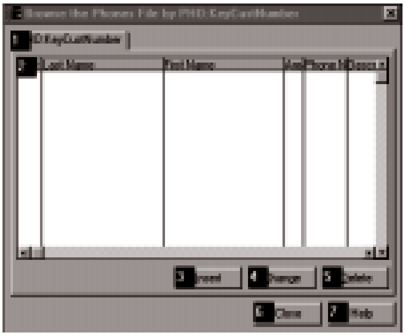
Redo

To redo the undone action, choose **Redo**. Not all actions may be redone.

Set Tab Order

To visually set the tab key order for selected controls, select a window, a Group Box, or an Option Box, then choose **Set Tab Order**. This opens the **Ordering Type** dialog, which lets you specify *how* the tab-stop order is set: Automatically or Manually, Horizontally or Vertically. See *Set Control Order* for an alternative method of setting tab key order.

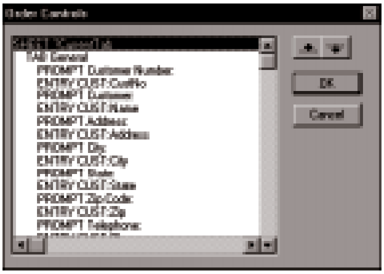
From the **Ordering Type** dialog, select the **Manual** radio button, then press the **OK** button to specify the tab-stop order by manually CLICKING on the controls. A number appears on each control, indicating the current order. CLICK on the controls to change the order to the order you wish.



Alternatively, from the **Ordering Type** dialog, select the **Automatic** radio button, then choose either the **Horizontally** or **Vertically** radio button. Press the **OK** button to automatically set the tab-stop order based on the position of the controls. **Horizontally** numbers the topmost controls first. **Vertically** numbers the leftmost controls first. Reselect the **Set Tab Order** command, or CLICK on the sample window title bar to return to normal editing mode.

Set Control Order

To set the tab key order, and move controls among overlapping tab controls, choose **Set Control Order**. This opens the **Order Controls** dialog, which displays all controls on the window in a hierarchical list. Reorder the controls and their tab key order by selecting a control then pressing the ▲ or ▼ button to move the control up or down within the list.



Order Controls

[illegible]
TAB General
PROMPT Customer Number:
ENTRY CUST:CustNo
PROMPT Customer:
ENTRY CUST:Name
PROMPT Address:
ENTRY CUST:Address
PROMPT City:
ENTRY CUST:City
PROMPT State:
ENTRY CUST:State
PROMPT ZipCode:
ENTRY CUST:Zip
PROMPT Telephone:

Synchronize Window

To apply the control attributes specified in the Data Dictionary to all the controls in the window, choose **Synchronize Window**. The attributes are applied as specified in the **Synchronization** tab of the **Application Options** dialog. See *Application Generator—Configuring the Application Generator.*

Control Templates

To add Extension templates to the procedure or to edit the Control and Extension template prompts for the procedure, choose **Control Templates**. This opens the **Extension and Control Templates** dialog where you can add Extension templates and you can edit both Control and Extension template prompts. See *Code and Extension Templates* in the *Application Handbook* for more information.

Control Menu

The **Control** menu lists all the controls that appear in the **Controls** toolbox, except Control Template and Dictionary Fields (see *Populate Menu*).

Executing a command from the **Control** menu is identical to clicking on the corresponding toolbox icon. The menu serves as a convenience. For a list of toolbox controls, see the *Window Formatter Tools*. Also see *Controls and Their Properties*.

Alignment Menu

The **Alignment** menu lists the same Alignment tools that appear in the Align Toolbox. Executing a command from the **Alignment** menu is identical to clicking on the corresponding toolbox icon. The menu provides the following additional options.

Snap All To Grid

Snaps all controls to the nearest grid coordinate.

Snap To Grid

Snaps the selected controls to the nearest grid coordinate.

For a list of Alignment tools, see *Window Formatter Tools*.

Menu Menu

The **Menu** menu lets you add, change, or delete menus on your window or on an OLE control.

When you specify a menu for your application window or for MDI child windows, Clarion automatically merges the application menu with the MDI child menu when the MDI child window has focus. This saves you the trouble of enabling, disabling, inserting and replacing various menu selections depending on which window has focus.

See the *The Menu Editor* and *Toolbars* below for directions on how to create menus and toolbars for your application.

Toolbar Menu

The **Toolbar** menu lets you add or delete a toolbar for your window.

Specify a toolbar for your application window, or for MDI child windows. Clarion automatically merges the application window toolbar with the MDI child toolbar when an MDI child window has focus. This saves you the trouble of enabling, disabling, inserting and replacing various tools depending on which window has focus.

See the *The Menu Editor* and *Toolbars* below for directions on how to create menus and toolbars for your application.

Populate Menu

The **Populate Menu** appears in the **Window Formatter** only when the Application Generator is active. It places a field or memory variable in the window, along with an appropriate control. For *fields*, the control *type* depends on how the field is defined in the data dictionary.

When active, two new tool icons appear at the bottom of the Controls toolbox, corresponding to the following commands:

Field

Places a control tied to a field or variable. When you CLICK in the window, the **File Schematic Definition** dialog appears. Select a field or variable, then CLICK in the window. This is equivalent to the Dictionary Fields tool in the Controls toolbox.

The **Window Formatter** places the control specified on the **Window** tab of the **Field Properties** dialog. If the Data Dictionary specifies no size for the control, it takes on the size of the other controls of that type already in the window. If there are no like controls in the window, the control is the default size.

Tip: **You can control the initial height and width of a control with the Window Formatter Options dialog. See the *Configuring the Window Formatter* for more information.**

Multiple Fields

Places a control tied to a field or variable. When you CLICK in the window, the **File Schematic Definition** dialog appears. Select a field or variable, then CLICK in the window. This is equivalent to the Dictionary Fields tool in the Controls toolbox.

The **Window Formatter** places the control specified on the **Window** tab of the **Field Properties** dialog.

After placing the first field, the **File Schematic Definition** dialog appears again, ready for you to place another field. When all fields are placed, press the **Cancel** button to return to normal editing.

Control Template

Lets you add a control template to the window under construction. Select one from the **Select a Control Template** dialog.

Control templates generate source code to declare controls *and* manage their associated data. For example, the Browse Box control template places a listbox in the window, lets you choose the fields for the list, and adds all the executable code for managing the listbox (loading it, scrolling it, etc.).

Once the control template is placed, you can specify its properties and actions by RIGHT-CLICKING and selecting **Properties** or **Actions** from the popup menu. See *Control Templates* in the *Application Handbook* for more information.

View Menu

The **View** menu lets you display and hide the **Window Formatter** grid tools and toolboxes.

Options

Open the **Window Formatter Options** dialog to set default spacing and control population coordinates. See *Configuring the Window Formatter*.

Show Grid

Toggle automatic grid snap on or off.

Tip: **Aligning the controls in your dialog boxes and windows gives your application a professional look. See Windows Design Issues for specific suggestions on how to align different types of controls.**

To set the width and height spacing between the grid dots, enter values in the **Width** and **Height** fields in the **Grid Settings** dialog. The values are in dialog units. See the Glossary for a definition of dialog units.

Show Commandbox

Toggle the Command toolbox on and off. When designing large windows, it may be useful to hide the toolbox, gaining additional room for the window. You may still access all the commands by choosing them from the **Exit!, Edit, View,** and **Preview!** menus.

Show Toolbox

Toggle the Controls toolbox on and off. When designing large windows, it may be useful to hide the toolbox, gaining additional room for the window. You may still access all the control tools by choosing them from the **Control** and **Populate** menus.

Show Alignbox

Toggle the Align toolbox on and off. This is a matter of individual preference. You may still access all the alignment commands by choosing them from the **Alignment** menu.

Show Propertybox

Toggle the Property toolbox on and off. This is a matter of individual preference. You may still eidt control properties with the **Command** toolbox or the **Edit** menu.

Show Fieldbox

Toggle the Populate Field toolbox on and off. This is a matter of individual preference. You may still populate fields using the **Controls** toolbox or the **Populate** menu.

VBX Custom Control Registry

Add a custom control library to the registry. Press the **Add** button in the **VBX Custom Control Registry** dialog, then DOUBLE-CLICK on the .VBX file name. This lets the **Window Formatter** place controls from the VBX library in your window (see *Custom Controls*). To remove a registration, press the **Remove** button.

Preview!

Display the window as your end user sees it at runtime. The only difference is, the window won't contain live data and the command buttons won't execute commands. To exit **Preview!** mode, press ESC.

Tip: **You should always test your windows and dialog boxes. Though the Window Formatter is visual, it does not show you how 3-D shading will affect the 'look' of your window, nor does it actually 'hide' a hidden control. Additionally, you may test the tab order while in Preview! mode to verify the current order makes sense.**

Window Properties Dialog

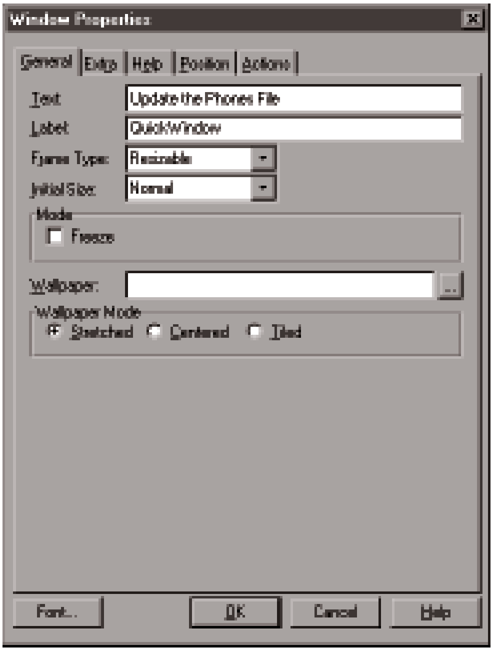
Use the **Window Properties** dialog to set *all* the properties or attributes of a WINDOW or APPLICATION. Properties include the window caption, whether the window is resizable, whether the window is scrollable, icons associated with the window, messages, help files, and cursor types associated with the window, and many others. In short, all the properties associated with windows as opposed to properties associated with procedures, controls, fields, etc.

To open the **Window Properties** dialog from the **Window Formatter**:

- ❏ RIGHT-CLICK on the sample window then choose **Properties** from the popup menu.
- ❏ Select the sample window then press ENTER.
- ❏ Select the sample window then choose **Edit ➤ Properties** from the menu.

Additionally, selecting a window from the **New Structure** dialog leads to the **Window Properties** dialog.

General Tab



Text: Update the Phones File
Label: QuickWindow
Frame Type: Resizable
Initial Size: Normal
Mode
☐ Freeze
Wallpaper:
Wallpaper Mode
● Stretched ○ Centered ○ Tiled

Text

To specify caption bar text for your window, type a string constant.

Tip: If you create a system modal window, leave the caption bar blank. The normal Windows style for this type of window is to display the window without a caption bar.

Tip: You may dynamically alter the caption bar text at run-time. See PROP:Text in the *Language Reference*.

Label

Specify the WINDOW label. The label is used to refer to the WINDOW in source code. In the following example "CustEntry" is the label for the CustEntry window:

```
CustEntry WINDOW...   !defines CustEntry window
          END
  CODE
  OPEN(CustEntry)     !opens CustEntry window
```

The label may contain upper or lower case letters, numbers, the underscore character, or a colon. *Spaces are forbidden*. The first character must be a letter or the underscore character. Clarion reserved words may not serve as labels.

Frame Type

Pick the frame type for your window from the drop-down list. The frame type defines the borders of the window. Choose from:

Single A single pixel frame which the user *cannot* resize. Most suitable for dialog boxes.

Double A thick frame, which the user *cannot* resize. Use this type frame for a system modal window (without a caption bar), or for a modal dialog box (with a caption bar).

Resizable A thick frame, which the user *may* resize. Choose this for application and MDI child windows.

None A single pixel frame under Windows 95, and no frame under Windows 3.1. Most suitable for dialog boxes. The user cannot resize this frame.

Initial Size

Select the initial size and state of your window from the drop-down list. Choose from:

Normal Display the window at the default size. If you don't specify a default size, Clarion's run-time library sets it for you.

Maximized The window fills the entire desktop, or the entire application frame, depending on whether the window is an application window, or an MDI child window.

Iconized In Windows 3.1, the window appears in an iconized state—as a 32 by 32 pixel window at the bottom of the desktop (application window) or at the inside bottom of the application frame (MDI child window).

In Windows 95, the window appears in an iconized state in the Taskbar.

Tip: If you choose the *iconized* selection, be sure to specify a file name in the Icon field. If not, your window may not receive a Restore command on its system menu, which means it will always remain iconized. Specifying a file name also adds a minimize button to the window, allowing the user to iconize the window again, after restoring it.

Freeze

To "freeze" all the controls on the window so that subsequent data dictionary changes are not applied, check this box. You can override the #Freeze attribute for all controls or for individual controls. See *Application Generator—Configuring the Application Generator*.

Wallpaper

To provide a background image for the window's client area, specify an image filename. Type the filename or press the ellipsis button (...) to select a file. See *WALLPAPER* in the *Language Reference*.

Wallpaper Mode

Specify how the window displays the background image. Choose from:

Stretched The image expands to fill the entire client area.

Centered The image displays at its default size and is centered in the window's client area.

Tiled The image displays at its default size and is repeated so it fills the entire client area.

Extra Tab

Icon

To associate an icon with the window (and add a system menu—see **System Menu** below), specify an icon file name (.ICO file) in this field. Type the file name or press the ellipsis button (...) to select a file name with the standard **Open File** dialog.

You should usually specify an icon for an application window and for an MDI child window. Specifying an icon automatically places a minimize button on the caption bar of your application or MDI child window.

Tip: If you embed SYSTEM{PROP:Icon}='~MyIcon.ICO' in the *After Opening the Window* embed point of your Application Frame, the Application Generator applies MyIcon.ICO to all of the application's MDI child windows that have no icon specified.

Palette

To specify maximum color depth, fill in the **Palette** field. This does *not* mean your end user's hardware will support the palette. Type the total number of colors you wish to support. For example, 24-bit color would be 16777215. Leave this field zero to specify the default for the end user's system.

Timer

To have the window receive Timer Event messages from Windows, fill in the **Timer** field. Specify the timer interval in hundredths of seconds. For example, if you specify 100 in the field, the window receives an EVENT:Timer once every second. This might be appropriate for adding a clock to a status bar. See also *Windows Design Issues—Background Processing* and TIMER in the *Language Reference*.

Tip: Though Windows places limits on the number of active timers, you can place as many timers on as many windows as you like in your Clarion application. At run-time, your application uses only one Windows timer.

Immediate

To generate an event each time the end user moves or resizes the window, check this box. You are responsible for the code that executes for the event.

Status Bar

To provide a message bar at the bottom of your window, check this box. See *Status Widths* below for information on segmenting or zoning your status bar.

Tip: A status bar in an application window is a good place to provide feedback to your user. Clarion makes it easy to post messages on the status bar advising the user of what your application is doing. Increasing user feedback makes the user more in control, more confident, and more efficient when using your application.

Modal Window

To specify a system modal window, check this box. This box is already checked when you choose **System Modal Window** from the **New Structure** dialog.

A system modal window seizes control of the entire system and prevents any other tasks—even in other applications—from executing until the window is closed.

Entry Patterns

To add the MASK attribute to your window, check this box. This causes Clarion to enforce the entry patterns for all the fields in this window. For example, you may have specified an entry pattern of @P###-##-####P for a Social Security number field. Checking the **Entry Patterns** box means the entry pattern will be enforced on this window.

Tip: Entry Patterns are also known as Picture Tokens.

The entry patterns, or picture tokens, are specified on the **General** tab of the **Entry Properties** dialog. See *Controls and Their Properties—Entry Properties* for information on specifying entry patterns. See the *Language Reference* for more information on the MASK attribute.

System Menu

To place a system menu in your window, check the **System Menu** box, or specify an icon file (see **Icon** above), or specify a maximize box (see **Maximize** below). When your window has the SYSTEM attribute Windows 95 and Windows NT display an icon in the upper left corner. If you specify an icon (see **Icon** above), that icon displays, otherwise the system default icon displays. Initially, the system default icon is set to the Clarion icon; however, you can specify a system default icon with:

```
System{PROP:Icon} = My.ico
```

Activate the system menu by CLICKING the button, box, or icon in the upper left corner of the window. Standard system menu choices include Restore, Minimize, Maximize, and Close.

Every application frame should have a system menu. For users on a system without a mouse, the system menu provides the only means of minimizing, maximizing or re-sizing the application window.

Auto display

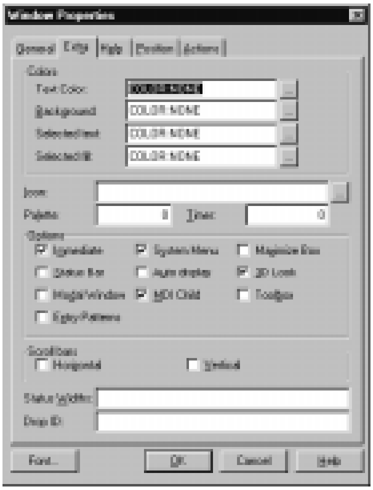
To add the AUTO attribute to your window, check this box. This automatically updates the contents of all controls on the window on each pass through the ACCEPT loop.

MDI Child

To specify an MDI child window, check this box. An MDI child window cannot move outside the main application window.

When you START a procedure on its own thread, the procedure and its window operate independently of other threads in the same program; that is, the end user can switch focus between each execution thread at will. These are “**modeless**” windows.

If you don’t initiate a new thread, the program behavior depends on whether the procedure’s window has the MDI attribute. A non-MDI child window on the same thread as its parent, blocks access to all other threads in the program. This is an “**application modal**” window. When the application modal window closes, the other execution threads are available again. An MDI child window on the same thread as its parent, blocks access only to its parent window. When the MDI child window closes, its parent window regains focus.



Text Color: [illegible]
Background: COLOR:NONE
Selected text: COLOR:NONE
Selected fill: COLOR:NONE
Icon:
Palette: 0 Timer: 0

☑ Immediate ☑ System Menu ☐ Maximize Box
☐ Status Bar ☐ Auto display ☑ 3D Look
☐ Modal Window ☑ MDI Child ☐ Toolbox
☐ Entry Patterns

☐ Horizontal ☐ Vertical

Status Widths:
Drop ID:

Maximize Box

To place a maximize button in your window (and a system menu—see **System Menu** below), check this box. In general, you should place a maximize button on application windows and MDI child document windows, not on dialog boxes.

3D Look

To provide the gray window background, and chiseled control look for your window, check this box. This is clearly a style consideration, but will go a long way in giving your application a professional look.

The gray background is not visible when you design your window with the **Window Formatter**. It is visible in **Preview!** mode and when your application is running.

Toolbox

To add the TOOLBOX attribute to your window, check this box. The TOOLBOX attribute makes your window always stay “on top” of other open windows.

Docking Options

These options allow you to specify the behavior of a dockable toolbox. These options are available only when the **Toolbox** box is checked. The check boxes set the DOCK attribute.

Left

Allows the toolbox to dock to the left side of the parent window.

Right

Allows the toolbox to dock to the right side of the parent window.

Top

Allows the toolbox to dock to the top of the parent window.

Bottom

Allows the toolbox to dock to the bottom of the parent window.

Float

Allows the toolbox to float within the client area of the parent window.

Initial State

Select the window’s initial docking state from the dropdown list. Only those options selected above are available in the list. This sets the DOCKED attribute.

Scroll Bars

Horizontal

To add a horizontal scroll bar to your window, check this box.

Vertical

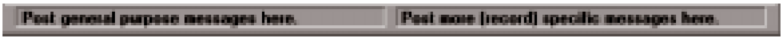
To add a vertical scroll bar to your window, check this box.

Status Widths

To set the width of status bar zone(s), type a value, or a list of values separated by commas. You must also check the **Status Bar** box. See above. The values you enter in this field provide the STATUS() attribute parameters for your window. See *STATUS* in the *Language Reference*.

If your application has *no* status bar, or has only *one* zone on the status bar, you may omit this field.

Status bar *zones* are the areas within the status bar marked off by the 3D shaded boxes. The first zone on the left, by default, displays MSG attribute text from the control with input focus. This is useful for showing brief instructions or other information to the user.



Post general purpose messages here.

Post more [record] specific messages here.

The values you enter represent the *width*, in dialog units, of each zone. A dialog unit is 1/4 the width of the average character in the default character set. Thus *a value of 40 produces a zone about 10 characters long*. A value of 400 produces a zone about 100 characters long.

You may specify an expandable zone by typing a *negative* number. A negative number creates a zone with a minimum width, that expands as far as the window size will allow.

Use property assignment syntax to place text in any zone. To place a string in the second zone, for example:

```
  MyWindowLabel{PROP:StatusText,2} = ‘A String’
```

Tip: A multi-zone status bar can give your application a professional look. You may display help text in zone one, and when editing a record, the current record number in zone two, for example.

Drop ID

To add the DROPID attribute to your window, type up to 16 comma delimited signatures. The DROPID attribute indicates this window is a valid target for “Drag and Drop” operations. The signature is a string constant that identifies which types of drag and drop operations are valid for this window.

Drag and Drop capability means the end user can select an item in one window or control, hold down the left mouse button, “drag” the item to another window or control, and release the mouse button, “dropping” the item onto the other window or control, which can then look at the item that was “dropped” on it, and do something with it.

Implementation of this capability requires that the *source* control have a DRAGID attribute with a *signature* that matches the target window’s or control’s DROPID *signature, and* that the procedures that drive each window have appropriate source code to process the drag and drop events. See the *Language Reference* for more details and examples. Also see *How to Add Drag and Drop Capability* in the on-line help.

Colors Tab

Enter a valid color value in any of the following fields to add the COLOR attribute to your WINDOW declaration. See the *Language Reference* for more information on COLOR and WINDOW.

See ..\LIBSRC\EQUATES.CLW for a list of valid color equates.

See the *Windows Design Issues* appendix for a discussion on using color to enhance your application.

Text Color

To apply a specific color to all client area text, type a valid color equate in this field, or press the ellipsis (...) button to select a color from the **Text Color** dialog.

Background

To apply a specific color to the entire window, except for title bar, selected text, and window controls, type a valid color equate in this field, or press the ellipsis (...) button to select a color from the **Background Color** dialog.

Selected text

To apply a specific color to the window's selected text, type a valid color equate in this field, or press the ellipsis (...) button to select a color from the **Selected Color** dialog.

Selected fill

To apply a specific color to the background of the window's selected text, type a valid color equate in this field, or press the ellipsis (...) button to select a color from the **Selected Background Color** dialog.

Help Tab

Cursor

To specify the cursor appearance for the window, choose a cursor from the drop-down list, or type in the name of a .CUR file. When the user passes the cursor over the window, the cursor takes the image defined in the .CUR file. Controls within the window automatically inherit the same cursor unless you override it.

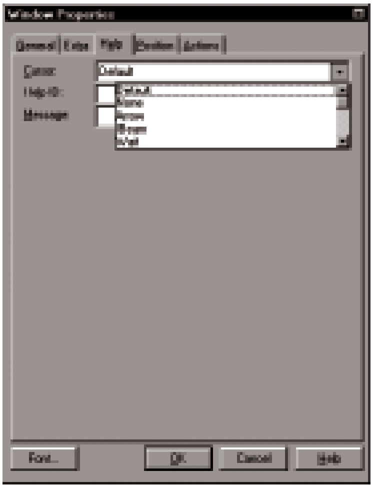
Tip: See the *Windows Design Issues* chapter for tips on when to use each cursor.

Help ID

To associate a Help ID with the window, fill in a keyword or context string (preface the context string with a tilde ~ character). This fills in the HLP attribute for the window.

Tip: You must author your help file using a word processor that supports output to .RTF files (such as Microsoft Word for Windows™). You must compile the help file with the Windows Help Compiler, which is available from Microsoft.

When the user calls Windows Help while the window is active, it opens to the associated topic. Should you set a Help ID for individual controls within the window, they override the window's Help ID while the control has focus.



When generating code, the Application Generator calls the context string or keyword in the .HLP file you specify in the **Application Properties** dialog. See *Application Generator* for more information.

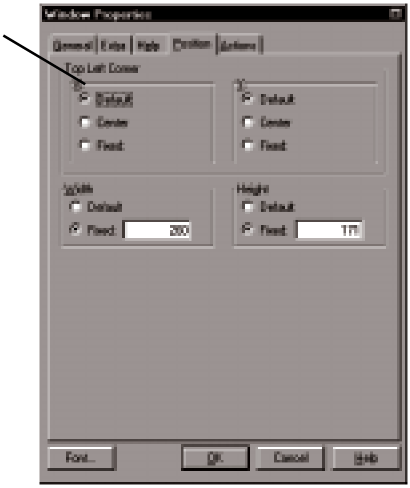
Message

Type a string to display in zone one of the status bar when the window is active. This provides the MSG attribute for the *window*. Messages may also be specified for *controls* in the window. When a control has focus, the control's MSG is displayed instead of the window's MSG. See *Status Bar* and *Status Widths* above for more information on the status bar. See *MSG* in the *Language Reference*.

Position Tab

Generally, you will want to size your window by dragging its handles in the **Window Formatter**. Handles are the tiny boxes that appear at the corners and the sides of selected items. However, you may specify the window size *and* its position with the **Position Tab.**

The Position tab controls the position *and* size of the window. When you choose *default* positions, Windows places your window at a point depending on where the last window (even from another application) was opened.



Window Properties

Top Left Corner

X: ● Default ○ Center ○ Fixed

Y: ● Default ○ Center ○ Fixed

Width: ○ Default ● Fixed: 280

Height: ○ Default ● Fixed: 170

Top Left Corner

To specify the initial position (of the top left corner) of your window, choose the desired **X** and **Y** coordinates. Choose from:

Default This instructs Windows to set the **X** and/or **Y** positions of the upper left corner of the window to a default value which will depend on the user's system and on the number of other active applications.

Tip: To give your application the "standard" look of other Windows applications, use the *Default* setting wherever possible.

Center Places the window in the center of the desktop. You may choose horizontal centering, vertical centering, or both. Adds the CENTER attribute to the WINDOW. See the *Language Reference* for more information.

Fixed To set a specific position, mark the **Fixed** choices for the **X** and **Y** coordinates. This fixes the position of the upper left corner of the window. For the APPLICATION frame this position is relative to the desktop, for an MDI child window, this position is relative to the APPLICATION frame's client area.

The measurement units for these coordinates are dialog units. Dialog units are a *relative* measure based on the default character set. A dialog unit is 1/4 of the width of the average character, and 1/8 of the height of the average character. Thus, Windows proportionally repositions the window at different screen resolutions.

Width

To specify the width of your window, choose the desired width value. Choose from:

Default This instructs Windows to set the width of the window to a default value which will depend on the user's system and on the number of other active windows and applications.

Fixed To set a specific width, mark the **Fixed choice** and specify a value. This sets the width of the window in dialog units.

Height

To specify the height of your window, choose the desired height value. The choices are the same as for **Width**.

Actions Tab

There are no default actions specific to a window. However, this tab provides alternative access to the File Schematic and the Embed points for the procedure. Simply press the corresponding buttons.

Placing Controls in a Window

This section explains how to *place* a control in a window. The *Controls and Their Properties* chapter explains how to *customize* the controls you place.

To place a control

1. CLICK on an icon in the **Controls** toolbox, or choose a control from the **Controls** menu or the **Populate** menu.
2. After you have selected a control or control template, pass the cursor over the sample window.

   The cursor becomes a crosshair (+). Position the crosshair where you want the control to appear.
3. CLICK the mouse.

   The **Window Formatter** places the upper left hand corner of the control at the intersection of the cursor crosshair when you CLICK the mouse. By default, the control takes on the size of the other controls of that type already in the window. If there are no like controls in the window, the control is the default size.

Tip: **You can control the initial height and width of a control with the [Control Defaults] section in the Clarion5.INI file (installed by default to \CLARION5\BIN). See the *Programmer's Guide* for more information on these settings.**

4. If you chose a LIST, a COMBO, or a BUTTON, and you checked the **Translate controls to control templates when populating** box in the **Application Options** dialog, then select the control or control template from the list.

Control templates generate source code to declare controls and manage their associated data. For example, the BrowseBox control template not only generates source code to display a listbox control, it also generates code to load data from a file into a QUEUE, then display the data in the listbox with complete scrolling and mouse-click selection capability.

Tip: Generally, it is to your advantage to use a Control template rather than a simple control.

Tip: When using the Window Formatter to place a Dictionary Field, it automatically opens the Select Field dialog so you can select or define a data dictionary field or memory variable to associate with the control. Once placed, you can access the control's Properties dialog from the Edit menu or from the right-click popup menu.

5. If necessary, CLICK and DRAG on a control handle to *resize* the control. CLICK and DRAG on the interior of the control to *move* the control.

Menu Editor

A menu is a list of the various actions your application may perform. In Clarion, this list of actions (menu) is declared using the MENUBAR structure, MENU structures, and ITEMs. In this chapter, the word menu generically refers to the list of actions your application may perform. The words MENUBAR, MENU, and ITEM refer to Clarion Language statements that define your application's menu.

The Menu Editor is a visual design tool—a subset of the **Window Formatter**—that generates Clarion Language statements to define your application's menu.

This section:

- Discusses dynamic menu management for Multiple Document Interface (MDI) applications.
- Shows you how to call the **Menu Editor** and create a menu.
- Describes how to automatically implement Standard Windows Behavior (SWB) for commands such as **Edit ➤ Copy** by linking a Clarion Standard ID (STD attribute) to an ITEM or MENU.

Merging MDI Menus

Multiple Document Interface (MDI) applications make special demands upon a program. Often, the program may support a variety of document windows, each of which has a slightly different set of commands from which the user may select. See *Windows Design Issues* for more information.

Normally in an MDI application, the developer writes code to monitor which window is active and to change the menus and toolbars to reflect the options currently available to the user. Clarion does this automatically by merging menus and toolbars according to preferences you specify with the **Menu Editor.** However, accurate specification requires some understanding and planning by the application developer.

Global Selections

On an APPLICATION frame, the MENUBAR defines the *Global* menu selections for the program. These Global menu selections are generally *available on all MDI "child" windows*. However, if the NOMERGE attribute is present on the application's MENUBAR, then *there is no Global menu*, and the application's menu is a *Local* menu displayed only when no MDI child windows are open.

Local Selections

On an MDI child window, the MENUBAR defines *Local* menu selections that are automatically merged with the *Global* menu selections defined on the application's MENUBAR. Both the Global and the Local menu selections are available while the MDI "child" window has input focus. Once the window loses focus, its Local menu selections are removed from the Global menu selections. If the NOMERGE attribute is specified on an MDI child window's MENUBAR, *the Local menu overwrites and replaces the Global menu*.

Non-MDI Windows

On a non-MDI window, the Local menu selections are *never* merged with the Global menu selections. A MENUBAR on a non-MDI window always appears in the window, and not on any application frame which may have been previously opened.

Merging Order

Normally, when an MDI window's menu (Local selections) merges into an application's menu (Global selections), the Global menu selections appear "first", followed by the Local menu selections. First means either toward the *left* or toward the *top*, depending on whether the merged selection is displayed on the action bar (horizontal list) or in a menu (vertical list).

The merge process also considers whether any Local MENUs *match* any Global MENUs. MENUs that have the same name and the same MENUBAR level, match. When there are no matches, the menus merge in the normal order. However, when MENUs match, a single menu (vertical list) results with the Global selections appearing *above* the Local selections. This new menu has all the attributes of the Global MENU, such as, MSG, FIRST, etc. Within this merged menu, any matching subMENUs are also merged into a single menu. Note that ITEMS are not merged, even when they match.

The normal merging order may be modified by using the **Menu Editor's Position** drop-down list (see *Specifying Menu Positions and Merging Behavior* below) to add FIRST or LAST attributes to individual MENUs and ITEMs. The merge position priority is:

1. Global selections with FIRST attribute
2. Local selections with FIRST attribute
3. Global selections without FIRST or LAST attributes
4. Local selections without FIRST or LAST attributes
5. Global selections with LAST attribute
6. Local selections with LAST attribute

See the *Language Reference* for more information on these attributes.

Planning and Implementing Menus

To create menus for MDI applications:

1. Create a master menu for the APPLICATION frame window.

   Most likely, this will include a File menu and a Help menu, since they contain functions that are available even when no document windows are open.

   Tip: Clarion's Application Frame procedure template comes with a predefined menu with many of the most common functions already provided for you.

   Use the **Window Formatter's Menu Editor** to create your menus. Be sure to choose the FIRST attribute for the File MENU, and the LAST attribute for the Help MENU from the **Position** drop-down list. This ensures that File and Help will maintain their relative positions when Clarion merges this global menu with local menus.

2. Plan the additional menus for the child windows.

   Can they all share the same menu titles? Do they share many of the same commands? Ideally, *most* of the MENUs and ITEMs can be active in *all* the child windows. If there are only a few commands specific to certain windows, plan on disabling those MENUs and ITEMs in the windows that don't support them, and enabling them in those that do.

3. Create the menu for the first child window.

   Again, you will use the **Window Formatter's Menu Editor** to create the menu. Add any window-specific MENUs to the first child window. That is, the window-specific MENUs the application frame lacks-—such as Edit, Insert, etc.

   Optionally, add a File MENU to the first child window. This is necessary only if the child window needs an ITEM on the File MENU that is not already included on the application's File MENU. For example, adding a Close command might be appropriate. If so, add the File MENU to the first child window. Add the Close ITEM to the File MENU.

   Add the Window MENU to the first child window. Window MENUs are standard for most windows programs. A typical Window MENU includes the following ITEMs: Arrange Icons, Tile, Cascade, plus a document (windows) list that displays all open child windows and allows the user to switch between them. In many cases this entire MENU, including the document list, can be implemented with standard ID's (StdID's). See *Creating Your Application's Menu* below.

4. Exit the **Menu Editor** and save the menu.

5. Test the interaction of these first two menus.

Do they merge the way you planned? Are the correct selections available for the window with focus? Make any adjustments with the **Menu Editor**.

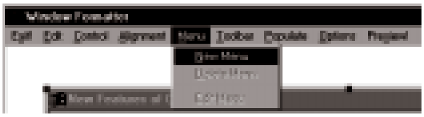
6. Repeat steps 3 through 5 for other child windows.

Calling the Menu Editor

To create a menu for your application, use the **Menu Editor**. You access the **Menu Editor** through the **Window Formatter**.

> **Tip: You can also create a toolbar for your application using the Window Formatter. See *Toolbars* for more information.**

This section provides detailed examples of using the **Menu Editor** to create menus. From the **Window Formatter**, choose **Menu ➤ New Menu** to create a new menu or choose **Menu ➤ Menu Editor** to edit an existing menu.



The **Menu Editor** dialog visually represents a Clarion MENUBAR data structure. The menu tree (on the left hand side of the dialog) appears as simplified Clarion language syntax, containing these Clarion keywords:

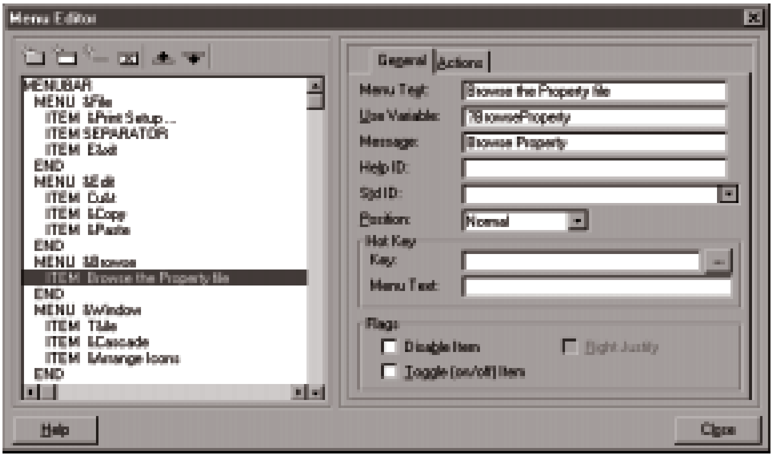
- A MENUBAR keyword at the top.
- A MENU statement or statements followed by a menu name, and a corresponding END statement.
- An ITEM statement or statements followed by an item name.

Menu Editor command buttons allow you to add and delete MENUs and ITEMs. You may also move MENUs and ITEMs within the MENUBAR structure with the ▲ and ▼ buttons.

The right hand side of the dialog lets you specify the text of your MENUs and ITEMs, the equate labels used to reference the MENUs and ITEMs in executable code, and the actions that occur when the user selects an ITEM.

> **Tip: When using the Application Generator, each ITEM you place on a MENU or MENUBAR automatically adds an embed point to the control event handling tree in the Embedded Source dialog. This lets you easily attach functionality to your ITEMs.**

The following section provides a step by step procedure for creating a menu. Following that are sections detailing the **Menu Editor** commands and options, and a discussion of considerations to keep in mind when creating MDI application menus.



```
MENUBAR
  MENU &File
    ITEM &Print Setup...
    ITEM SEPARATOR
    ITEM E&xit
  END
  MENU &Edit
    ITEM Cu&t
    ITEM &Copy
    ITEM &Paste
  END
  MENU &Browse
    ITEM Browse the Property file
  END
  MENU &Window
    ITEM T&ile
    ITEM &Cascade
    ITEM &Arrange Icons
  END
```

Menu Text: Browse the Property file

Use Variable: ?BrowseProperty

Message: Browse Property

Help ID:

Std ID:

Position: Normal

Key:

Menu Text:

□ Disable Item □ Right Justify

□ Toggle (on/off) Item

Creating Your Application’s Menus

Here are the steps for creating a menu starting from an empty window within the **Window Formatter**.

1. Choose the **Menu ➤ New Menu** command.

   The **Menu Editor** dialog appears. Only the MENUBAR statement is present.

2. Press the new menu button.

   This adds the first MENU statement, its name, and its corresponding END statement, ready for editing.

   The ampersand within the MENU name signifies that the character following the ampersand is the accelerator key. That is, the character is underlined (for example: Menu1), and, when the user presses ALT+*accelerator key*, the menu displays.

3. In the **Menu Text** field, type the text to display for this MENU.

   For example, type *&File*, so the end user sees **File**.

   You can embed tabs and other special characters within your menu text. To embed special characters (such as the tab character) in your menu text:

   In the **Procedure Properties** dialog, press the **Window** ellipsis (...) button.

   This opens the Text Editor to the WINDOW declaration for this procedure. When you embed special ASCII characters in your WINDOW declaration, you must edit the source code directly, because the **Window Formatter** doesn’t recognize the ASCII delimiter characters (<,>).

   In your ITEM text, type <*n*> where you want the character to appear.

Where *n* is the ASCII code for the character. The brackets (<,>) tell the Clarion compiler to insert the ASCII characters specified within. For example:

```
ITEM('Cut<9>Ctrl+X'),USE(?CutText),KEY(CtrlX),STD(STD:Cut)
```

Exit the Text Editor and save.

Tip: You can embed special characters within POPUP menu text and MESSAGE text by using the ASCII delimiters (<,>).

4. In the **Use Variable** field, type a Field Equate Label.

   A Field Equate Label has a leading question mark (?), and you should make it descriptive. For example ?File shows this menu is to manipulate a file. You can refer to the MENU in your source code by its Field Equate Label.

5. Press the new item button.

   This inserts an ITEM between the MENU statement and its END statement. Note that ITEMs are used to execute commands or procedures, whereas MENUs are used to display a selection of other MENUs or ITEMs.

6. In the **Menu Text** field, type the text to display for this menu ITEM.

   For example, type *&Open*, so the end user sees **Open**. The ampersand within the ITEM name signifies the character following the ampersand is the accelerator key. That is, the character is underlined, and, when the user presses the accelerator key, the action associated with the ITEM executes.

Note: A MENU accelerator key requires the ALT key to take effect, whereas an ITEM accelerator key does *not* require the ALT key, but does require that the ITEM is currently displayed. See *Adding a Hot Key* below for another method of invoking your MENUs and ITEMs.

7. In the **Use Variable** field, type a Field Equate Label.

   A Field Equate Label has a leading question mark (?), and you should make it descriptive. For example ?FileOpen shows at a glance the intended purpose of this ITEM: to open a file.

   You refer to an ITEM within source code by its Field Equate Label.

8. In the **Message** field, type the MSG attribute contents.

   This message text displays in the status bar (if enabled) when the user highlights this MENU or ITEM.

9. In the **Help ID** field, type either a help keyword or a context string present in a .HLP file.

If you fill in the **Help ID** for a MENU or an ITEM, when the user highlights the MENU or ITEM and presses F1, the help file opens to the referenced topic. If more than one topic matches a keyword, the search dialog appears.

The **Help ID** field (HLP attribute) takes a string constant specifying the key for accessing a specific topic in a Windows Help file. This may be either a Help keyword or a context string.

A Help keyword is a word or phrase indexed so that the user may search for it in the *Help* **Search** dialog.

> **Tip: When authoring a Windows Help file, you indicate a keyword with the 'K' footnote. A Help context string is the arbitrary string which uniquely identifies each topic page for the Windows Help Compiler. When creating the Help file, the '#' footnote marks a context string. These tasks are all done for you by many help authoring tools.**

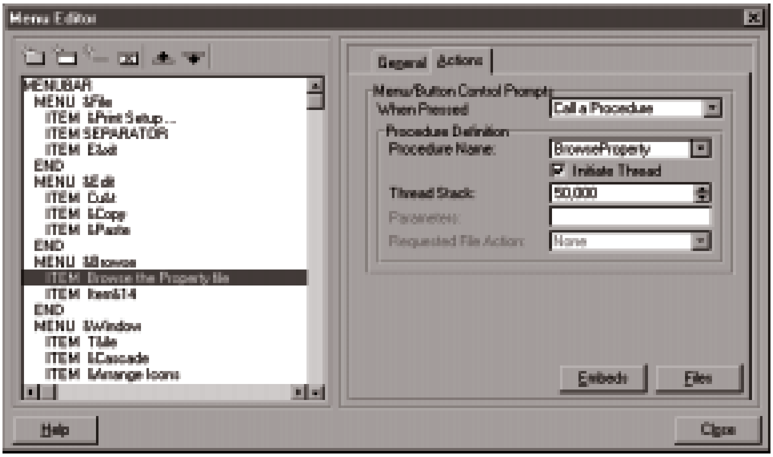
When referencing a context string in the **Help ID** field, you must identify it with a leading tilde (~).

10. From the **Actions Tab**, choose **Call a Procedure** from the **When Pressed** drop-down list.

    The procedure you specify executes when the user selects this ITEM. You may specify parameters to pass and standard file actions (insert, change, delete, or select) if applicable (Clarion's Procedure templates understand and carry out the requested file actions). Or you may initiate a new thread. The *procedure* appears as a "ToDo" item in your Application Tree (unless you named a procedure that already exists).

When you START a procedure on its own thread, the procedure and its window operate independently of other threads in the same program; that is, the end user can switch focus between each execution thread at will. These are "**modeless**" windows.

If you don't initiate a new thread, the program behavior depends on whether the procedure's window has the MDI attribute. *A non-MDI* child window on the same thread as its parent, blocks access to all other threads in the program. This is an "**application modal**" window. When the application modal window closes, the other execution threads are available again. *An MDI* child window on the same thread as its parent, blocks access only to its parent window. When the MDI child window closes, its parent window regains focus.



```
MENUBAR
  MENU &File
    ITEM &Print Setup...
    ITEM SEPARATOR
    ITEM E&xit
  END
  MENU &Edit
    ITEM Cu&t
    ITEM &Copy
    ITEM &Paste
  END
  MENU &Browse
    ITEM Browse the Property file
    ITEM Item&14
  END
  MENU &Window
    ITEM T&ile
    ITEM &Cascade
    ITEM &Arrange Icons
```

When Pressed: Call a Procedure

Procedure Name: BrowseProperty

☑ Initiate Thread

Thread Stack: 50,000

Parameters:

Requested File Action: None

This is one way to add functionality to your ITEM. You may also add functionality by choosing **Run a Program** from the drop-down list, by embedding source code, or by typing an STD ID in the **STD ID** field. STD IDs give your application Standard Windows Behavior (SWB) for common actions such as File/Open and Edit/Cut, Copy, and Paste. See *Implementing Standard Windows Behavior.*

After following these steps, you have a single MENU called **File**, with a single ITEM called **Open**. To add other ITEMS to the MENU, repeat steps *4* through *11*. To add a second MENU, select the END statement and press the **Menu** button. To add a subMENU, select a MENU or ITEM statement and press the **Menu** button.

11. To finish the menu and return to the **Window Formatter**, press the **Close** button.

Implementing Standard Windows Behavior

There are some menus and commands that you see in almost every windows program. For example, Cut, Copy, and Paste. Clarion provides an easy method for implementing these standard actions in your application menus—with the **Std ID** field on the **Menu Editor** dialog.

To specify a standard action for your menu ITEM, enter one of the equates listed below in the **Std ID** field. Clarion automatically implements the command using standard windows behavior; you do not need any other support for it in your code. The standard equate labels and their associated actions are also contained in the \LIBSRC\EQUATES.CLW file.

STD:PrintSetup	Opens Printer Options Dialog
STD:Close	Closes active window
STD:Undo	Reverses the last editing action
STD:Cut	Deletes selection, copies to clipboard
STD:Copy	Copies selection to clipboard

STD:Paste	Pastes clipboard contents at the insertion point
STD:Clear	Deletes selection
STD:TileWindow	Arranges child windows edge to edge
STD:TileHorizontal	Arranges child windows edge to edge
STD:TileVertical	Arranges child windows edge to edge
STD:CascadeWindow	Arranges child windows so title bars are visible
STD:ArrangeIcons	Arranges iconized child windows
STD:WindowList	Select child windows from a MENU
STD:Help	Opens .HLP file to the contents page
STD:HelpIndex	Opens .HLP file to the index
STD:HelpOnHelp	Opens Microsoft's Help system.HLP file
STD:HelpSearch	Opens Microsoft's Help Search .HLP file.

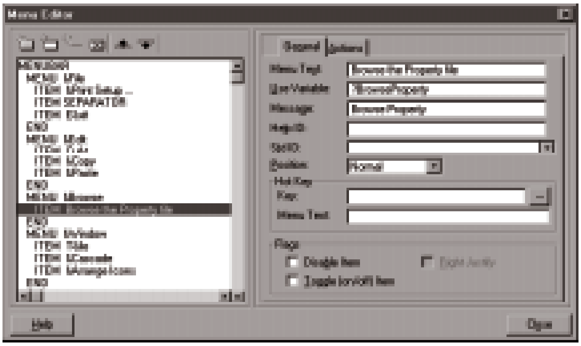
Menu Positions and Merging Behavior

The **Position** drop-down list lets you specify MENU and ITEM order priority when Clarion merges menus. Choose from:

Normal
Set normal merging order. In normal merging, Global selections precede Local selections. See *Merging Menus* above.

First
Force the selected MENU or ITEM to the first position when merging menus. This adds the FIRST attribute to the MENU or ITEM statement. See the *Language Reference* for more information. Also, see *Merging Menus* above.

Last
Force the selected MENU or ITEM to the last position when merging menus. This adds the LAST attribute to the MENU or ITEM statement. See the *Language Reference* for more information. Also, see *Merging Menus* above.



```
MENUBAR
  MENU &File
    ITEM &Print Setup ...
    ITEM SEPARATOR
    ITEM E&xit
  END
  MENU &Edit
    ITEM Cu&t
    ITEM &Copy
    ITEM &Paste
  END
  MENU &Browse
    ITEM &Browse the Property file
  END
  MENU &Window
    ITEM T&ile
    ITEM &Cascade
    ITEM &Arrange Icons
  END
```

Menu Text: Browse the Property file
Use Variable: ?BrowseProperty
Message: Browse Property
Help ID:
Std ID:
Position: Normal
Hot Key
Key:
Menu Text:
Flags
□ Disable Item
□ Toggle (on/off) Item
□ Right Justify

The following two **Flags** let you specify whether or not your menu can be merged, and right justification of selections displayed on the actionbar:

Do Not Merge

To never merge this MENUBAR with other MENUBARs, check this box. This is available only for the MENUBAR. See *NOMERGE* in the *Language Reference* for more information.

Right Justify

To right justify the selected MENU, check this box. This is available only for MENUs on the action bar. Nested MENUs (subMENUs) cannot be right justified. Checking this box displays the selected MENU, and all MENUs after the selected MENU, at the far right of the action bar.

Adding Hot Keys

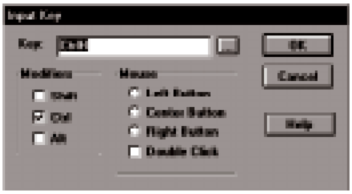
A hot key is very similar to an accelerator key. A hot key or hot key combination allows the end user to immediately display a MENU, or execute the action associated with an ITEM, without mouse clicking, and *without displaying the menu that contains the* ITEM. Customarily, hot keys take the form of CTRL + *character*, or CTRL + SHIFT + *character*. To add a hot key:

1. Press the **Key** ellipsis (...) button to select the hot key combination.

   This opens the **Input Key** dialog. Use this dialog to add the KEY attribute to your MENU or ITEM. The KEY attribute specifies a "hot" key or key combination.

2. From the **Input Key** dialog, specify the hot key or key combination by pressing the desired key or keys.

   The keys you press appear in the **Key** field, and are supplied as the parameter to the KEY attribute for this menu item.



Input Key

Key: CtrlH

Modifiers

□ Shift

☑ Ctrl

□ Alt

Mouse

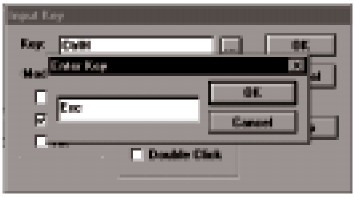
○ Left Button

○ Center Button

○ Right Button

□ Double Click

Mouse clicks may be used as hot keys; however, mouse clicks *cannot* be specified by clicking the mouse. For mouse clicks, check the corresponding check box(es). For example, to execute the Open command when the user double-clicks, check the **Left Button** box *and* the **Double Click** box.



Input Key

Key: CtrlH

Enter Key

Esc

□ Double Click

The ESC, ENTER, and TAB keys may be used as hot keys, but they *cannot* be specified by pressing them. For these keys, press the ellipsis (...) button and type “esc,” “enter,” or “tab.”

> **Tip: See *Windows Design Issues* for a list of common hot keys associated with standard windows commands.**

3. Press the **OK** button to return to the **Menu Editor** .
4. To display the hot key combination (or any other text) as right-justified menu text, type the text in the **Menu Text** field.

   For example, if you set the hot key as CTRL+*h*, type ctrl+h in the **Hot Key Menu Text** field. Then, each time the end user uses the menu, the right-justified text reminds her that the menu choice can be invoked with the ctrl+h keystrokes.

Other Menu Behavior—Disabling and Toggling

The following two **Flags** let you disable a selection, and set up an ITEM as toggle switch.

Disable Item

To disable a MENU or ITEM (dim the text and make it unavailable to the user), check this box. This adds the DISABLE attribute to the MENU or ITEM statement.

> **Tip: The Disable box is handy when you incorporate modality into a program—that is, when one type of child window does *not* support the same commands another type does. For the type that doesn’t support the command, disable the ITEM rather than omitting it. This will avoid confusing the user with menu ITEMs that disappear and reappear depending on which window is active.**

Toggle (on/off) Item

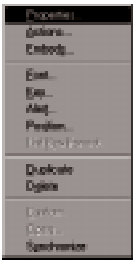
To create an on/off toggle for a selected ITEM, check this box. The ITEM should have a numeric variable in the **Use Variable** field. The variable should be declared using one of the data dialogs, or in embedded source. See *Creating Your Application’s Menu* above. The **Menu Editor** adds the CHECK attribute to this ITEM.

With the CHECK attribute, when the user selects the item for the first time, the item is “on,” the USE variable’s value is one (1), and a check mark appears beside the item. When the user selects the item a second time, the item is “off,” the USE variable’s value is zero (0) and no check mark is displayed. You should add source code to control the application’s behavior depending on the state of the USE variable.

Managing Your Menus

Press this button to add a separator bar after the highlighted MENU or ITEM.

Tip: Separator bars can provide the user with a visual cue that a group of ITEMs on the menu perform related functions.



Press this button to delete the highlighted MENU or ITEM. If you delete a MENU, all nested ITEMs and MENUs, and its associated END statement are also deleted.

To move the highlighted MENU or ITEM up in the menu list, press this button. When moving a MENU, all ITEMs and MENUs within it and its associated END statement move also.

To move the highlighted MENU or ITEM down in the menu list, press this button. When moving a MENU, all ITEMs and MENUs within it and its associated END statement move also.

Toolbars

With a simple command in the **Window Formatter**, you can add a toolbar to any window. You can place any controls on a toolbar, but the ones you will probably use the most are command buttons, check boxes, radio buttons, and drop-down listboxes. As with menus, Clarion automatically merges global and locao toolbars in certain situations.

Merging Toolbars

Global and Local Toolbars

The TOOLBAR structure declares the tools displayed for an APPLICATION or WINDOW. On an APPLICATION, the TOOLBAR defines the Global tools for the program. Global tools are active and available on all MDI child windows unless the MDI child window's TOOLBAR structure has the NOMERGE attribute.

If you specify the NOMERGE attribute on the APPLICATION's TOOLBAR, there are no global tools; the tools are local and are displayed only when no MDI child windows are open.

Note: To merge toolbars, the APPLICATION's toolbar AT width must be less than the APPLICATION's frame width. In the Procedure Properties dialog, press the Window's ellipsis (...) button, then set the TOOLBAR's width (third AT attribute) equal to the X position plus the width of the rightmost toolbar control.

MDI Windows

On an MDI window, the TOOLBAR defines tools that are automatically merged with the Global toolbar. Both the Global and the local MDI window's tools are active while the MDI child window has input focus. Once the window loses focus, its specific tools are removed from the Global toolbar. If the NOMERGE attribute is specified on an MDI window's TOOLBAR, the tools overwrite and replace the Global toolbar.

Non-MDI Windows

On a non-MDI WINDOW, the TOOLBAR is *never* merged with the Global menu. A TOOLBAR on a non-MDI window always appears with the window itself, and not on the parent window.

Merging Order

When an MDI window's TOOLBAR is merged into an application's TOOLBAR, the global tools appear first, followed by the local tools. The toolbars are merged so that the fields in the window's toolbar begin just right of the position specified by the value of the width parameter of the application TOOLBAR's AT attribute. The height of the displayed toolbar is the maximum height of the "tallest" tool, whether global or local. If any part of a control falls below the bottom, the height is increased accordingly.

Toolbar Merging Checklist

Application Properties

1. Clear the NOMERGE box on the APPLICATION's toolbar.
2. TOOLBAR's width must be less than APPLICATION's width.

   In the **Procedure Properties** dialog, press the Window's ellipsis (...) button, then set the TOOLBAR's width (third AT attribute) equal to the X position plus the width of the rightmost toolbar control.

Child Window Properties

1. Clear the NOMERGE box on the WINDOW's toolbar.
2. Check the MDI box.

Adding Toolbars

The following describes how to add a toolbar to a window, then how to add a control to the toolbar. The starting point is the **Window Formatter**, open to an empty window:

1. From the **Toolbar** menu, choose **New Toolbar**.

   The **Window Formatter** places a toolbar at the top of the window (gray rectangle).
2. Optionally choose **Options ➤ Snap to Grid**.

   This makes sizing and placing the controls easier.
3. To place a control on the toolbar, CLICK on a control in the Controls toolbox, then CLICK inside the toolbar in the sample window.

   This places the control in the toolbar. See *Controls and Their Properties* for more information on various controls.

Tip: Uses .ICO files that are 32 x 32 pixels for toolbar buttons. These larger .ICO files contain both the enabled and the disabled icon in the same file, rather than requiring a separate file. When creating a custom .ICO file for a toolbar button, place your the image in the *center* of the icon file. Clarion automatically *crops* the icon image to fit the button size.

List box Formatter

The **List box Formatter** is a visual design tool—a subset of the **Window Formatter**—that generates Clarion Language statements to format your application's LISTs. The **List box Formatter** provides a wide degree of flexibility to create and modify your listboxes, drop-down listboxes, and combo boxes. You may invoke the **List box Formatter** from the **Window Formatter** or from the **Report Formatter**.

Once you specify a QUEUE to provide the data for the list (done automatically when specifying a BrowseBox template), the **List box Formatter** lets you customize your list in the following ways:

- You can set the number, content, and formatting of columns, with or without resizable borders.
- You can specify that a record (row) from the QUEUE occupies more than one listbox row.
- You can add horizontal scrollbars for each column or group of columns in the listbox.
- You can specify the focus on rows or individual "cells," spreadsheet fashion.
- You can specify headers for the listbox columns.
- You can add a locator control that allows users to quickly find the item they need.
- You can enable selection of multiple rows in the list.
- You can enable colorization of items in the list.
- You can enable iconization of items in the list.
- You can enable nesting (indenting) of items in the list.

List Overview

LIST controls come in a variety of forms, the most common of which are listboxes, drop-down listboxes, and combo boxes.

A *listbox*, by convention, is a read-only display of data in a tabular format (rows and columns). It is scrollable and may display many records and fields. It efficiently displays large amounts of data.

A *drop-down listbox*, by convention, is a read-only display of mutually exclusive selections or choices. It is often scrollable and is called a drop-down list because it initially appears as a single row, but "drops down" to display multiple rows, like a menu. It forces valid user selections, provides a visual cue reminding the user that a selection is required, offers a default selection, and doesn't take up much screen space.

A *combo box*, is a drop-down listbox with the ability to accept user input.

When creating a listbox control, you define its data source, its functionality, and its format. Clarion's development environment divides these property definitions among several dialogs:

- The **List Properties** dialog specifies a drop-down list versus a regular list, specifies the queue that supplies the list data, and specifies the general scrolling capability, that is, all the properties of the listbox that are *not* column-specific. See *Controls and Their Properties—List Properties*.
- The **List box Formatter** dialog lets you add, delete, reorder, and resize the specific fields or columns displayed in the listbox. And it defines the appearance and behavior of individual listbox columns as well as *groups* of columns. For example, define individual column headers, widths, and scrolling, or spread a header across several columns. This dialog is discussed here.

After you've started defining your listbox with the **List Properties** dialog, these are the general steps for completing your listbox.

Understanding the List box Formatter

Add columns, one by one, and format them

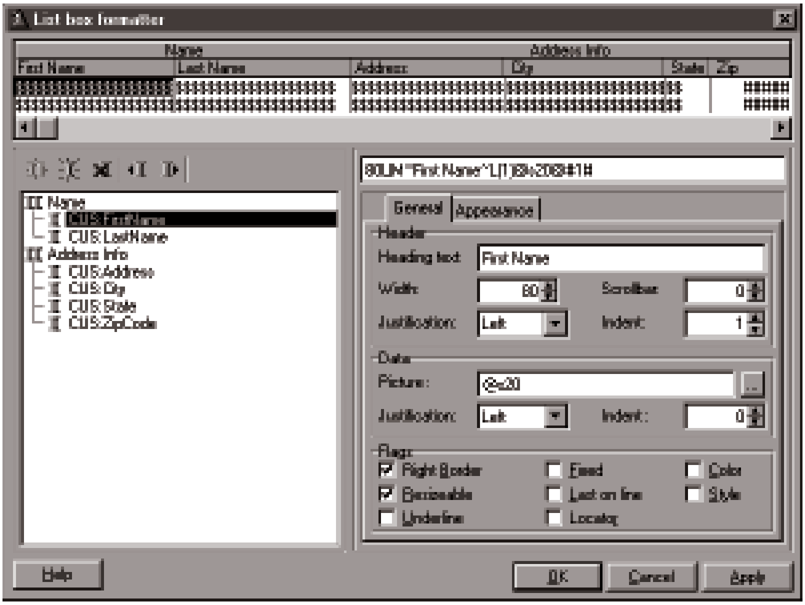
From the **Window Formatter**, RIGHT-CLICK on the LIST or COMBO control, then choose **List Box Format...** from the popup menu to open the **List box Formatter** dialog.

The Sample Listbox

The **List box Formatter** displays a representation of the listbox—the sample listbox. Each field appears as a column in the listbox, the data represented by "$" characters for strings, or "<" and "#" characters for numbers. If any field contains a header, a header row appears over the list.

You format the fields one by one, which updates the sample listbox. For your convenience, the **List box Formatter** provides a horizontal scroll bar whether or not you specify one in the **List Properties** dialog.

The **List box Formatter** does *not* display a vertical scroll bar, even if you checked the **Vertical** box in the **List Properties** dialog. However, the vertical scroll bar *does* appear at run time, *if* the queue contains more items than will fit in the listbox.



List box formatter

80L|M~First Name~L(1)@s20@#1#

Name
- CUS:FirstName
- CUS:LastName

Address Info
- CUS:Address
- CUS:City
- CUS:State
- CUS:ZipCode

Header
Heading text: First Name
Width: 80
Scrollbar: 0
Justification: Left
Indent: 1

Data
Picture: @s20
Justification: Left
Indent: 0

Flags
☑ Right Border
☐ Fixed
☐ Color
☑ Resizeable
☐ Last on line
☐ Style
☐ Underline
☐ Locator

- ❏ Press the new column button to add a new column.

  When working from within the Application Generator, choose a field from the **Select Field** dialog. The **List box Formatter** adds the selected field.

- ❏ Use the **General** tab to set the column-specific heading width and text, data format, scrolling, and other attributes.

- ❏ Use the **Appearance** tab to set the column-specific icons, colors, fonts, and nesting.

Tip: Formatting for the first column becomes the default format for subsequent columns.

For each setting you make, the **List box Formatter** creates the appropriate FORMAT attribute for the LIST statement that defines your listbox. See *FORMAT* in the *Language Reference* for more information.

Tip: At run-time, the PROP:Format property always contains the current format of the listbox, including any user changes. To save the user's column sizes, use GETINI and PUTINI to save and restore the PROP:Format values:

```
PUTINI(‘Settings’,‘UserList’,?List{PROP:Format},”MYAPP.INI’)
```

Tip: You can update the FORMAT string by typing directly into it, and your changes are reflected in the corresponding List box Formatter field.

- ❏ To remove a field from the listbox, press the button.

- To cancel the formatting changes, press the **Cancel** button.
- To accept the formatting changes and continue formatting, press the **Apply** button.
- To accept the formatting changes and close the **List box Formatter**, press the **OK** button.
- To move the selected field to the left, CLICK the ◀ button, or press CTRL+UP ARROW.

  If the selected field is leftmost in a group, the ◀ button moves the field *out* of the group, but the *order* of the fields is unaffected. Conversely, the ◀ button moves the selected field *into* a group at its immediate left, and the order of the fields is unaffected.
- To move the selected field to the right, CLICK the ▶ button, or press CTRL+DOWN ARROW.

  If the selected field is rightmost in a group, the ▶ button moves the field *out* of the group, but the *order* of the fields is unaffected. Conversely, the ▶ button moves the selected field *into* a group at its immediate right, and the order of the fields is unaffected.

List box Formatter General Tab

The tab lets you set the following formatting options.

Column Heading

Heading Text

Optionally specify header text for the column. The header appears as a gray row above the listbox data items. To specify no header, leave this field blank. If any field included in the listbox has a header, a header appears across the entire listbox; fields with no header text have a blank header.

The heading appears within the FORMAT string enclosed in tilde (~) characters, as in "~My Header~."

Width

Specify the width in dialog units for the column data. By default, the Formatter sets the value to four times the number of characters specified in the field picture in the data dictionary. For variables, the default is four times the number of characters in the picture token defined for it.

Tip: **As a rough guide, allow four dialog units for an average character. For example, if you want a column 10 characters wide, type 40 in the Width field.**

> **Tip: After you’ve placed a field, you can drag the column separators in the Sample Listbox to resize the column width. The cursor changes when you place it on top of the separator, to indicate you can resize it.**

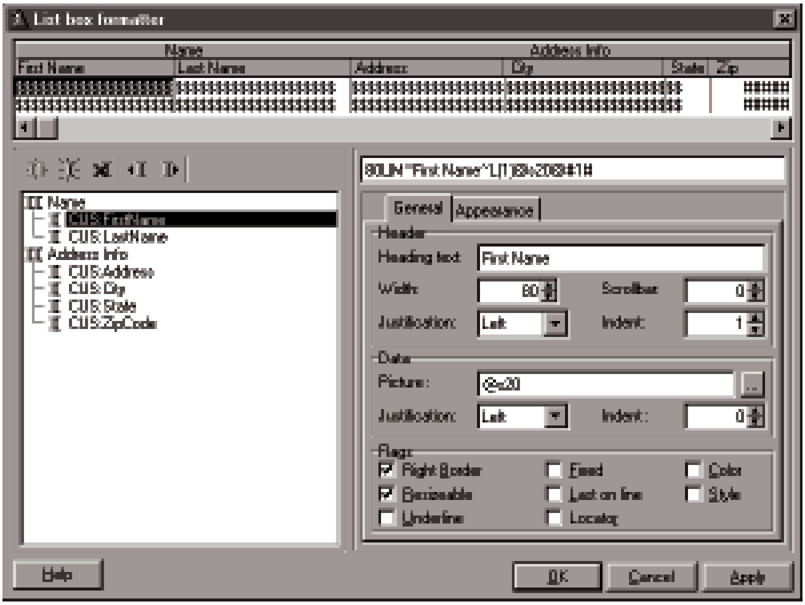
The data width you set appears within the FORMAT string for the field, preceding the Justification code, as in “40L.”

Scroll Bar

Type a non-zero value to specify a horizontal scroll bar for this column only. If the overall listbox already has a scroll bar, the column scroll bar appears above the listbox scroll bar. The value pecifies, in dialog units, how far the column scrolls.

For example, if your *data* is fifty (50) characters, and your listbox *column* width is about forty (40) characters (one hundred sixty (160) *dialog units*), you should specify a value of fifty (50). Fifty (50) additional dialog units are enough to display the ten characters that extend beyond the width of the listbox column.

The scroll bar and size appear in the FORMAT string together, as in “S(4).”



80L|M~First Name~L(1)|@s20@#1#

Heading text: First Name
Width: 80
Scrollbar: 0
Justification: Left
Indent: 1
Picture: @s20
Justification: Left
Indent: 0

☑ Right Border
☐ Fixed
☐ Color
☑ Resizeable
☐ Last on line
☐ Style
☐ Underline
☐ Locator

Justification

Choose from the drop-down list to specify left, right, center or decimal header justification.

This appears within the FORMAT string following the header, as in “~My Header~L.”

Indent

Optionally specify an indent, in dialog units, for the heading text. Indent moves the data by the number of dialog units specified, in the opposite direction to the justification. An indent of two (2) on left justification improves listbox readability.

This appears within the FORMAT string following the header, as in “~My Header~L(8).”

Column Data

Picture

Specify the picture token for the data. The **List box Formatter** displays the data according to the picture token. For example, the picture token @P(###) ###-####P displays a phone number as (555) 555-5555.

The picture token you specify appears in the FORMAT string.

Justification

Choose from the drop-down list to specify left, right, center or decimal. Decimal justification aligns decimal numbers by their decimal points.

The justification appears in the FORMAT string following the data width, as in “40R.”

Indent

Optionally specify an indent, in dialog units, for the listbox data. Indent moves the data by the number of dialog units specified, in the opposite direction to the justification. An indent of two (2) on left justified data improves listbox readability.

The indent appears within the FORMAT string surrounded by parentheses and preceded by a letter indicating the justification, as in “L(8).”

Formatting Flags

Right Border

Check this box to specify column separators between fields in the listbox at run time.

The FORMAT string includes the pipe symbol (|), immediately preceding the header text, as in “|~MyHeader~.”

Resizeable

Check this box to specify that the user can resize the width of the columns at run time.

The FORMAT string includes the “M” character, immediately preceding the header text as in “M~MyHeader~.”

Underline

Check this box to add the underline style to the listbox text. In effect, this creates a bottom border for each row in the column, giving your listbox a spreadsheet or cell-like appearance.

The FORMAT string includes the underscore character, immediately preceding the header text, as in "_~My Header~."

Fixed

Check this box to specify that the column always remains visible in the listbox, even if other columns scroll.

The FORMAT string includes the "F" character, immediately preceding the header text as in "F~MyHeader~."

Last on Line

Checking this box specifies that the next field in the group will appear immediately below the current field. In effect, it stacks two or more fields below the group header.

Tip: **The field must be part of a group. See *Creating Column Groups*.**

The FORMAT string includes the "/" character, immediately preceding the header text as in "/~MyHeader~."

Locator

By default, the first field in a multi-column COMBO displays in the entry portion of the COMBO. Check the **Locator** box to specify that this field (instead of the first field) displays in the entry box portion of a multi-column COMBO control.

The FORMAT string includes the "?" character, immediately preceding the header text as in "?~MyHeader~."

Color

Check this box to allow conditional runtime colors for individual list items—that is, to conditionally override the default colors for individual list rows. The color information for each row is contained in four LONG fields that immediately follow the data field in the QUEUE. Assign the color value to the appropriate QUEUE field at runtime, and Clarion's runtime library does the rest.

See *Control Templates—BrowseBox Control* in the *Application Handbook* for information on specifying conditional BrowseBox colors, and see *FORMAT* in the *Language Reference*.

Adds an asterisk "*" to the FORMAT string.

Style

Check this box to allow conditional runtime fonts for individual list items—that is, to conditionally override the default fonts for individual list rows. The font (style) information for each row is contained in a LONG field that immediately follows the data field in the QUEUE. Assign the style value to the appropriate QUEUE field at runtime, and Clarion's runtime library does the rest.

See *Control Templates—BrowseBox Control* in the *Application Handbook* for information on specifying conditional BrowseBox colors, and see *FORMAT* in the *Language Reference*.

Adds a "Y" to the FORMAT string.

List box Formatter Appearance Tab

Color and Style

Use these prompts to set the default colors for all list rows and columns.

Text

To set the default color for normal (unselected) list text, type a valid color equate in this field, or press the ellipsis (...) button to select a color from the color dialog.

Adds an "E(*color*,,,)" to the FORMAT string.

Background

To set the default color for normal (unselected) list background, type a valid color equate in this field, or press the ellipsis (...) button to select a color from the color dialog.

Adds an "E(,*color*,,)" to the FORMAT string.

Selected Text

To set the default color for normal (unselected) list text, type a valid color equate in this field, or press the ellipsis (...) button to select a color from the color dialog.

Adds an "E(,,*color*,)" to the FORMAT string.

Selected Background

To set the default color for normal (unselected) list background, type a valid color equate in this field, or press the ellipsis (...) button to select a color from the color dialog.

Adds an "E(,,,*color*)" to the FORMAT string.

Default Style

Type the default style number. The style number sets the font typeface, size, style, and color for all list rows and columns.

Adds a "Z(*n*)" to the FORMAT string, where *n* is the style number.

Icon

None

Select this to display no icons in the column.

Normal

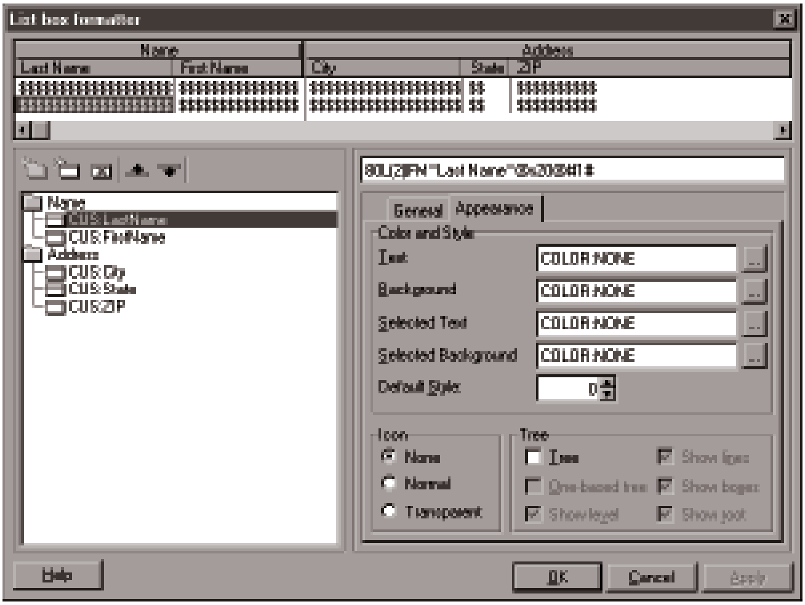
Select this to create an area to the left of the data in the column for displaying a normal image (.ICO) that you supply. See *Control Templates—BrowseBox Control* in the *Application Handbook* for information on specifying BrowseBox icons, and see *Prop:IconList* in the *Language Reference*.

Adds an “I” to the FORMAT string.

Transparent

Select this to create an area to the left of the data in the column for displaying a transparent image (.ICO) that you supply. See *Control Templates—BrowseBox Control* in the *Application Handbook* for information on specifying BrowseBox icons, and see *Prop:IconList* in the *Language Reference*.

Adds a “J” to the FORMAT string.



List box formatter

80L(2)|FM~Last Name~@s20@#1#

Name		Address		
Last Name	First Name	City	State	ZIP
$$$$$$$$$$$$$$$$$$$$	$$$$$$$$$$$$$$$	$$$$$$$$$$$$$$$$$$$$	$$	$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$	$$$$$$$$$$$$$$$	$$$$$$$$$$$$$$$$$$$$	$$	$$$$$$$$$$

- Name
  - CUS:LastName
  - CUS:FirstName
- Address
  - CUS:City
  - CUS:State
  - CUS:ZIP

General | Appearance

Color and Style

Text	COLOR:NONE
Background	COLOR:NONE
Selected Text	COLOR:NONE
Selected Background	COLOR:NONE
Default Style	0

Icon

● None
○ Normal
○ Transparent

Tree

□ Tree
☑ Show lines
□ One-based tree
☑ Show boxes
☑ Show level
☑ Show root

Tree

Tree

Checking this box displays this column in a hierarchical tree diagram. See *Application Handbook—Control Templates—Relation Tree* for more information. See also Relation Tree in the on-line help.

Adds a “T” to the FORMAT string.

One-based tree

Checking this box allows the root level to collapse, that is, all the items in the tree can collapse to a single line.

Adds a “(1)” to the “T” in the FORMAT string, resulting in “T(1).”

Show Level

Checking this box causes each descending level of the Tree hierarchy to be indented.

Clearing this box appends “(I)” to the “T” in the FORMAT string, resulting in “T(I).”

Show Lines

Checking this box adds connecting lines between related items in the tree diagram.

Clearing this box appends “(L)” to the “T” in the FORMAT string, resulting in “T(L)” to suppress lines.

Show Boxes

Checking this box adds expand (+) and contract (-) boxes to the tree diagram.

Clearing this box appends “(B)” to the “T” in the FORMAT string, resulting in “T(B)” to suppress boxes.

Show Root

Checking this box displays a root item for the tree diagram.

Clearing this box appends “(R)” to the “T” in the FORMAT string, resulting in “T(R)” to suppress display of a root item.

Creating Column Groups

Listbox groups contain two or more fields which share common formatting elements, such as a header, a scroll bar, or even the same vertical space (multi-row records).

To create a group, select a listbox column in the **List box Formatter,** then press the ▣ button. The selected field becomes the first member of the group; the next field you populatebecomes the next member of the group. Alternatively, use the ⬆ and ⬇ buttons to move fields into and out of an existing group. To delete a group, move or delete all fields out of the group.

From the **List box Formatter** dialog, you can specify a *group* header that appears above the *column* headers. The group header stretches across all the columns in the group.

You can use group headers to visually link related data in different columns. For example, you can link first and last name columns by placing “Name” in the group header, then “First” and “Last” in the column headers. You can also use a group header to break up the header text into two lines when the column header is extra long.

General Tab

This tab lets you set the following column group formatting options.

Heading Text

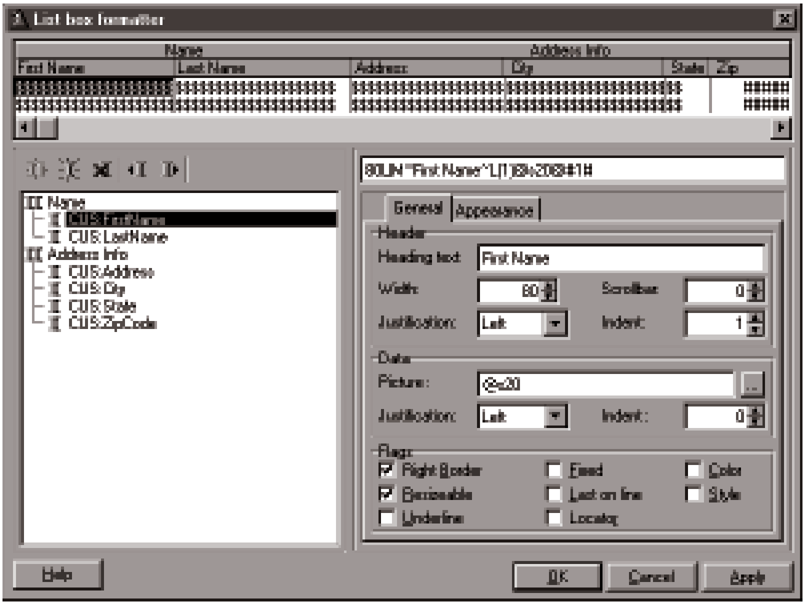
Optionally specify header text for the group. The header appears as a gray row above the listbox data items. To specify no header, leave this field blank. If any field included in the listbox has a header, a header appears over each field in the listbox; those fields with no header text will have a blank header.

Scroll Bar

Type a non-zero value to specify a horizontal scroll bar for this column only. If the overall listbox already has a scroll bar, the column scroll bar appears above the listbox scroll bar. The value pecifies, in dialog units, how far the column scrolls.

For example, if your *data* is fifty (50) characters, and your listbox *column* width is about forty (40) characters (one hundred sixty (160) *dialog units*), you should specify a value of fifty (50). Fifty (50) additional dialog units are enough to display the ten characters that extend beyond the width of the listbox column.

The scroll bar and size appear in the FORMAT string together, as in “S(4).”



List box formatter

Name		Address Info			
First Name	Last Name	Address	City	State	Zip
$$$$$$$$$$$$$$$$$$$$	$$$$$$$$$$$$$$$$$$$$	$$$$$$$$$$$$$$$$$$$$	$$$$$$$$$$$$$$$$$$$$	$$	#####
$$$$$$$$$$$$$$$$$$$$	$$$$$$$$$$$$$$$$$$$$	$$$$$$$$$$$$$$$$$$$$	$$$$$$$$$$$$$$$$$$$$	$$	#####

```
Name
  CUS:FirstName
  CUS:LastName
Address Info
  CUS:Address
  CUS:City
  CUS:State
  CUS:ZipCode
```

80L|M~First Name~L(1)@s20@#1#

General | Appearance

Header
Heading text: First Name
Width: 80 Scrollbar: 0
Justification: Left Indent: 1

Data
Picture: @s20
Justification: Left Indent: 0

Flags
☑ Right Border ☐ Fixed ☐ Color
☑ Resizeable ☐ Last on line ☐ Style
☐ Underline ☐ Locator

Justification

Choose from the drop-down list to specify left, right, center or decimal justification.

Indent

Optionally specify an indent, in dialog units, for the heading. This moves the data by the number of dialog units specified in the opposite direction to the justification. An indent of two (2) on left justification improves listbox readability. An indent of (2+4*decimal places) is appropriate for decimal justification.

Right Border

Check this box to specify column separators between fields in the listbox at run-time.

The FORMAT string includes the pipe symbol (|), immediately preceding the header text, as in "|~MyHeader~."

Resizeable

Check this box to specify that the user can resize the width of the group at run time.

The FORMAT string includes the "M" character, immediately preceding the header text as in "M~MyHeader~."

Underline

Check the **Underline** box to add the underline style to the listbox text. In effect, this creates a bottom border for each row in the group, giving your listbox a spreadsheet or cell-like appearance.

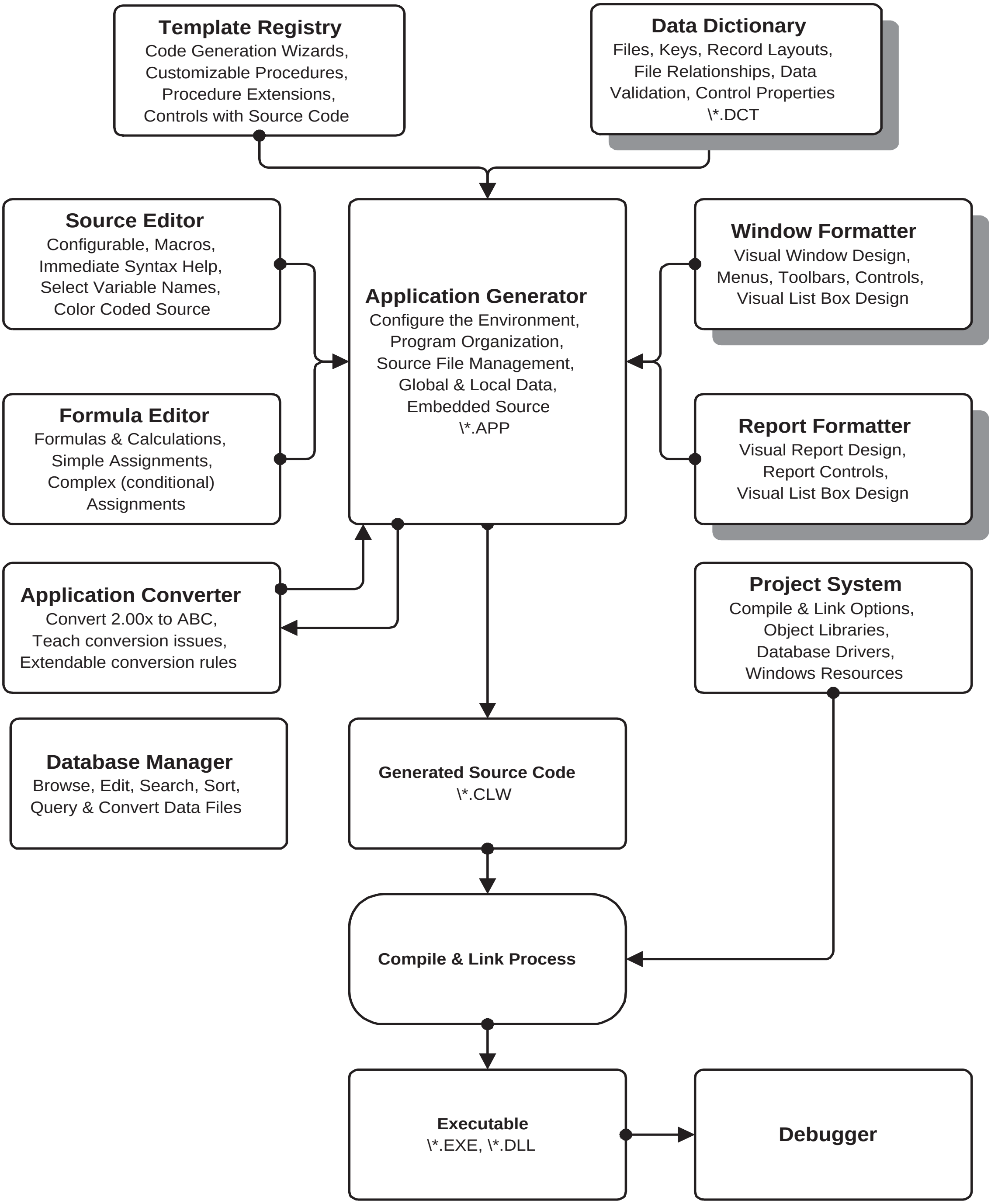
The FORMAT string includes the underscore character, immediately preceding the header text, as in "_~My Header~."

Fixed

Check this box to specify that the group always remains visible in the listbox.

The FORMAT string includes the "F" character, immediately preceding the header text as in "F~MyHeader~."

5 - CONTROLS AND THEIR PROPERTIES


Template Registry
Code Generation Wizards, Customizable Procedures, Procedure Extensions, Controls with Source Code
Data Dictionary
Files, Keys, Record Layouts, File Relationships, Data Validation, Control Properties
*.DCT
Source Editor
Configurable, Macros, Immediate Syntax Help, Select Variable Names, Color Coded Source
Application Generator
Configure the Environment, Program Organization, Source File Management, Global & Local Data, Embedded Source
*.APP
Window Formatter
Visual Window Design, Menus, Toolbars, Controls, Visual List Box Design
Formula Editor
Formulas & Calculations, Simple Assignments, Complex (conditional) Assignments
Report Formatter
Visual Report Design, Report Controls, Visual List Box Design
Application Converter
Convert 2.00x to ABC, Teach conversion issues, Extendable conversion rules
Project System
Compile & Link Options, Object Libraries, Database Drivers, Windows Resources
Database Manager
Browse, Edit, Search, Sort, Query & Convert Data Files
Generated Source Code
*.CLW
Compile & Link Process
Executable
*.EXE, *.DLL
Debugger

Overview

About This Chapter

This chapter shows you how to set control properties. It assumes you understand how to use the **Window Formatter** to choose, place, and size controls (see *Window Formatter*).

It provides an overview of the types of controls as they relate to data entry, discusses the properties applicable to the controls, then covers each control type individually. It also shows you how to associate the contents of a variable with an entry or display control.

Some of the specific tasks covered in this chapter include:

- How to set control properties with the Data Dictionary.
- How to customize a button with text, picture, or both.
- How to specify radio button and check box properties, including customizing them with a 3D look suitable for toolbars.
- How to create an entry control and how to associate it with a variable that holds the data the user enters.
- How to specify spin box control properties, such as the increment value when the user presses the increase or decrease buttons.
- How to specify group, sheet, and tab control properties, which visually organize related controls in a window.
- How to format list box controls, including how to create multi-column list boxes, and hierarchical lists.
- How to include graphic controls such as bitmaps, metafiles, lines, boxes, and ellipses in a window.

Setting Control Properties with the Data Dictionary

Control properties may be set for a single control using the **Window Formatter**. Better still, with the Data Dictionary, you may set control properties for *every control associated with a specific database field* (press the **Properties** button on the **Window** and **Report** tabs of the **Field Properties** dialog). When control properties are set in this manner (with the Data Dictionary), the Application Generator applies these properties every time you place the field on a window or report and every time you synchronize your application and data dictionary.

Types of Controls

For this discussion, we will divide controls into three categories: Interactive Controls, Non-Interactive Controls, and Custom Controls.

Interactive Controls

Interactive controls are clicked on or typed into by the user.

- Action controls—BUTTON—lead to an instantaneous result. For example, closing a dialog box and completing its pending operations. Clarion also supports associating a continuous action with a button—pressing the button and holding it down is the same as clicking the button repeatedly.
- User Choice controls—CHECK, OPTION, RADIO, COMBO, SPIN and LIST—let the user enter data by choosing from a finite group of alternatives. No keyboard input is required. They create streamlined user entry since it is usually faster to pick an item from a list than to type the name of an item you may not remember. Use choice controls to force the user to choose only one of a set of mutually exclusive selections—create "latched" buttons that stay depressed until pressed again, or groups of radio buttons where only one button can be selected at a time.
- Entry controls—ENTRY and TEXT—allow data entry from the keyboard. Clarion provides extensive options for automatically validating user data entry, plus supports Windows standard Cut, Copy, and Paste operations for these text fields.
- Hybrid controls—SHEETs, TABs and REGIONs—interact with the user by guiding them to other controls, and by generating events that your program can detect and act upon.

Non-Interactive Controls

Non-interactive controls provide visual cues that help the user understand and operate the interactive controls.

- Static controls—PANEL, PROMPT, GROUP, BOX, ELLIPSE, LINE and IMAGE—don't perform an action, but instead guide the user to other controls or otherwise provide visual candy. They can take the form of a group box, a line, or a graphic image, all of which visually organize or emphasize other controls. GROUP controls also help you develop and maintain your application by letting you apply common attributes such as positioning, enabling, or hiding to several controls at once.

Custom Controls

Custom controls are defined outside the Clarion Development Environment and may be either interactive or non-interactive. Custom controls are discussed in the following chapter.

Common Control Attributes

The attributes you add to a control determine how the control looks and acts. Different controls support different functions, and so require different attributes. However, all Clarion controls allow you to set two common attributes: USE and AT. Additionally, most controls allow you to set TEXT, COLOR, KEY, ALRT, FONT, SKIP, HIDE, DISABLE, SCROLL, CURSOR, HLP, MSG, and TIP attributes. This section explains how to set these common control attributes. Each attribute is discussed fully in the *Language Reference*.

Tip: ALT+DOUBLE-CLICK a control to access its properties.

Setting the USE Attribute

The **Use** field takes either a Field Equate Label or a variable label. If you placed a control template, you can accept the default label or you may specify your own label.

Field Equate Labels

Use a Field Equate Label when you *don't* need to assign a value from the control to a variable. A field equate label is a valid Clarion label prefixed by a question mark(?). This label references the control within your source code (see the *Language Reference* for more information). For example:

```
HIDE(?MyButton)       !hides the control with USE attribute ?MyButton
```

Variable Labels

Use a variable label when you *do* need to assign a value from the control to a variable or vice versa.

Tip: Some controls, such as PROMPTs and LINEs, cannot accept variable values and therefore only accept Field Equate Labels as their USE attribute.

The variable must be declared in your program, module, or procedure. Press the ellipsis (...) button in the control's properties dialog to declare the variable or to select the variable from those already declared.

The variable label automatically serves as the field equate label for the control too! For example:

```
MESSAGE('My Variable = '& MyVar)  !displays the variable MyVar
HIDE(?MyVar)                      !hides the ?MyVar entry control
```

Duplicate Field Equate Labels

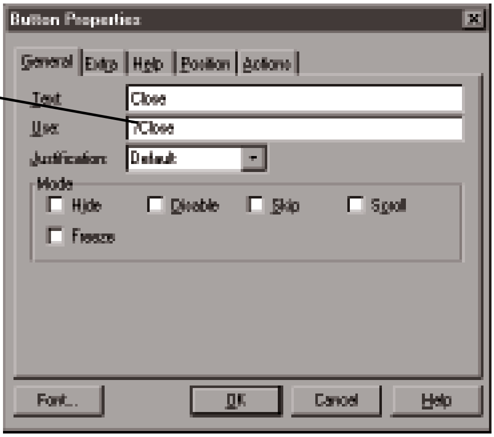
Two or more entry controls may update the same variable. However, they cannot have the same field equate label. In this circumstance, the **Window Formatter** automatically creates unique field equate labels by appending a sequence number to the field equate labels that would otherwise be duplicated. The unique field equate label is specified by the third parameter of the USE attribute (see USE in the *Language Reference*). For example, if you place three controls on a single window that all update the same variable, the **Window Formatter** generates code something like this:

```
window  WINDOW('Caption'),AT(,,183,119),GRAY
          SPIN(@s20),AT(66,27),USE(MyVar)
          COMBO(@s20),AT(66,50),USE(MyVar,,?MyVar:2)
          ENTRY(@s20),AT(67,7),USE(MyVar,,?MyVar:3)
        END
```

To set the USE attribute

1. RIGHT-CLICK the control, then choose **Properties** from the popup menu.

   This displays the **General** tab of the respective control properties dialog, which lets you specify the USE attribute for your control.



Text: Close

Use: ?Close

Justification: Default

☐ Hide ☐ Disable ☐ Skip ☐ Scroll

☐ Freeze

Setting the USE attribute for your control, so you can refer to the control by name in your source code.

2. If the control is an entry control or has an associated variable (CHECK, COMBO, CUSTOM, ENTRY, LIST, OPTION, PROGRESS, SHEET, SPIN, STRING, or TEXT), press the **Use** field ellipsis (...) button to select or define a data dictionary field or memory variable from the **Select Field** dialog.

   If the control does not have an associated variable (BOX, BUTTON, ELLIPSE, GROUP, IMAGE, LINE, PANEL, PROMPT, RADIO, REGION, or TAB), type a valid Clarion label prefixed with a question mark (?). Notice there is no **Use** field ellipsis (...) button for these controls.

   Remember, you can always use the default label supplied by the **Window Formatter**.

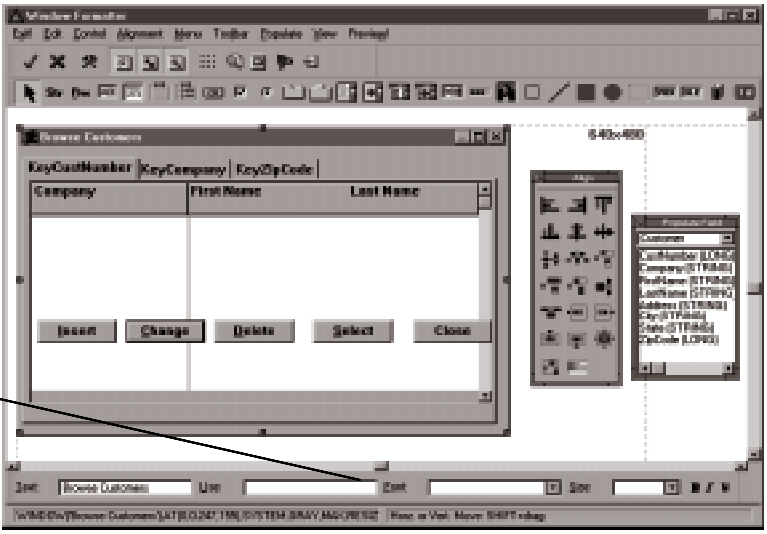
3. Press the **OK** button.

> **Tip:** **Field equate labels and Clarion's property syntax let you modify the control at run-time. For example, you can use the DISABLE statement to "dim out" controls in situations when they should be unavailable to the user:**
>
> ```
> DISABLE(?MyList)
> ```

Following is an alternative method for setting the USE attribute. This method works best for controls with no associated variable.

1. From the **Window Formatter** menu, choose **Options ➤ Show Propertybox**.

   This displays the **Property** toolbox, which lets you specify the USE attribute for your controls.



Alternatively, use the Property toolbox to set the USE attribute.

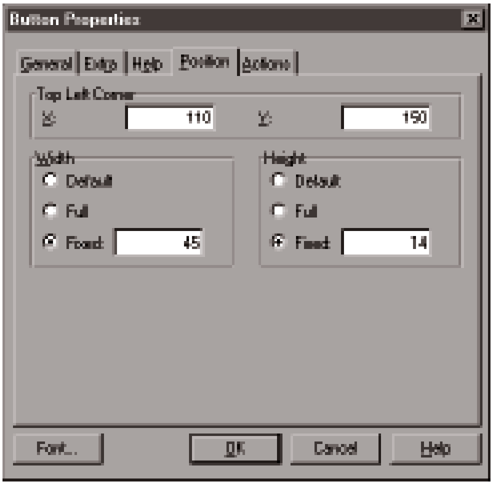
2. CLICK the control you want to change.
3. In the **Property** toolbox **Use** field, type a Field Equate Label or the label of a variable to use with the control.

Setting the AT Attribute

The AT attribute defines the control's position and size. The **Window Formatter** lets you visually set the AT attribute of each control simply by dragging it wherever you want. You may also specify position and size by manually typing coordinates in the control's properties dialog and by using the Alignment tools. To set the AT attribute, which defines the control's position and size:

1. RIGHT-CLICK on the control, then choose **Position** in the popup menu.

This displays the **Position** tab of the respective control properties dialog.



Top Left Corner
X: 110 Y: 150

Width
○ Default
○ Full
● Fixed: 45

Height
○ Default
○ Full
● Fixed: 14

2. Specify coordinates for the top left corner of the control.

Type in an '**X**' (horizontal) and '**Y**' (vertical) coordinate. This places the top left corner of the control relative to the top left corner of the window. The unit of measurement for the coordinates is the dialog unit. See the *Glossary* for the definition of dialog units. These provide a relative measure based on the size of the character set currently in use.

3. Specify **Width** and **Height** options.

Choose **Default** and Clarion automatically selects a size based on the display picture on an entry control. Or choose **Fixed** and type in width and height values for the control in dialog units.

Tip: For IMAGE controls, Default displays the picture at the size it was created.

You may also specify that the control fills the window by choosing the **Full** options. This adds the FULL attribute to the control. See the *Language Reference*for more information on this attribute.

Note: Default and Full are mutually exclusive choices. For example, you cannot set Width to Full and Height to Default.

Tip: You can provide your users a full window text editor for MEMO fields. Create a window and place a TEXT control in it. Optionally change the cursor to an I-Beam, then set the Width and Height of the TEXT control to Full.

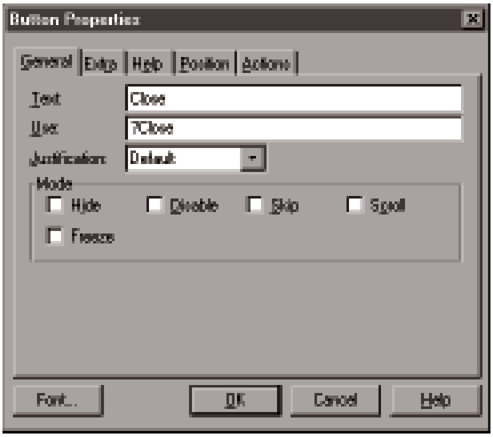
Setting the Text Attribute

Many controls, such as BUTTONs, TABs, CHECKs, GROUPs, etc., display constant text on their face. This is the most straightforward and common method for telling the user how and when to use the control.

To set the text attribute:

1. RIGHT-CLICK on the control then choose **Properties** from the popup menu.

   This displays the **General** tab of the respective control properties dialog, which lets you specify the text attribute for your control.



Text: Close

Use: ?Close

Justification: Default

Mode

□ Hide □ Disable □ Skip □ Scroll

□ Freeze

2. In the **Text** field, type the text to display.

   An ampersand (&) within the text means the next character is the mnemonic key for the control. When displayed, the character is underlined, and when the user presses ALT + the mnemonic key, the control's action initiates. For example type *&Print* to display **Print** and to let ALT +P initiate the control's action.

Tip: Microsoft recommends you do *not* place a mnemonic key on buttons labeled 'OK,' or 'Cancel.'

3. Press the **OK** button.

Tip: Field equate labels allow you to use executable statements and Clarion's property syntax to modify the text of the control at run-time. For example, you can change text on the fly with:

```
?MyButton{PROP:Text}='new text'
```

Following is an alternative method for setting the text attribute.

1. From the **Window Formatter** menu, choose **Options** ➤ **Show Propertybox**.

This displays the **Property** toolbox, which lets you specify the text attribute for your controls.

2. CLICK on the control you want to change.
3. In the **Property** toolbox **Text** field, type the text to display.

Setting the Display Picture

Many controls, such as ENTRYs, COMBOs, STRINGs, etc., display variable values as well as constant text. These controls provide a **Picture** field instead of the **Text** field.

To set the display picture for variable values, type a picture token in the **Picture** field or press the ellipsis button to use the **Edit Picture String** dialog (see the *Picture Editor* chapter). See the *Language Reference* for more information on pictures. See also *Dictionary Editor—Defining Field Properties*.

Setting the COLOR Attribute

The sparing use of color can improve the look and functionality of your program. See *Windows Design Issues* for more information on the use of colors in the Windows environment.

Enter a valid color value in any of the following fields to add the COLOR attribute to your control declaration. See ..\LIBSRC\EQUATES.CLW for a list of valid color equates.

Text Color
To apply a specific color to the control text, type a valid color equate in this field, or press the ellipsis (...) button to select a color from the **Text Color** dialog.

Background
To apply a specific color to the entire control, except for selected text, type a valid color equate in this field, or press the ellipsis (...) button to select a color from the **Background Color** dialog.

Selected text
To apply a specific color to the control's selected text, type a valid color equate in this field, or press the ellipsis (...) button to select a color from the **Selected Color** dialog.

Selected fill
To apply a specific color to the background of control's selected text, type a valid color equate in this field, or press the ellipsis (...) button to select a color from the **Selected Background Color** dialog.

Grid Color (LIST and COMBO only)

To apply a specific color to the LIST's grid lines, type a valid color equate in this field, or press the ellipsis (...) button to select a color from the **Grid Color** dialog.

Border Color (BOX, ELLIPSE, and REGION only)

To apply a specific color to the control's border, type a valid color equate in this field, or press the ellipsis (...) button to select a color from the **Border Color** dialog.

Fill Color (BOX, ELLIPSE, REGION, and PANEL only)

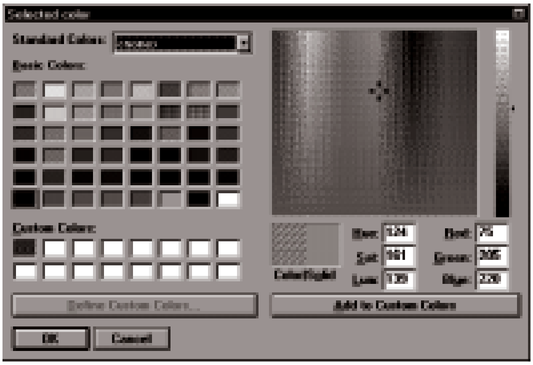
To apply a specific color to the control's interior, type a valid color equate in this field, or press the ellipsis (...) button to select a color from the **Fill Color** dialog.

The Color Dialog

You invoke the **Color** dialog from several places within the Clarion environment, but primarily when setting color for a window or for a window control. The **Color** dialog is a Microsoft product, but is documented here for your convenience.

Standard Colors

Choose from a drop-down list of the standard colors that Clarion uses to create its default Windows standard application interface. These colors match the default colors used by Clarion's template generated procedures.


Selected color

Hue: 124
Sat: 161
Lum: 139
Red: 75
Green: 205
Blue: 220

Basic Colors

Choose from 48 predefined color samples. CLICK on the color you want then press the **OK** button.

Custom Colors

Choose from 16 custom color samples that you define. CLICK on the color you want then press the **OK** button.

Define Custom Colors

To define a custom color, CLICK on one of the 16 Custom Color sample boxes, then press the **Define Custom Colors** button. The Color dialog expands so you can define the custom color.

Color Continuum Pad
Displays a continuum of color choices. click on this pad to set the gross definition for your custom color. Fine tune your color definition with the controls described below.

Luminance Continuum Slider
Displays a continuum of luminance choices. click and DRAG inside the elongated rectangle to adjust the luminance of the color from darkest (black) to lightest (white).

Color|Solid Displays a sample of the currently defined (mixed) color and its nearest solid color equivalent. To convert the currently defined color to its nearest solid color equivalent, type **Alt+O**. The conversion automatically adjusts the values of the six components listed below to make the appropriate solid color equivalent.

Hue An integer from 0 to 240 representing the hue.

Sat An integer from 0 to 240 representing the saturation.

Lum
An integer from 0 to 240 representing the luminance.

Red An integer from 0 to 255 representing red.

Green
An integer from 0 to 255 representing green.

Blue
An integer from 0 to 255 representing blue.

Add to Custom Colors
When you are satisfied with the custom color definition press this button.

Setting the KEY Attribute

The KEY attribute applies to any control which may receive focus (Combo Box, Entry Box, Group Box, List Box, Option Box, Property Sheet, Spin Box, Tab, Text Field, Button, Check Box, Custom Control, and Radio Button). It specifies a hot key which gives immediate focus to the control. For an action control, such as a command button, the hot key initiates the action as well. See the *Language Reference* for more information.

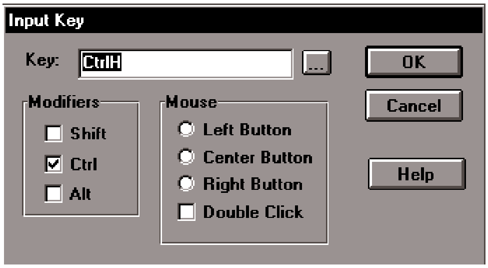
To set the KEY attribute:

1. RIGHT-CLICK on the control, and choose **Key** in the popup menu.

   This opens the **Input Key** dialog, which lets you specify the KEY attribute for your control.

2. Press the desired key or key combination.

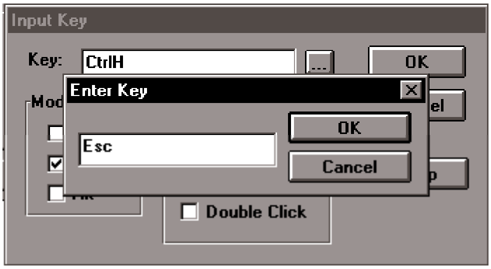
The key or key combination you press appears in the **Key** field, and is used as the parameter to the KEY attribute for this control. Alternatively, press a character or function (F1, F2, etc.) key and check a combination of the **Ctrl, Shift**, or **Alt** boxes to specify a hot key combination.



Key: CtrlH

□ Shift
☑ Ctrl
□ Alt

Mouse clicks may be used as hot keys; however, mouse clicks *cannot* be specified by clicking the mouse. For mouse clicks, check the corresponding box(es). For example, to give focus to a control when the user ALT+DOUBLE-CLICKS, check the **Alt** box, the **Left Button** box, *and* the **Double Click** box.



Esc

The ESC, ENTER, and TAB keys *cannot* be specified by simply pressing them, because these keys are standard Windows navigation keys. For these keys, press the ellipsis (...) button and type "esc," "enter," or "tab."

3. Press the **OK** button.

Tip: Avoid using ALT plus letter combinations as hot keys. These combinations should be reserved for menu accelerator keys.

Setting the ALRT Attribute

The ALRT attribute applies to any control which may receive focus. It specifies an alert key which is enabled when the control has focus. When the user presses an alerted key, it generates an EVENT:AlertKey. This lets you execute an action while the user is still in the entry field. For example, you may set an ALRT to display additional information to the user upon a function key press. See the *Language Reference* for more information.

To set the ALRT attribute:

1. RIGHT-CLICK on the control, then choose **Alert** in the popup menu.

   This opens the **Alert Keys** dialog, which manages the ALRT attributes for your control. You may set as many alert keys as you like for a control.

2. Press the **Add** button.

   This opens the **Insert Key** dialog, which lets you specify the ALRT attribute for your control. This is the same dialog used to specify the KEY attribute. See *Key* above for information on how to use this dialog.

3. Press the **OK** button.

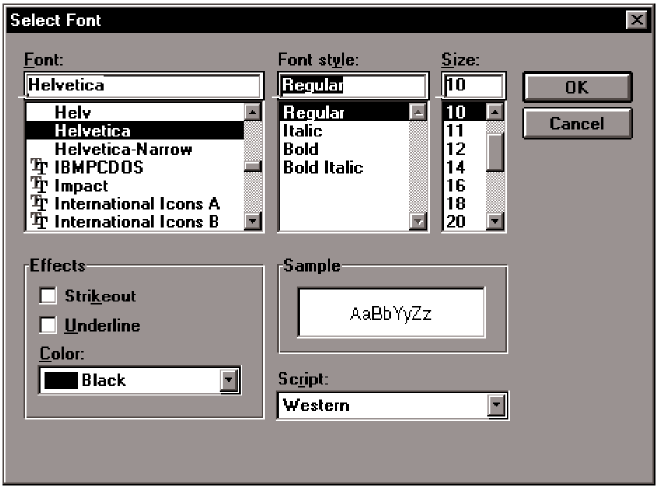
Setting the FONT Attribute

You may specify the appearance of the text displayed on a control. Select typeface, size, color, style, and script from standard drop-down lists. Choose strikeout and underline effects too. See *FONT* the *Language Reference* for more information.

To set the FONT attribute:

1. RIGHT-CLICK on the control and choose **Font** in the popup menu.

   This displays the **Select Font** dialog.



Select Font

Font: Helvetica

Font style: Regular

Size: 10

Effects
☐ Strikeout
☐ Underline

Color: Black

Sample: AaBbYyZz

Script: Western

2. Select typeface, size, color, style, and script from standard drop-down lists.

   The dialog displays a sample of the text design you have chosen.

4. Check the **Strikeout** or **Underline** boxes.

5. Press the **OK** button.

Tip: Be sure the font you pick is present on the user's system. If not, Windows will try to substitute an equivalent font; however, since you have no control over the substitution, you cannot be sure of the result.

Following is an alternative method for setting the FONT attribute:

1. From the **Window Formatter** menu, choose **Options ➤ Show Propertybox**.

   This displays the **Property** toolbox, which lets you specify the FONT attribute for your controls.

2. CLICK on the control you want to change.
3. In the **Property** toolbox, select font typeface and size from standard drop-down lists.
4. In the **Property** toolbox, select font style with standard bold, italic, and underline buttons.

Setting Control Modes

The **General** tab of the various control properties dialogs lets you set several attributes that control the "mode" of your window controls. To set the control's mode:

1. RIGHT-CLICK on the control, and choose **Properties** in the popup menu.

   This displays the **Properties** dialog for the selected control.

2. Select the **General** tab.

   This displays the **General** tab which contains the **Mode** check boxes.

3. Check any combination of the **Mode** boxes.

   The choices and their effects are:

   Skip

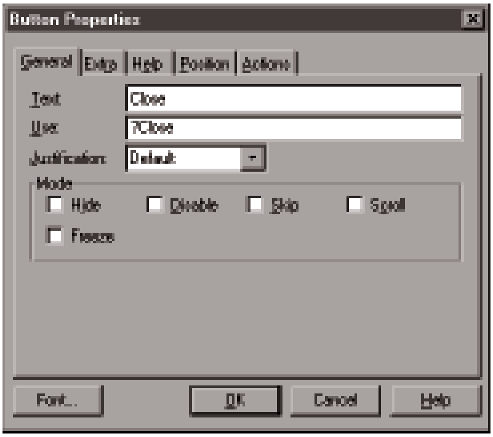
   Instructs the **Window Formatter** to omit the control from the Tab Order (the order in which controls get input focus as the user presses the TAB key). When the user TABS from field to field in the dialog box, Windows will not give the control focus. This is useful for seldom-used controls, because the user can still access the control by CLICKING on it. The **Window Formatter** places the SKIP attribute on the control (see the *Language Reference*).

   Disable

   Disables (or dims) the control when your program initially displays it, so it is unavailable to the user. The **Window Formatter** places the DISABLE attribute on the control. You can use the ENABLE statement to allow the user to access the control (see the *Language Reference*).

Hide

Makes the control invisible at the time Windows would initially display it. Windows actually creates the control — it just doesn't display it on screen. The **Window Formatter** places the HIDE attribute on the control. You can use the UNHIDE statement to display the control (see the *Language Reference*).



Text: Close

Use: ?Close

Justification: Default

Mode

□ Hide □ Disable □ Skip □ Scroll

□ Freeze

Scroll

Specifies whether the control should remain in the window when the user scrolls the window. By default (unchecked), the control *remains in the window*. Check the **Scroll** box to create a control that can be "scrolled off" the window. The **Window Formatter** places the SCROLL attribute on the control (see the *Language Reference*).

Transparent

Specifies whether the control is drawn on the window, or only its text or icon. By default (unchecked), the control is drawn. Check the **Transparent** box to create a control that is invisible, except for its text or icon. The **Window Formatter** places the TRN attribute on the control (see the *Language Reference*).

Freeze

"Freezes" the control, and all its children, so that subsequent data dictionary changes are not applied (see *Synchronize*). You can override the #Freeze attribute for all controls, or for individual controls. See *Application Generator—Configuring the Application Generator*.

4. Press the **OK** button.

Setting Help Attributes

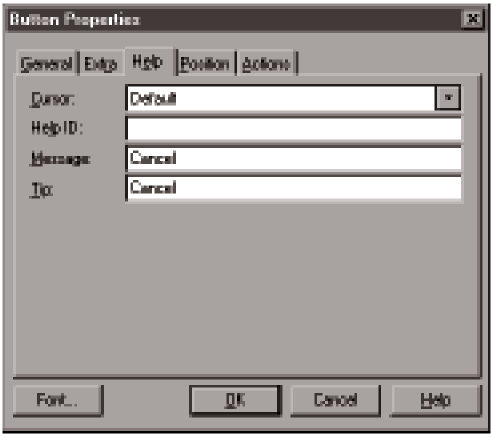
The **Help** tab of the various control properties dialogs lets you set several attributes that supply information to the user about the control.

1. RIGHT-CLICK on the control, and choose **Properties** in the popup menu.

   This displays the **Properties** dialog for the selected control.

2. Select the **Help** tab.

   This displays the **Help** tab which contains cursor, help, and message entry boxes.



Cursor: Default
Help ID:
Message: Cancel
Tip: Cancel

3. Optionally fill in any of the four entry fields.

   The fields and their effects are:

Cursor
Lets you specify an alternate shape for the cursor when the user passes it over the control. The **Cursor** drop-down list provides standard cursor choices such as **I-Beam** and **Crosshair**. To select an external cursor file (whose extension must be .CUR), choose **Select File...** from the drop-down list, then pick the file using the standard file dialog. The **Window Formatter** places the CURSOR attribute on the control (see the *Language Reference*).

Tip: The I-Beam, which signals text entry, is an excellent choice for the active cursor for an entry or text control.

Help ID
Sets the HLP attribute for a control (see the *Language Reference*). When the control has focus and the user presses F1, the Windows Help file opens to the topic referenced by the HLP attribute. In the **Help ID** field, type either a help keyword or a help context string present in a .HLP file.

A Help keyword is a word or phrase indexed so the user may search for it in the Help Search dialog. If more than one topic matches a keyword, the search dialog appears.

A Help context string is the arbitrary string which uniquely identifies each topic page for the Windows Help Compiler. A help context string must be prefixed with a tilde (~).

Tip: **When authoring a Windows Help file, you indicate a keyword with the 'K' footnote. A Help context string is the arbitrary string which uniquely identifies each topic page for the Windows Help Compiler. When creating the Help file, the '#' footnote marks a context string. These tasks are all done for you by many help authoring tools.**

Message

Sets the MSG attribute for the control (see the *Language Reference*). The MSG attribute specifies text to display in the first zone of the status bar when the control has focus. In the **Message** field, type the text to display in the status bar.

Tip Sets the TIP attribute for the control (see the *Language Reference*). TIP displays text in a small box near the cursor when the cursor is idle on the control for a specified period. The default period is half a second. This technique is also known as "Balloon Help." In the **Tip** field, type the text to display in the tip box.

Interactive Controls

Button Properties

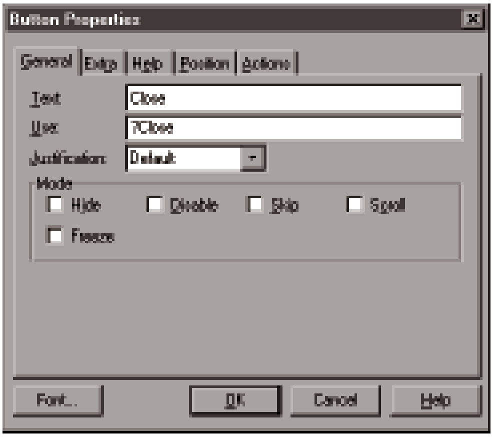
A BUTTON is a control that performs an *action* when the user presses it. In addition to the common control attributes described above, the **Window Formatter** lets you set the following button properties:

- The Button *text*.
- The Button *icon* or picture.
- The action to take *When Pressed*.
- The *STD ID* specifying a standard windows action for the button to take.
- Whether the button's action is the *default* action.
- The *Drop ID* specifying drag and drop operations for which the button is a valid target.

By convention, a button is a rectangular area containing text, picture, or both. When the user presses (CLICKS on) the button, it executes the command described by the text or picture.

To specify button properties, RIGHT-CLICK the button control and choose **Properties** from the popup menu. The **Button Properties** dialog appears. This dialog helps you to specify the attributes for the BUTTON statement.

General Tab



Text: Close
Use: ?Close
Justification: Default
Mode
□ Hide □ Disable □ Skip □ Scroll
□ Freeze

1. In the **Text** field, type the text to display on the button.

An ampersand (&) within the text means the next character is the mnemonic key for the button. When the user presses ALT + the corresponding key, the button's action initiates. See *Common Control Properties—Setting the Text Attribute*.

Tip: Microsoft recommends you do *not* place a mnemonic key on buttons labeled 'OK,' or 'Cancel.'

2. In the **Use** field, type a field equate label.

   A field equate label is a valid Clarion label, prefixed with a question mark (?). Use the field equate label to refer to the button in program statements. See *Common Control Properties—Setting the USE Attribute*.

3. Use the **Justification** drop-down list to position the button text relative to the button icon.

   Default places the icon above any text. **Right Justified** places the icon to the right of any text. **Left Justified** places the icon to the left of any text.

4. **Mode** options:

   See *Common Control Properties—Setting Control Modes.*

Extra Tab

1. In the **Icon** field, select a standard icon from the drop-down list.

   This displays a small bitmap on the button face in addition to any button text. To select a standard icon, choose one of the named items in the drop-down list. To select an icon file (extension .ICO), choose **Select File...** from the drop-down list, then pick the file using the standard open file dialog.

Tip: An icon and text gives a button with both. The text appears below the icon picture by default, however the Justification prompt provides alternative positioning.

2. Check the appropriate **Options** boxes.

   There are several button options which you may toggle on or off independently.

   Flat
   (the FLAT attribute) creates a button control which appears flat until the mouse cursor passes over it. This is typically used on toolbar buttons. This feature works best if the ICON attribute names a .GIF file.

   Required
   (the REQ attribute) specifies that, when pressed, your program automatically checks that all ENTRY controls with the REQ attribute are neither blank nor zero. A button with this attribute is a 'required fields check' button. Specify this type of button when a window also contains an ENTRY or TEXT control field with

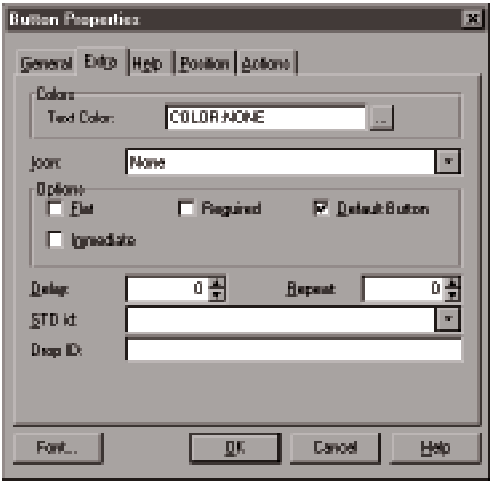
the REQ attribute (or else use the INCOMPLETE function to test the ENTRY controls). When the user presses a button with the REQ attribute and an ENTRY field is blank or zero, the first required control which is blank or zero receives focus.

Default Button

(the DEFAULT attribute), 'presses' the button when the user presses the ENTER key. A heavy border appears around the button to signal the default button to the user. In general, place the DEFAULT attribute on the button that represents the most likely action the user will take. Place only one default button in a window.

Immediate

(the IMM attribute) lets you create a button control which repeats the executable action continuously, for as long as the user holds the button down. Normally, buttons generate an event only after the user presses *and releases* the mouse.



Button Properties

Colors
Text Color: COLOR:NONE

Icon: None

Options
□ Flat □ Required ☑ Default Button
□ Immediate

Delay: 0 Repeat: 0
STD Id:
Drop ID:

3. For buttons with the IMM attribute, set the delay between first and second EVENT:Accepted event in the **Delay** box.

   This is how long the second EVENT:Accepted occurs in hundredths of seconds, after the first EVENT:Accepted when the end user clicks and holds the button.

4. For buttons with the IMM attribute, set the event generation rate for EVENT:Accepted in the **Repeat** box (the REPEAT attribute).

   This is how often the EVENT:Accepted occurs in hundredths of seconds when the end user clicks and holds the button.

5. In the **STD ID** field, optionally select a standard windows action from the drop-down list.

This is one way to tell your button what action to take. There are some actions you see in almost every windows program. For example, Cut, Copy, and Paste. Clarion provides an easy method for implementing these standard actions in your application—with the **STD ID** field.

Clarion automatically executes any of the standard actions from the drop-down list using standard windows behavior; you do not need any other support for it in your code. The STD ID equate labels and their associated actions are in the ..\LIBSRC\EQUATES.CLW file.

> **Tip: Do not combine a procedure or program call with an STD ID, because a control with an STD ID does not generate an event when the user activates the control.**

6. In the **Drop ID** field, optionally type up to sixteen (16) comma delimited signatures.

   The **Window Formatter** adds the DROPID attribute to your button. The DROPID indicates this button is a valid target for drag and drop operations. The signature is a string constant that identifies which types of drag and drop operations are valid for the button.

   Drag and drop capability means the user can select an item in one window or control, hold down the left mouse button, drag the item to another window or control, and release the mouse button, dropping the item onto the other window or control, which can then look at the item that was dropped and do something with it.

   Implementation of this capability requires that the source control have a DRAGID attribute with a signature that matches the target's DROPID signature, and that the procedures that drive each window have appropriate source code to process the drag and drop events. See the *Language Reference* for more details and examples.

Help Tab

See *Common Control Attributes—Setting Help Attributes*.

Position Tab

See *Common Control Attribute*s*—Setting the AT Attribute.*

Actions Tab

The **Actions** tab prompts come from the templates, in other words, the prompts you see here vary with the template used to create the control. Following are the standard action prompts for all button controls. See the *Control Templates* in the *Application Handbook* for more information on these and other prompts.

1. From the **When Pressed** drop-down list, choose **Call a Procedure**, **Run a Program**, or **No Special Action**.

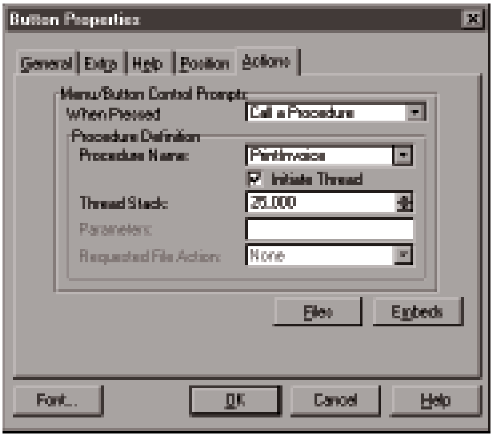
The procedure or program you specify executes when the user pushes the button. The choices are:

Call a Procedure

You must specify the **Procedure Name**, and whether the procedure **Initiates a Thread**.

Procedure Name Choose an existing procedure name from the drop-down list, or type a new procedure name. A new procedure appears as a "ToDo" item in your Application Tree.

Initiate a Thread Check this box to START a new thread and specify the **Thread Stack** size. If the procedure initiates a thread, you cannot specify **Parameters** or **Requested File Action**. If the procedure does not initiate a thread, you can specify **Parameters**, **Requested File Action**, or both.



Button Properties

Menu/Button Control Prompts
When Pressed: Call a Procedure
Procedure Definition
Procedure Name: PrintInvoice
☑ Initiate Thread
Thread Stack: 25,000
Parameters:
Requested File Action: None

When you START a procedure on its own thread, the procedure and its window operate independently of other threads in the same program; that is, the end user can switch focus between each execution thread at will. These are "**modeless**" windows.

If you don't initiate a new thread, the program behavior depends on whether the procedure's window has the MDI attribute. *A non-MDI* child window on the same thread as its parent, blocks access to all other threads in the program. This is an "**application modal**" window. When the application modal window closes, the other execution threads are available again. *An MDI* child window on the same thread as its parent, blocks access only to its parent window. When the MDI child window closes, its parent window regains focus.

Tip: A BUTTON on an application frame toolbar that calls an MDI child procedure must initiate a thread.

Thread Stack Accept the default value in the **Thread Stack** spin box unless you have extraordinary program requirements. To change the value, type in a new value or click on the spin box arrows.

Parameters In the **Parameters** field, optionally type a comma delimited list of variable labels to pass to the procedure.

Requested File Action

From the **Requested File Action** drop-down list, optionally select **None, Insert, Change, Delete**, or **Select**. The default selection is **None**. The GlobalRequest variable gets the selected value. The called procedure can then check the value of GlobalRequest and perform the requested file action.

Run a Program

You must specify the **Program Name**, and optionally, any parameters.

Program Name In the **Program Name** field, type the program name. The template generates:

```
RUN('MyPgm.exe')
```

Parameters In the **Parameters** field, optionally type a list of values that are passed to the program.The template generates:

```
RUN('MyPgm.exe MyParameter')
```

No Special Action

Choose this option if you are providing your button’s functionality with another method, such as embedded source, or an STD ID.

Tip: You may combine a procedure or program call with embedded source, but not with an STD ID.

2. Optionally press the **Files** button to access the file schematic for this procedure.

3. Optionally press the **Embeds** button to embed source code at points surrounding the event handling for this button only.

4. Press the **OK** button to return to the **Window Formatter**.

Radio Button Properties

A Radio Button, also called an option button, provides the user a set of mutually exclusive choices. For example, in a group of three buttons, only one can be selected at a time.

By default, Radio Buttons display as small circles; the selected button is filled. You can make Radio Buttons look like push buttons simply by adding icons to each button; the selected button remains depressed.

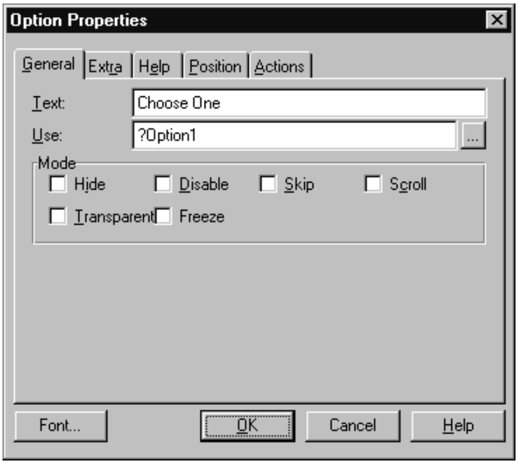
Relationship Between RADIOs and OPTIONs

An option box—an OPTION structure—must always surround the radio button choices. The **Window Formatter** automatically prompts you to create an option box if you try to place a radio button outside an option box. The option box appears at run time as a rectangle with a caption in the top border, and radio buttons inside. When you set radio button properties, you should also set the properties for the option box.

When the user selects a radio button, the OPTION’s USE variable receives a value indicating which button was selected: either the text of the selected button, the button number, or another value that you specify. Your program can then take appropriate action based on the value of the OPTION’s USE variable.

To place a radio button and an associated option box, CLICK the Radio Button tool, then CLICK in the sample window. The **Window Formatter** automatically adds an option box and a radio button.

General Tab—Option Box


Option Properties

Text: Choose One

Use: ?Option1

Mode

□ Hide □ Disable □ Skip □ Scroll

□ Transparent □ Freeze

1. In the **Text** field, type the text to display.

This string appears at runtime in the top border of the option box. See *Common Control Properties—Setting the Text Attribute*.

> **Tip: Though the OPTION structure *must* be present, it does not have to appear on screen. You may hide it from the user by clearing the 'Boxed' box on the 'Extra' tab of this dialog.**

2. In the **Use** field, type the label of a variable.

   The **Use** field (the USE attribute) takes the label of a variable. When the user selects a radio button, the OPTION's USE variable receives a value indicating which radio button is selected. When the USE variable is a string data type, it receives either the text of the selected button, or another string value that you specify (see *General Tab—Radio Button* below). When the USE variable is a numeric data type, it receives the button number.

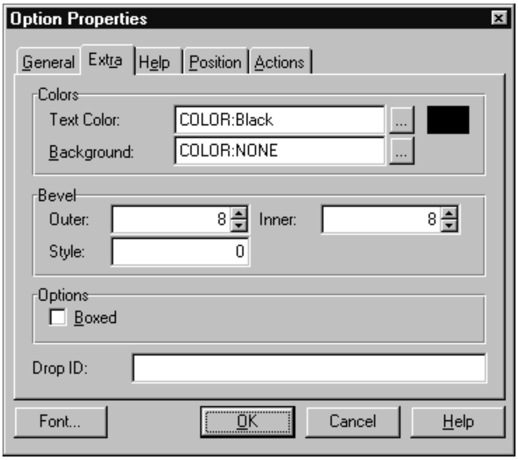
3. Check any combination of the **Mode** boxes.

   See *Common Control Properties—Setting Control Modes.*

Extra Tab—Option Box

1. Optionally, select a Text or Background color for the option.

   See *Common Control Attributes—Setting the COLOR Attribute.*



Option Properties

Colors

Text Color: COLOR:Black

Background: COLOR:NONE

Bevel

Outer: 8 Inner: 8

Style: 0

Options

□ Boxed

Drop ID:

2. Optionally, set the BEVEL attribute for the OPTION.

   The BEVEL attribute gives a three dimensional look to the option box. The box appears raised, depressed, or both.

   Outer

   A positive value makes the group box appear raised above the plane of the window. The higher the value, the further the box is raised. A negative value makes the group box appear depressed below the plane of the window. The bevel effect begins at the outer border of the box.

Inner

A positive value makes the group box appear raised above the plane of the window. The higher the value, the further the box is raised. A negative value makes the group box appear depressed below the plane of the window. The bevel effect begins immediately inside the outer bevel.

Style

A integer, whose sixteen bits define the style (but not the size) of the four edges of the OPTION. The STYLE attribute gives you very fine control over the bevel appearance. See the *Language Reference* for more information.

3. Optionally, clear the **Boxed** box to hide the option box, but not the radio buttons, from the user at run time.

   This produces a slightly different effect than the HIDE attribute. The HIDE attribute hides both the option box and the controls inside the box.

4. In the **Drop ID** field, optionally type up to sixteen (16) comma delimited signatures.

   The **Window Formatter** adds the DROPID attribute to your control. The DROPID indicates this control is a valid target for drag and drop operations. The signature is a string constant that identifies which types of drag and drop operations are valid for the button.

   Drag and drop capability means the user can select an item in one window or control, hold down the left mouse button, drag the item to another window or control, and release the mouse button, dropping the item onto the other window or control, which can then look at the item that was dropped and do something with it.

   Implementation of this capability requires that the source control have a DRAGID attribute with a signature that matches the target's DROPID signature, and that the procedures that drive each window have appropriate source code to process the drag and drop events. See the *Language Reference* for more details and examples.

Help Tab (Option Box)

See *Common Control Attributes—Setting Help Attributes*.

Position Tab (Option Box)

See *Common Control Attribute*s*—Setting the AT Attribute.*

1. Press the **OK** button to finish setting attributes for the OPTION box.

   The OPTION box appears in the sample window. If you cleared the **Boxed** option, it will be visible in layout mode, but will become invisible in **Preview!** mode.

Now that you have completed the option box properties, you should set the radio button properties; RIGHT-CLICK the radio button then select **Properties** from the popup menu; this opens the **Radio Button Properties** dialog.

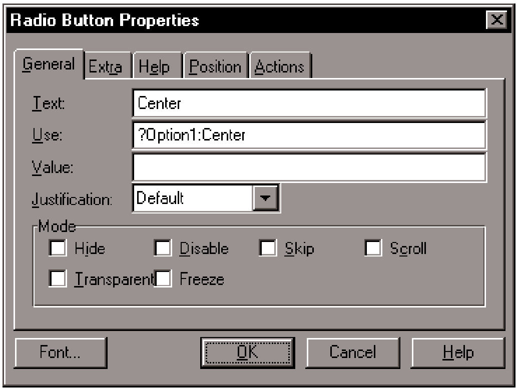
General Tab—Radio Button

1. In the **Text** field, type the text to display.

   See *Common Control Properties—Setting the Text Attribute*.

2. In the **Use** field, type a field equate label.

   A field equate label is a valid Clarion label, prefixed with a question mark (?). Use the field equate label to refer to the radio button in program statements. See *Common Control Properties—Setting the USE Attribute*.



Text: Center
Use: ?Option1:Center
Value:
Justification: Default
□ Hide □ Disable □ Skip □ Scroll
□ Transparent □ Freeze

3. Type a value in the **Value** field.

   When the user selects a radio button, the OPTION’s USE variable receives the value that you specify here. The value you enter should match the data type of the OPTION’s USE variable.

   If you leave the **Value** field blank, the OPTION’s USE variable receives either the string found in the **Text** field, or the button number, depending on the data type of the OPTION’s USE variable.

   The button number corresponds to the button’s position within the option box. From the **Window Formatter** choose **Edit ➤ Order Control dialog** to see the button’s tab order position within the option box.

4. From the **Justification** drop-down list, choose **Left Justification, Right Justification,** or **Default**.

   Left Justification arranges the button (or icon) to the left of the text. Right Justification arranges the button (or icon) to the right of the text. Default arranges the button according to any applicable settings in the data dictionary.

5. Check any combination of the **Mode** boxes.

   See *Common Control Properties—Setting Control Modes*.

Extra Tab—Radio Button

1. Optionally, select a **Text Color** and **Background** color for the Radio button.

   See *Common Control Attributes—Setting the COLOR Attribute.*

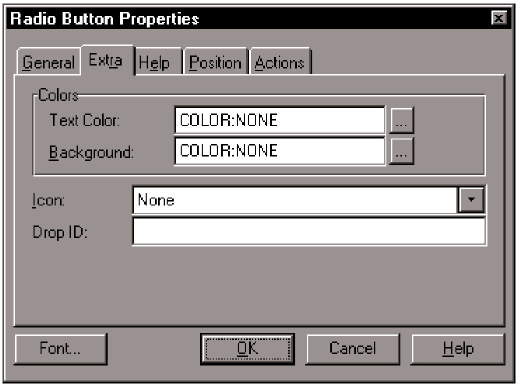
2. From the **Icon** drop-down list, optionally select a standard icon, or select a custom icon file.

   Adding an icon to a radio button makes the radio button look like a command button.

   Tip: When you require a set of buttons for the toolbar, only one of which can be active at a time, use radio buttons with the ICON attribute filled in.

   To select a standard icon, choose one of the named items in the drop-down list. To select an icon file (whose extension must be .ICO), choose **Select File...** from the drop-down list, then pick the file using the standard open file dialog.

   Tip: If you add an icon and text, you get a radio button with both! Make the resulting button large enough to display both.



Text Color: COLOR:NONE

Background: COLOR:NONE

Icon: None

Drop ID:

3. In the **Drop ID** field, optionally type up to sixteen (16) comma delimited signatures.

   The **Window Formatter** adds the DROPID attribute to your control. The DROPID indicates this control is a valid target for drag and drop operations. The signature is a string constant that identifies which types of drag and drop operations are valid for the button.

   Drag and drop capability means the user can select an item in one window or control, hold down the left mouse button, drag the item to another window or control, and release the mouse button, dropping the item onto the other window or control, which can then look at the item that was dropped and do something with it.

Implementation of this capability requires that the source control have a DRAGID attribute with a signature that matches the target's DROPID signature, *and* that the procedures that drive each window have appropriate source code to process the drag and drop events. See the *Language Reference* for more details and examples.

Help Tab—Radio Button

See *Common Control Attributes—Setting Help Attributes*.

Position Tab—Radio Button

See *Common Control Attribute*s*—Setting the AT Attribute.*

Tip: **To create professional looking radio button groups, turn the Grid control on and use the Alignment tools. Grid Settings appear on the Window Formatter's Options menu. They allow you to easily line up your radio buttons.**

Check Box Properties

A check box manages a variable that the user may turn on or off. Select the Check Box tool, or choose **Check Box** from the **Control** menu, then click in the sample window. The **Window Formatter** opens the **Select Field** dialog so you can choose or create a data dictionary field or memory variable to associate with the check box. Once placed, RIGHT-CLICK the check box and select **Properties** from the popup menu; the **Check Box Properties** dialog appears.

General Tab

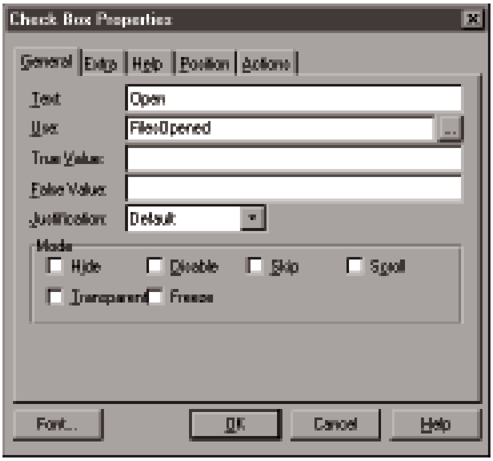
1. In the **Text** field, type the text to display.

   By default, the text appears immediately to the right of the check box. Use **Justification** to change this default. See *Common Control Properties—Setting the Text Attribute*.

2. The **Use** field should already contain a variable label.

   If not, type the label of a variable, or press the ellipsis button (...) to choose or create a data dictionary field or a memory variable with the **Select Field** dialog. Use a BYTE variable for best results.

Tip: **For best results, use a BYTE variable with a check box. Also, because the variable's value should be limited to exactly two values, you should update the variable with a check box or with a control that restricts input to the two valid values.**



Check Box Properties

Text: Open

Use: File:Opened

True Value:

False Value:

Justification: Default

Mode: □ Hide □ Disable □ Skip □ Scroll □ Transparent □ Freeze

Tip: **Give the variable an appropriate initial value, especially if you use True Value and False Value. See *Dictionary Editor—Adding or Modifying Fields*.**

3. Optionally, in the **True Value** field, type the value to assign when the box is checked. In the **False Value** field, type the value to assign when the box is cleared. These values are also used to determine the initial state of the check box (checked or cleared) when it is displayed.

   True Value and **False Value** let you easily manage legacy data with a check box, or let you use character values such as "T" and "F" or "Yes" and "No" where appropriate. For example, if your legacy field contains "True" and "False" or "Y" and "N," rather than 1 and 0, then **True Value** and **False Value** can modify the check box's default behavior to be consistent with the legacy data. If you leave both fields blank, you get the default values and behavior, that is, 1 for checked and 0 for cleared.

Tip: **True Value and False Value are case sensitive, so "True" is not the same as "TRUE" and "T" is not the same as "t."**

4. From the **Justification** drop-down list, choose from:

Left Justified	Arranges the check box (or icon) to the left of its text.
Right Justified	Arranges the check box (or icon) to the right of its text.
Default	Arranges the check box according to any applicable settings in the data dictionary.

5. **Mode** options:

   See *Common Control Attributes—Setting Control Modes.*

Extra Tab

1. Optionally, select a **Text Color** and **Background** color for the Check box.

   See *Common Control Attributes—Setting the COLOR Attribute.*

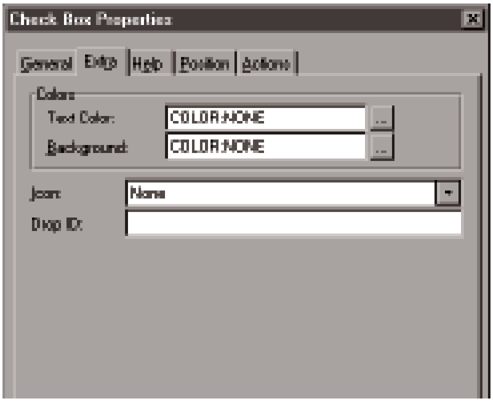
2. In the **Icon** field, select a standard icon from the drop-down list.

   Adding an icon to a check box converts the check box to a latched button. A latched button "stays depressed" when CLICKED, then returns to its original state when CLICKED a second time.

   The icon appears as a small bitmap on the button face in addition to any check box text.

   To select a custom icon file (extension .ICO), choose **Select File...** from the drop-down list, then pick the file using the standard open file dialog.

Tip: If you add an icon and text, you get a check box with both! Make the resulting button large enough to display both.



Text Color: COLOR:NONE
Background: COLOR:NONE
Icon: None
Drop ID:

3. In the **Drop ID** field, optionally type up to sixteen (16) comma delimited signatures.

   The **Window Formatter** adds the DROPID attribute to your check box. The DROPID indicates this check box is a valid target for drag and drop operations. The signature is a string constant that identifies which types of drag and drop operations are valid for the check box.

   Drag and drop capability means the user can select an item in one window or control, hold down the left mouse button, drag the item to another window or control, and release the mouse button, dropping the item onto the other window or control, which can then look at the item that was dropped and do something with it.

   Implementation of this capability requires that the source control have a DRAGID attribute with a signature that matches the target's DROPID signature, and that the procedures that drive each window have appropriate source code to process the drag and drop events. See the *Language Reference* for more details and examples.

Help Tab

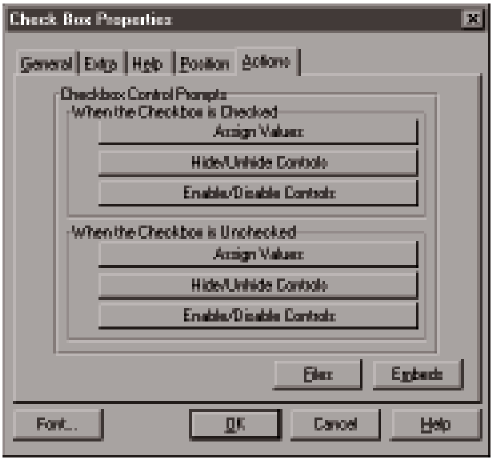
See *Common Control Attributes—Setting Help Attributes*.

Position Tab

See *Common Control Attributes—Setting the AT Attribute.*

Actions Tab

The **Actions** tab prompts come from the templates, in other words, the prompts you see here vary with the template used to create the control. Following are the standard action prompts for all check box controls.



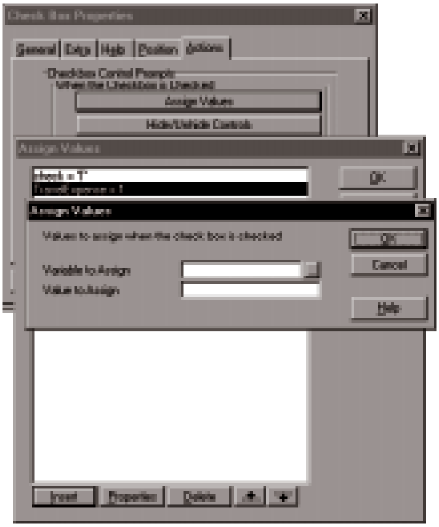
The **Actions** tab leads to dialogs allowing you to name variables (other than the USE variable) and change their values when the user checks or clears the box. Additionally, you can hide or unhide other controls in the window.

Two group boxes with two pairs of buttons appear on the **Actions** tab. These buttons set the behavior for **When the Check Box is Checked**, and **When the Check Box is Unchecked**.

1. Press the **Assign Values** button to open the **Assign Values** dialog.

   You may specify multiple variable assignments. Press the **Insert** button to add a new assignment. In the **Variable to Assign** entry box, type the variable name, or press the ellipsis (...) button to choose or create a data dictionary field or a memory variable with the **Select Field** dialog.

   In the **Value to Assign** entry box, type the value assigned to the variable. Press the **OK** button to end the dialogs.



2. Press the **Hide/Unhide Controls** button to open the **Hide/Unhide Controls** dialog.

You may specify multiple controls to hide/unhide. Press the **Insert** button to add a new hide/unhide action to the list. In the **Control to hide/unhide** drop-list, select the control’s equate label.

In the **Hide or unhide control** drop-list, select Hide or Unhide. Press the **OK** button to end the dialogs.

3. Press the **Enable/Disable Controls** button to open the **Enable/Disable Controls** dialog.

You may specify multiple controls to enable/disable. Press the **Insert** button to add a new enable/disable action to the list. In the **Control to Enable/Disable** drop-list, select the control’s equate label.

In the **Enable or Disable control** drop-list, select Enable or Disable. Press the **OK** button to end the dialogs.

4. Optionally press the **Files** button to access the **File Schematic Definition** dialog for this procedure.

5. Optionally press the **Embeds** button to embed source code at points surrounding the event handling for this check box only.

6. Press the **OK** button to return to the **Window Formatter**.

Check Box Behavior

The state of the check box (checked or cleared) affects the value of the associated variable and, conversely, the value of the associated variable affects the state of the check box, however, the effects may not be what you expect. This topic describes these effects.

Default Behavior (without True Value/False Value)

Numeric Fields

Assigning the Variable

Checking the box always assigns a value of 1 to the associated variable. Clearing the box always assigns a value of 0 to the associated variable.

Drawing the Check Box

Any non-zero value produces a checked box, whereas only a zero value produces a cleared box.

Character Fields

Assigning the Variable

Checking the box always assigns a value of 1 to the associated variable. Clearing the box always assigns a value of blank to the associated variable.

Drawing the Check Box

Any non-blank value produces a checked box, whereas only a blank value produces a cleared box.

Non-Default Behavior (with True Value/False Value)

Numeric Fields

Assigning the Variable

Checking the box assigns the value specified in the **True Value** field, if the value is numeric. If the value is not numeric, no assignment occurs. Clearing the box assigns the value specified in the **False Value** field, if the value is numeric. If the value is not numeric, no assignment occurs.

Drawing the Check Box

Any value except zero or the **False Value** produces a checked box, conversely, either a zero value or the **False Value** produces a cleared box.

Character Fields

Assigning the Variable

Checking the box always assigns the value specified in the **True Value** field. Clearing the box always assigns the value specified in the **False Value** field.

Drawing the Check Box

Any value except blank or the **False Value** produces a checked box, conversely, either a blank value or the **False Value** produces a cleared box.

Creating List Boxes

The LIST control is most useful for presenting a great number of choices to the user. It can display a large amount of data in a small area, which has led to its use as an all-purpose data control. Using Clarion you can create list boxes which look like spreadsheets, perform drag and drop tasks, and more.

Further, you may use Control templates (BrowseBox, FileDrop, etc.) to generate code that declares the LIST and *manages* it too (loads it, scrolls it, locates items, selects items for processing, etc.).

> **Tip: As you read this section, please be aware that all the information that applies to LIST controls also applies to COMBO controls.**

When you create a list box (whether a control template or a simple control), you define its data source, its format, and its functionality. The development environment divides these property definitions among three dialogs:

- The **List Properties** dialog specifies the queue that supplies the list data, the general scrolling capability, and whether the list is a drop-down list or a regular list. In other words, the **List Properties** dialog specifies all the properties of the list box that are not column-specific. This dialog is discussed in this chapter.

- The **List Box Formatter** dialog lets you add, delete, reorder, and resize the specific fields or columns that are displayed in the list box. In other words, the **List Properties** dialog specifies all the column-specific properties of the list box. See *Window Formatter—List Box Formatter*.

List (and Combo Box) Properties

1. From the **Window Formatter**, choose **Control ➤ List Box** from the menu (or choose **Populate ➤ Control Template** then select BrowseBox), then click in the window.

   This opens the **List Box Formatter** dialog. This dialog manages the columns or fields in your list box (see *Window Formatter—List Box Formatter)*.

2. Press the **OK** button to return to the **Window Formatter.**

3. RIGHT-CLICK the list box then select **Properties** from the popup menu.

   This opens the **List Properties** dialog.

General Tab

1. The **Use** field takes either a field equate label, or the label of a variable to receive the value the user selects from the list.

If you placed a control template, you can accept the default field equate label, or supply your own label.

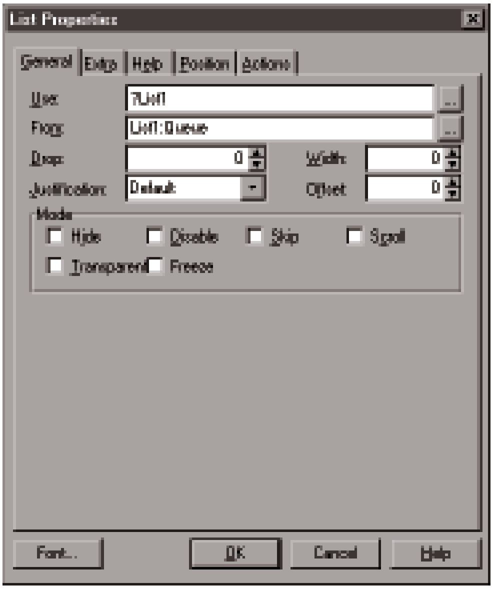
Use a field equate label when you *don't* need to assign the user's selection to a variable (for a LIST). This label references the control in your source code, for example:

```
HIDE(?MyList)
```

Use a variable label when you *do* need to assign the user's selection to a variable (for a COMBO). Press the ellipsis (...) button to select or declare a data dictionary field or memory variable. The variable label also serves as the field equate label for the list box! For example:

```
...
MESSAGE('You Selected '& MyList)  !displays the variable MyList
                                  !which contains the selected item
HIDE(?MyList)                     !hides the ?MyList control
...
```

See *Common Control Properties—Setting the USE Attribute*.



List Properties

Use: ?List1

From: List1:Queue

Drop: 0

Width: 0

Justification: Default

Offset: 0

Mode

□ Hide □ Disable □ Skip □ Scroll

□ Transparent □ Freeze

2. In the **From** field, supply the data source for the list.

Sets the FROM attribute for the LIST. See the *Language Reference* for more details. Generally, this is the label of a QUEUE structure, but may also be a field within a QUEUE or a string constant.

If you use a Control template or a Wizard to build your list box, the QUEUE label is supplied for you, as well as the code needed to declare and load the QUEUE.

3. In the **Drop** field specify the number of rows to "drop" the list.

Place a zero in the **Drop** field for a normal list box, with no drop-down. To create a drop-down list box, type the number of drop elements you wish to display.

4. In the **Width** field specify the width of the “dropped” elements in dialog units.

   This is useful for displaying a single field in the undropped list, but showing multiple fields in the dropped list. A **Width** of zero (0) sets the dropped elements to the same width as the undropped element. See *PROP:DropWidth* in the *Language Reference*.

5. From the **Justification** drop-down list, choose **Left Justified**, **Centered**, **Right Justified**, **Decimal**, or **Default**.

   Adds the LEFT, CENTER, RIGHT, or DECIMAL attribute to the LIST. See the *Language Reference* for details. **Left Justified**, **Centered**, and **Right Justified** position the list data predictably, left, center, or right justified in the list box. **Default** positions the data according to any applicable settings in the data dictionary. **Decimal** justification aligns values by their decimal points. Each justification may be **Offset** by a distance you specify.

   These attributes are superseded by the FORMAT attribute (generated by the **List Box Formatter**).

6. In the **Offset** field, specify a justification offset in dialog units.

   See the Glossary for the definition of dialog units. Sets the offset value for the LEFT, RIGHT, CENTER, and DECIMAL attributes. For CENTER justification, a negative value offsets to the left of center and a positive value offsets to the right of center.

   For DECIMAL justification, a negative value offsets to the left of the decimal and a positive value offsets to the right of the decimal.

Tip: For Decimal justification, use an offset equal to 4 * (decimal places + 1)

7. Check any combination of the **Mode** boxes.

   See *Common Control Properties—Setting Control Modes*.

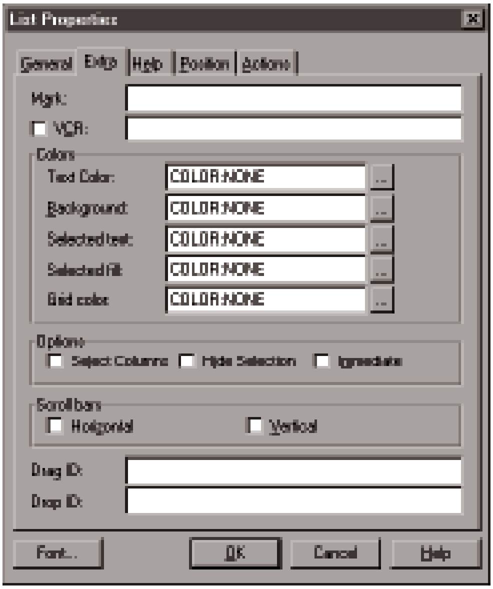
Extra Tab

1. In the **Mark** field, type the name of a QUEUE field.

   Clarion’s runtime library uses this field to let the user select or “mark” more than one item in the list.

Tip: For control template lists, look at the generated source code to find the name of the QUEUE field generated by the control template to store the marks.

The QUEUE field may be in the same QUEUE that displays in the LIST (the FROM attribute), or it may be in another QUEUE; however, it is your responsibility to ensure that the two QUEUES have corresponding, synchronized entries. The QUEUE field flags the selected items. Selected items get a '1', unselected items get a '0.' The field may be a numeric or character data type.



List Properties

General | Extra | Help | Position | Actions

Mark:

□ VCR:

Colors

Text Color: COLOR:NONE

Background: COLOR:NONE

Selected text: COLOR:NONE

Selected fill: COLOR:NONE

Grid color: COLOR:NONE

Options

□ Select Columns □ Hide Selection □ Immediate

Scrollbars

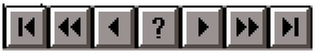
□ Horizontal □ Vertical

Drag ID:

Drop ID:

2. Check the **VCR** box to provide VCR scrolling buttons for your list box.

These special scrolling buttons provide the following functionality: Top of List, Page Up, Row Up, Locate, Row Down, Page Down, and Bottom of List.



Tip: VCR buttons provide a smoother, more reliable navigation mechanism than vertical scrollbars for very large datasets or skewed datasets presented in a page-loaded list such as the BrowseBox Control template. See *Control Templates—BrowseBox* in the *Application Handbook* for more information.

Note that the Locate (?) button is not meaningful in all LIST implementations. For example, to use Locate with the BrowseBox Control template, you must browse the file in key order and use an entry locator, not a step locator or an incremental locator. See *Control Templates—BrowseBox* in the *Application Handbook* for more information.

Tip: Use ?List1{PROP:VCR} = True to add just the navigation buttons and omit the locator button.

Do not confuse these VCR buttons, which only operate on a single LIST, with the similar FrameBrowseControl toolbar buttons which operate globally on BrowseBox and RelationTree Control templates and Form procedures with the FormVCRControls Extension template. See *Control Templates—FrameBrowseControl* in the *Application Handbook* for more information.

3. In the entry box to the right of the **VCR** check box (the VCR Locator Field), optionally type the field equate label of an entry control whose USE variable is the access key for the browsed file.

Tip: Use this field only with a simple LIST locator. The BrowseBox and FileDrop Control templates do not require this field.

The ENTRY control identified in the VCR Locator Field must be declared before the associated LIST control. Choose **Edit ➤ Set Control Order** to move the ENTRY before the LIST.

At runtime, the user types a value into the locator entry field, then presses the **Locate** button to scroll the list to the item that is nearest the value in the locator entry field.

4. Optionally, select colors for the List box.

See *Common Control Attributes—Setting the COLOR Attribute* above.

5. Optionally, check the **Select Columns** box to enable individual column selection in a multi-column list box.

Allows the user to highlight a multi-column list box field by field, rather than one row at a time. This provides for spreadsheet grid style movement of the highlight bar.

6. Optionally, check the **Hide Selection** box to place the NOBAR attribute on the LIST.

The NOBAR attribute specifies the currently selected element in the LIST is only highlighted when the LIST control has focus.

7. Optionally, check the **Immediate** box to place the IMM attribute on the LIST.

The IMM attribute generates an event whenever the user presses a key (such as the down arrow or page up key, or even the alphabet keys) while the list box has focus.

Do not check this box unless you want to hand code the LIST’s scrolling behavior.

8. Optionally, check the **Horizontal** or **Vertical** boxes to add scroll bars to your list box.

Check the scroll bar components you wish. The scroll bars manipulate the *entire* list. You can add horizontal scroll bars for individual columns with the **List Box Formatter** (see *Window Formatter—List Box Formatter*). Vertical scroll bars only appear when the number of items exceeds the available space in the list box.

9. In the **Drag ID** field, optionally type up to sixteen (16) comma delimited signatures.

   The **Window Formatter** adds the DRAGID attribute to your control. The DRAGID indicates this control is a valid source for drag and drop operations. The signature is a string constant that identifies which types of drag and drop operations are valid for the control.

   Drag and drop capability means the user can select an item in one window or control, hold down the left mouse button, drag the item to another window or control, and release the mouse button, dropping the item onto the other window or control, which can then look at the item that was dropped and do something with it.

   Implementation of this capability requires that the source control have a DRAGID attribute with a signature that matches the target's DROPID signature, and that the procedures that drive each window have appropriate source code to process the drag and drop events. See the *Language Reference* for more details and examples.

10. In the **Drop ID** field, optionally type up to sixteen (16) comma delimited signatures.

    The **Window Formatter** adds the DROPID attribute to your control. The DROPID indicates this control is a valid target for drag and drop operations. The signature is a string constant that identifies which types of drag and drop operations are valid for the control. See the *Language Reference* for more details and examples.

Help Tab

See *Common Control Attributes—Setting Help Attributes*.

Position Tab

See *Common Control Attribute*s*—Setting the AT Attribute.*

Actions Tab

The **Actions** tab is blank unless you are using a list control template. See the *Control Templates* in the *Application Handbook* for information on the prompts for each control template.

1. Optionally press the **Files** button to access the **File Schematic Definition** dialog for this procedure.

2. Optionally press the **Embeds** button to embed source code at points surrounding the event handling for this LIST only.

3. Press the **OK** button to return to the **Window Formatter**.

Combo Box Properties

The COMBO control combines an entry box with a list box. It is useful for selecting data which *usually* is a member of the list, but sometimes is not.

Combo Box properties are set exactly like List Box properties except for the following four additional properties associated with the entry box portion of the COMBO control.

General Tab

1. In the **Picture** field, specify the picture token for the control.

   Pressing the ellipsis button lets you select the picture token from the **Edit Picture String** dialog. See *Common Control Properties—The Picture Editor*.

   The picture token forces the input data into a specific format. For example, a picture token of @P##/##/##P forces a typical date format.

   You may check the user entry against the picture at two points: as the user types the data in, or when the user moves the focus to another control (for example, by TABBING to another field).

   Checking the data as the user types it incurs a slight performance penalty. To do so, check the **Entry Patterns** box in the **Window Properties** dialog. This adds the MASK attribute to the WINDOW. As the user types in data, the program attempts to convert it to match the picture. If the program cannot convert the data to match the picture, it sounds an audible alarm and returns focus to the control so the user can try again.

   If the MASK attribute is off, entry checking takes place when the user moves the focus to another control.

Extra Tab

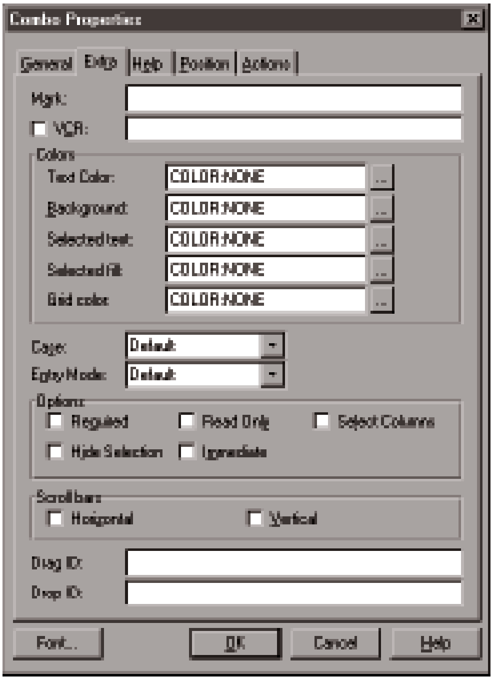
2. Specify **Case** attributes for the entry field.

   The entry box can automatically convert character from one case to another as the user types. **Uppercase** (UPR attribute) automatically converts to all caps. **Capitalize** (CAP attribute) is equivalent to “Proper Name” (the first letter of each word is uppercase). **Default** (no attribute) accepts input in the case the user types it.

3. In the **Entry Mode** drop-down list, choose **Default, Insert**, or **Overwrite**.

Sets the entry mode for the entry field of the combo box. **Insert** causes each keystroke to insert a new character and push existing characters to the right. **Overwrite** causes each keystroke to type a new character over an existing character. **Default** causes each keystroke to behave according to current system settings.

The **Entry Mode** applies only for windows with the MASK attribute. See the *Language Reference* for more information.



Mask:
□ VCR:
Colors
Text Color: COLOR:NONE
Background: COLOR:NONE
Selected text: COLOR:NONE
Selected fill: COLOR:NONE
Grid color: COLOR:NONE
Case: Default
Entry Mode: Default
Options
□ Required □ Read Only □ Select Columns
□ Hide Selection □ Immediate
Scroll bars
□ Horizontal □ Vertical
Drag ID:
Drop ID:

4. Optionally, check the **Read Only** box, to prevent data entry in this control.

   Adds the READONLY attribute to the combo box (see the *Language Reference*).

Spin Box Properties

A SPIN control is a specialized entry box that only accepts values in a predefined range. The SPIN also provide 'increase' and 'decrease' buttons, which many people like because they can use the mouse to change the value. You can also type a value directly into the control.

General Tab

1. In the **Picture** field, specify the picture token for the control.

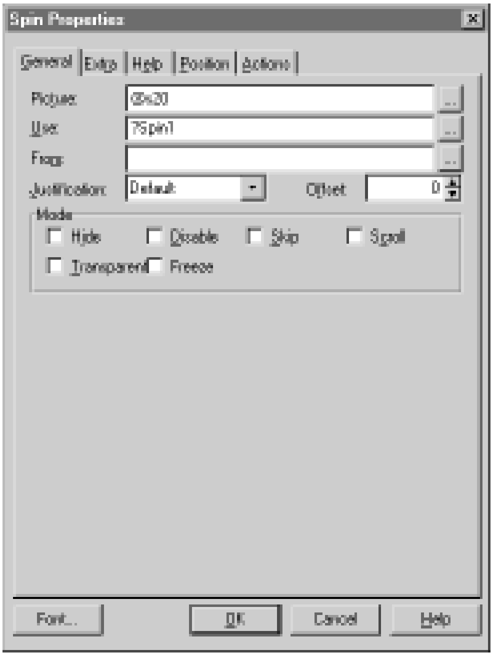
Pressing the ellipsis button lets you select the picture token from the **Edit Picture String** dialog. See *Common Control Properties—The Picture Editor*.

The picture token forces the input data into a specific format. For example, a picture token of @P##/##/##P forces a typical date format.

You may check the user entry against the picture at two points: as the user types the data in, or when the user moves the focus to another control (for example, by TABBING to another field).

Checking the data as the user types it incurs a slight performance penalty. To do so, check the **Entry Patterns** box in the **Window Properties** dialog. This adds the MASK attribute to the WINDOW. As the user types in data, the program attempts to convert it to match the picture. If the program cannot convert the data to match the picture, it beeps, and returns focus to the control so the user can try again.

If the MASK attribute is off, entry checking takes place when the user moves the focus to another control.



Picture: @s20

Use: ?Spin1

From:

Justification: Default

Offset: 0

□ Hide □ Disable □ Skip □ Scroll

□ Transparent □ Freeze

2. In the **Use** field, type a variable label.

   The variable receives the value the user selects. The same label, prefixed by a question mark (?) is the field equate label that references the spin box in source code statements. See *Common Control Properties—Setting the USE Attribute*.

3. In the **From** field, supply the data source for the spin box.

This sets the FROM attribute for the SPIN. See the *Language Reference* for more details. This is the label of a QUEUE structure, a field within a QUEUE, or a string constant.

The FROM attribute is useful for values that progress in an irregular increment. You may also wish to provide the user with string constants formatted as Spin Box choices when the choices are a limited progression such as the days of the week or the months of the year.

The **From** field and **Range** fields are mutually exclusive.

4. From the **Justification** drop-down list, choose **Left Justified**, **Centered**, **Right Justified**, **Decimal**, or **Default**.

   Adds the LEFT, CENTER, RIGHT, or DECIMAL attribute to the SPIN. See the *Language Reference* for details. **Left Justified**, **Centered**, and **Right Justified** position the data predictably, left, center, or right justified in the spin box. **Default** positions the data according to any applicable settings in the data dictionary. **Decimal** justification aligns values by their decimal points. Each justification may be offset by a distance you specify.

5. In the **Offset** field, specify a justification offset in dialog units.

   See the Glossary for the definition of dialog units. Sets the offset value for the LEFT, RIGHT, CENTER, and DECIMAL attributes. For CENTER justification, a negative value offsets to the left of center and a positive value offsets to the right of center.

   For DECIMAL justification, a negative value offsets to the left of the decimal and a positive value offsets to the right of the decimal.

Tip: For Decimal justification, use an offset equal to 4 * (decimal places + 1)

6. Check any combination of the **Mode** boxes.

   See *Common Control Properties—Setting Control Modes*.

Extra Tab

1. Optionally, select colors for the Combo box.

   See *Common Control Attributes—Setting the COLOR Attribute* above.

2. Specify the upper and lower **Range**, and the **Step** value.

   Place the highest value which the control should contain in the **Range Upper** field. The value should be formatted to match the **Picture** field. Place the lowest acceptable value in the **Lower** field. Place the step value—the amount by which each press of the increase or decrease buttons should change the spin box value—in the **Step** field.

   The **From** field and **Range** limits fields are mutually exclusive.

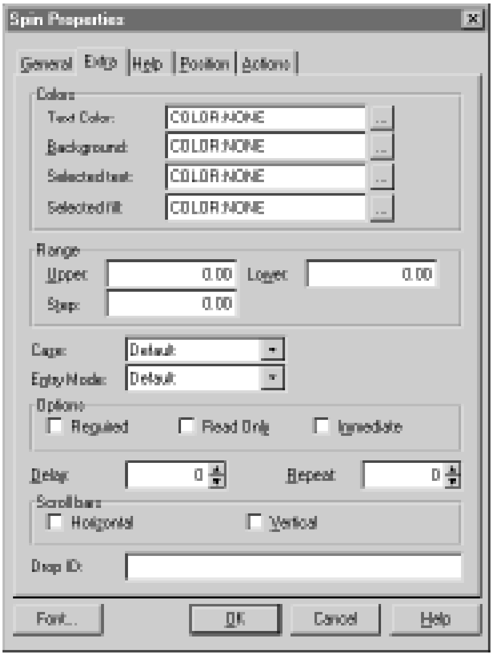
3. Specify **Case** attributes for the spin box.

The entry box can automatically convert character from one case to another as the user types. **Uppercase** (UPR attribute) automatically converts to all caps. **Capitalize** (CAP attribute) is equivalent to “Proper Name” (the first letter of each word will appear in caps). **Default** (no attribute) accepts input in the case the user types it.

4. In the **Entry Mode** drop-down list, choose **Default, Insert**, or **Overwrite**.

   Sets the entry mode for the entry field of the spin box. **Insert** causes each keystroke to insert a new character and push existing characters to the right. **Overwrite** causes each keystroke to type a new character over an existing character. **Default** causes each keystroke to behave according to current system settings.

   The **Entry Mode** applies only for windows with the MASK attribute set. See the *Language Reference* for more information.



Spin Properties

Colors
Text Color: COLOR:NONE
Background: COLOR:NONE
Selected text: COLOR:NONE
Selected fill: COLOR:NONE

Range
Upper: 0.00 Lower: 0.00
Step: 0.00

Case: Default
Entry Mode: Default

Options
□ Required □ Read Only □ Immediate

Delay: 0 Repeat: 0

Scroll bars
□ Horizontal □ Vertical

Drop ID:

5. Check the appropriate **Options** boxes.

   There are three option flags you may toggle on or off independently.

Required
(the REQ attribute) specifies that the control may not be left blank or zero.

Read Only
(the READONLY attribute) prevents data entry in this control. Use this to declare display-only data.

Immediate
(the IMM attribute) specifies immediate event generation whenever the user presses a key.

6. Set the delay between first and second EVENT:NewSelection event in the **Delay** box.

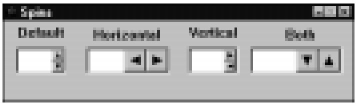
   This is how long the second EVENT:NewSelection occurs in hundredths of seconds, after the first EVENT:NewSelection when the end user clicks and holds the spin box arrow button.

7. Set the event generation rate for EVENT:NewSelection in the **Repeat** box (the REPEAT attribute).

   This is how often the EVENT:NewSelection occurs in hundredths of seconds when the end user clicks and holds the spin box arrow button.

8. Set the appearance of the Spin control buttons by checking any combination of the two **Scroll Bar** boxes.

   Checking neither box or the **Vertical** box produces smaller, vertically stacked buttons. Checking both boxes or the **Horizontal** box produces larger buttons arranged side by side.



9. In the **Drop ID** field, optionally type up to sixteen (16) comma delimited signatures.

   The **Window Formatter** adds the DROPID attribute to your control. The DROPID indicates this control is a valid target for drag and drop operations. The signature is a string constant that identifies which types of drag and drop operations are valid for the control.

   Drag and drop capability means the user can select an item in one window or control, hold down the left mouse button, drag the item to another window or control, and release the mouse button, dropping the item onto the other window or control, which can then look at the item that was dropped and do something with it.

   Implementation of this capability requires that the source control have a DRAGID attribute with a signature that matches the target's DROPID signature, and that the procedures that drive each window have appropriate source code to process the drag and drop events. See the *Language Reference* for more details and examples.

Help Tab

See *Common Control Attributes—Setting Help Attributes*.

Position Tab

See *Common Control Attributes—Setting the AT Attribute.*

Entry Box Properties

An ENTRY control allows the user to enter data from the keyboard. Clarion provides extensive options for automatically validating user entry.

- You may specify a picture for the field, automatically formatting the data the user enters.
- You may specify an initial value for the field.
- You may validate the data the user enters either at the time it's typed, or when the focus changes to another control.

To set the properties for an entry box RIGHT-CLICK the entry box then select **Properties** from the popup menu.

General Tab

1. In the **Picture** field, specify the picture token for the control.

   Pressing the ellipsis button lets you select the picture token from the **Edit Picture String** dialog. See *Common Control Attributes—The Picture Editor*.

   The picture token forces the input data into a specific format. For example, a picture token of @P##/##/##P forces a typical date format.

   You may check the user entry against the picture at two points: as the user types the data in, or when the user moves the focus to another control (for example, by TABBING to another field).

   Checking the data as the user types it incurs a slight performance penalty. To do so, check the **Entry Patterns** box in the **Window Properties** dialog. This adds the MASK attribute to the WINDOW. As the user types in data, the program attempts to convert it to match the picture. If the program cannot convert the data to match the picture, it beeps, and returns focus to the control so the user can try again.

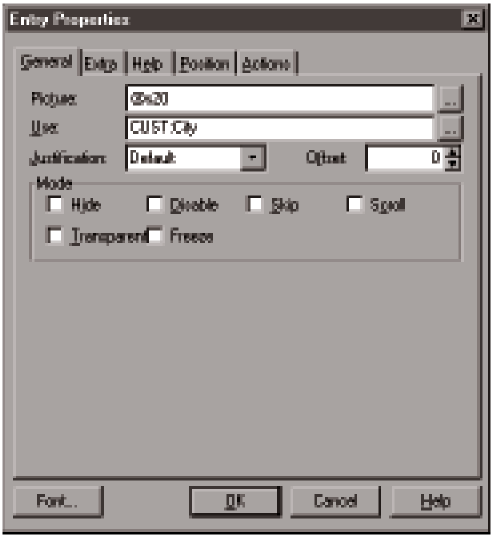
   If the MASK attribute is off, entry checking takes place when the user moves the focus to another control.

2 Specify a **Use** attribute.

   Type the name of the variable to receive the value the user enters in the control; or press the ellipsis (...) button to select or define a variable. See *Common Control Attributes—Setting the USE Attribute*.

3. From the **Justification** drop-down list, choose **Left Justified**, **Centered**, **Right Justified**, **Decimal**, or **Default**.

Adds the LEFT, CENTER, RIGHT, or DECIMAL attribute to the ENTRY. See the *Language Reference* for details. **Left Justified**, **Centered**, and **Right Justified** position the data predictably, left, center, or right justified in the spin box. **Default** positions the data according to any applicable settings in the data dictionary. **Decimal** justification aligns values by their decimal points. Each justification may be offset by a distance you specify.



Picture: @s20

Use: CUST:City

Justification: Default

Offset: 0

Mode

□ Hide □ Disable □ Skip □ Scroll

□ Transparent □ Freeze

4. In the **Offset** field, specify a justification offset in dialog units.

   See the Glossary for the definition of dialog units. Sets the offset value for the LEFT, RIGHT, CENTER, and DECIMAL attributes. For CENTER justification, a negative value offsets to the left of center and a positive value offsets to the right of center.

   For DECIMAL justification, a negative value offsets to the left of the decimal and a positive value offsets to the right of the decimal.

Tip: For Decimal justification, use an offset equal to 4 * (decimal places + 1)

5. Set **Mode** options.

   See *Common Control Attributes—Setting Control Modes.*

Extra Tab

1. Optionally, select colors for the entry box.

   See *Common Control Attributes—Setting the COLOR Attribute.*

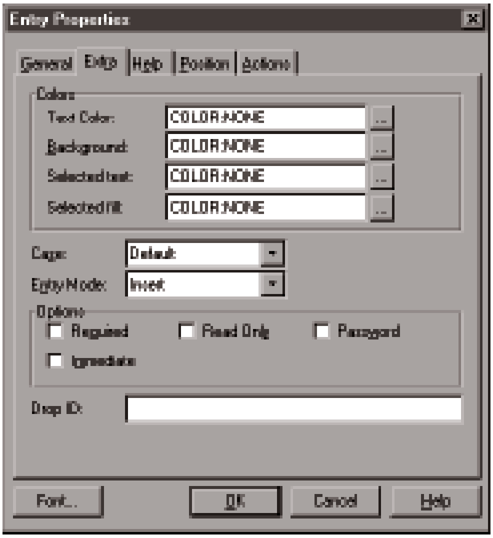
2. Specify **Case** attributes for the entry field.

The entry box can automatically convert character from one case to another as the user types. **Uppercase** (UPR attribute) automatically converts to all caps. **Capitalize** (CAP attribute) is equivalent to “Proper Name” (the first letter of each word will appear in caps). **Default** (no attribute) accepts input in the case the user types it.

3. In the **Entry Mode** drop-down list, choose **Default, Insert**, or **Overwrite**.

   Sets the entry mode for the entry field of the entry box. **Insert** causes each keystroke to insert a new character and push existing characters to the right. **Overwrite** causes each keystroke to type a new character over an existing character. **Default** causes each keystroke to behave according to current system settings.

   The **Entry Mode** applies only for windows with the MASK attribute set. See the *Language Reference* for more information.



Text Color: COLOR:NONE

Background: COLOR:NONE

Selected text: COLOR:NONE

Selected fill: COLOR:NONE

Case: Default

Entry Mode: Insert

□ Required □ Read Only □ Password

□ Immediate

Drop ID:

4. Set the Option flags.

   You may toggle the following options on or off independently:

Required
(the REQ attribute) specifies that the control may not be left blank or zero.

Read Only
(the READONLY attribute) prevents data entry in this control. Use this to declare display-only data.

Password
(the PASSWORD attribute) specifies non-display of data entered in the control, that is, all characters typed are displayed as asterisks, plus standard Copy and Cut are disabled for the control so users cannot copy a password and paste it into a text editor or another control.

Immediate

(the IMM attribute) specifies immediate event generation whenever the user presses any key.

5. In the **Drop ID** field, optionally type up to sixteen (16) comma delimited signatures.

   The **Window Formatter** adds the DROPID attribute to your control. The DROPID indicates this control is a valid target for drag and drop operations. The signature is a string constant that identifies which types of drag and drop operations are valid for the control.

   Drag and drop capability means the user can select an item in one window or control, hold down the left mouse button, drag the item to another window or control, and release the mouse button, dropping the item onto the other window or control, which can then look at the item that was dropped and do something with it.

   Implementation of this capability requires that the source control have a DRAGID attribute with a signature that matches the target's DROPID signature, and that the procedures that drive each window have appropriate source code to process the drag and drop events. See the *Language Reference* for more details and examples.

Help Tab

See *Common Control Attributes—Setting Help Attributes*.

Tip: The I-Beam, which signals text entry, is the standard choice for the active cursor for a text or entry control.

Position Tab

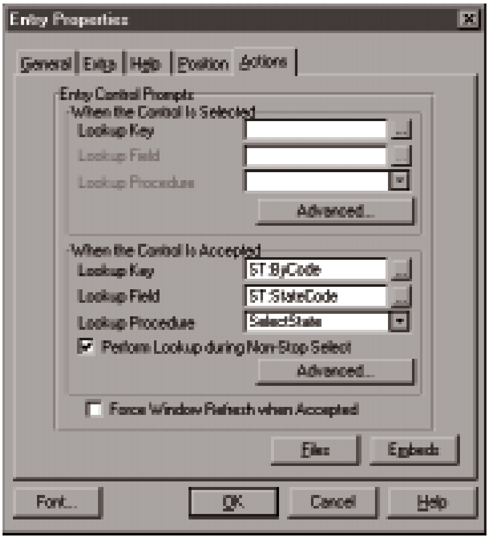
See *Common Control Attributes—Setting the AT Attribute*.

Actions Tab

The **Actions** tab prompts are all from the templates, in other words, the prompts you see here vary with the template used to create the control. Following are the standard action prompts for all entry controls. See the *Control, Code, and Extension Templates* chapter for more information.

The standard **Actions** prompts are designed to provide data validation support for your entry controls. The tab is divided into two parallel sections. The **When the Control is Selected** section provides validation when the control *receives* focus (when the user TABS onto, or mouse CLICKS the control). The **When the Control is Accepted** section provides data validation when the control *loses* focus after data have been entered in it. The control loses focus when the user TABS off the control, mouse CLICKS to a different control or window, or closes the window without cancelling. The two sections are not mutually exclusive, so you can provide validation at both points.

The standard **Actions** prompts are designed with selection list lookup validation in mind, however, they are flexible enough to allow any custom validation you might want to provide.



Lookup Key
Lookup Field
Lookup Procedure

When the Control Is Accepted

Lookup Key: ST:ByCode
Lookup Field: ST:StateCode
Lookup Procedure: SelectState
☑ Perform Lookup during Non-Stop Select

☐ Force Window Refresh when Accepted

1. In the **Lookup Key** field, type a key label from the lookup file, or press the ellipsis (...) button to select a key from the **Select Key** dialog.

   A lookup file is a file which contains all the valid values for the entry field, and they are directly accessible through a unique key, which is the lookup key you name here.

   For example, a file containing all of the customer numbers for your application could be a lookup file. The key label could be ST:ByCode.

   The **Select Key** dialog lets you select from files and keys already defined in the Data Dictionary, or to define a new key if necessary.

Note: Defining a new key changes the file format and may therefore require you to convert any *existing* files to the new format.

Tip: This lookup validation works best with a single component unique key.

2. In the **Lookup Field** field, type the label of a component field of the lookup key, or press the ellipsis (...) button to select a field from the **Select component from key** dialog.

   This is the field within the key that contains the same value being validated. Ideally, this field is the only component of a unique key. Following our example above, the field label could be ST:StateCode.

3. In the **Lookup Procedure** combo box, type a procedure name, or choose an existing procedure from the drop-down list.

This is the procedure that is called when the user enters an invalid value, and the lookup *fails*. The usual purpose of this procedure is to allow the user to choose a valid value from the lookup file.

Select procedures (Browse procedures) generated by Clarion's Wizards are appropriate for this purpose. Alternatively, you may hand-code a procedure. Continuing our example above, the procedure name could be SelectState.

4. Optionally, press the **Advanced** button to customize the (Selected or Accepted) event handling source code for this entry control.

   Pressing the **Advanced** button calls the **Embedded Source** dialog. The only embed point shown is after the code generated to call the lookup procedure specified above. For more embed points press the **Embeds** button. Also see *Application Generator—Embedded Source Code*.

5. Optionally, check the **Perform lookup during non-stop select** box.

   Checking this box tells Clarion to perform the validation when the *window* is accepted, even if the *entry control* never received focus. From a practical viewpoint, checking this box prevents the user from entering blanks by virtue of having pressed the window's "OK button" without ever TABBING or CLICKING onto the entry field.

   This option is only applicable to the **When the Control is Accepted** section.

6. Optionally, check the **Force Window Refresh when Accepted** box.

   Checking this box ensures that everything (including formulas and other entry fields) on the window is current and up-to-date when the user TABS off this entry control.

7. Optionally press the **Files** button to access the **File Schematic Definition** dialog for this procedure.

8. Optionally press the **Embeds** button to embed source code at points surrounding the event handling for this check box only.

9. Press the **OK** button to return to the **Window Formatter**.

Text Properties

The TEXT control provides a multi-line data entry field. This control is especially suitable for holding a long STRING or a MEMO. The TEXT control provides automatic word wrapping.

To set the properties for a text box RIGHT-CLICK the text box then select **Properties** from the popup menu.

General Tab

1. Specify a **Use** attribute.

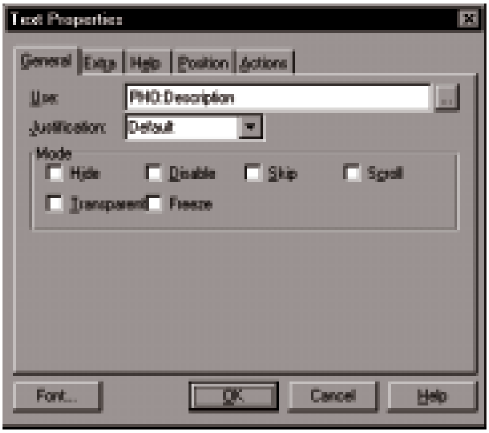
   Type the name of the variable to receive the value that the user enters in the control; or press the ellipsis button (...) to select or define the variable. Be sure the variable is large enough to hold the amount of data you expect your users to enter in the control. See *Common Control Properties—Setting the USE Attribute*.

Tip: **You can control line breaks within TEXT control text by specifying the ASCII code for a carriage return/line feed within angle brackets at the appropriate place within the text. In fact, you can include any special characters with this technique. For example:**

```
MyText = ‘Here comes a line break. <13,10> Yada yada...’
```

2. From the **Justification** drop-down list, choose **Left Justified, Right Justified, Centered** or **Default**.

   Adds the LEFT, CENTER, or RIGHT attribute to the TEXT. See the *Language Reference* for details. **Left Justified, Right Justified** and **Centered** position the data predictably, left, center, or right justified in the text box. **Default** positions the data according to any applicable settings in the data dictionary.



Use: PHO:Description

Justification: Default

Mode: □ Hide □ Disable □ Skip □ Scroll □ Transparent □ Freeze

3. Set **Mode** options.

   See *Common Control Properties—Setting Control Modes*.

Extra Tab

1. Optionally, select colors for the text box.

   See *Common Control Attributes—Setting the COLOR Attribute*.

2. Specify case attributes for the text field.

The entry box can automatically convert character from one case to another as the user types. **Uppercase** (UPR attribute) automatically converts to all caps. **Default** (no attribute) accepts input in the case the user types it.

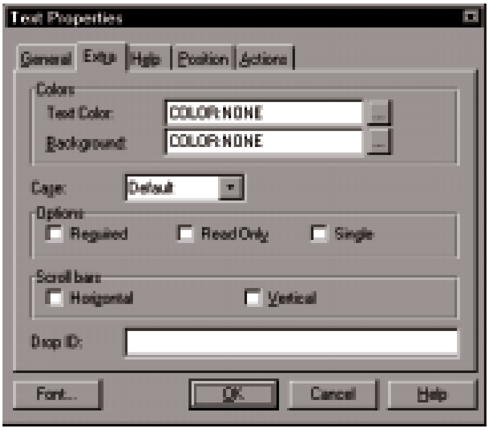
3. Set the Option flags.

   There are several option flags you may toggle on or off independently.

   Required
   (the REQ attribute) specifies that the control may not be blank or zero.

   Read Only
   (the READONLY attribute) prevents data entry in this control. Use this to declare display-only data.

   Single
   (the SINGLE attribute) specifies the control is only for single line data entry. This lets you use TEXT controls instead of ENTRY controls for languages that write from right to left (such as Hebrew or Arabic).



Text Color: COLOR:NONE

Background: COLOR:NONE

Case: Default

4. Optionally, check the **Horizontal** or **Vertical** boxes to add scroll bars to your text box.

5. In the **Drop ID** field, optionally type up to sixteen (16) comma delimited signatures.

   The **Window Formatter** adds the DROPID attribute to your control. The DROPID indicates this control is a valid target for drag and drop operations. The signature is a string constant that identifies which types of drag and drop operations are valid for the control.

   Drag and drop capability means the user can select an item in one window or control, hold down the left mouse button, drag the item to another window or control, and release the mouse button, dropping the item onto the other window or control, which can then look at the item that was dropped and do something with it.

Implementation of this capability requires that the source control have a DRAGID attribute with a signature that matches the target's DROPID signature, and that the procedures that drive each window have appropriate source code to process the drag and drop events. See the *Language Reference* for more details and examples.

Help Tab

See *Common Control Attributes—Setting Help Attributes*.

> **Tip: The I-Beam, which signals text entry, is the standard choice for the active cursor for a text control.**

Position Tab

See *Common Control Attribute*s*—Setting the AT Attribute.*

Sheet Properties

The SHEET control declares a *group* of TAB controls that offer the user *multiple pages* of controls for a single window. The multiple TAB controls in the SHEET structure define the pages displayed to the user. The SHEET structure's USE variable receives the text of the TAB control selected by the user.

General Tab

1. Place a field equate label in the **Use** field.

   The field equate label references the property sheet in program statements. See *Common Control Attribute*s*—Setting the USE Attribute.*

2. From the **Justification** drop-down list, set the location of the tabs for this SHEET. Choose from:

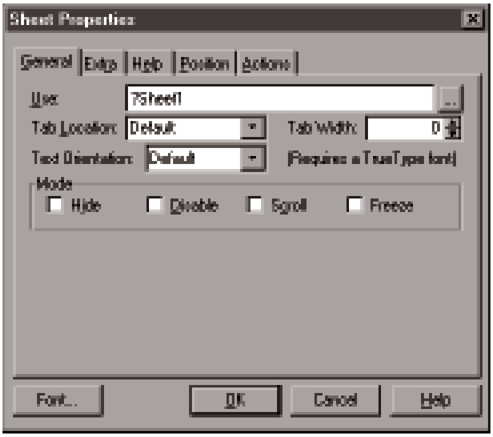
Default	The tabs run across the top of the sheet.
Left	The tabs run down the left side of the sheet. This adds the LEFT attribute to the SHEET.
Right	The tabs run down the right side of the sheet. This adds the RIGHT attribute to the SHEET.
Above	The tabs run across the top of the sheet. This adds the ABOVE attribute to the SHEET.
Below	The tabs run across the bottom of the sheet. Adds the BELOW attribute to the SHEET.

Tip: To fit many tabs on a sheet, set the Justification to Above and set the Text Orientation to Up. Alternatively set the Justification to Right and set the Text Orientation to Default. Also, see Extra Tab to make scrolling tabs!

3. In the **Tab Width** field, specify the tab width in dialog units.

   This sets the value of the width parameter for the LEFT, RIGHT, ABOVE, or BELOW attribute. By setting this width, you can make all your tabs the same size, regardless of varying text lengths per tab.

   Tab width is the distance between the edges of the tab that are perpendicular to the text orientation. That is, width determines how much space appears on either end of your tab's text, not how much space appears above and below the text. This is true, regardless of text orientation.



Sheet Properties

Use: ?Sheet1

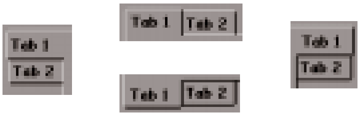
Tab Location: Default

Tab Width: 0

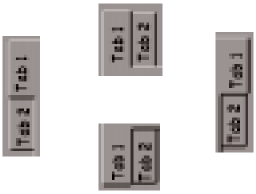
Text Orientation: Default

4. Use the **Text Orientation** drop-down list to set the orientation of the tabs and their text. A **Text Orientation** other than *Default* requires a TrueType font. See *Common Control Properties—Setting the FONT Attribute* for more information. Choose from the following orientations:

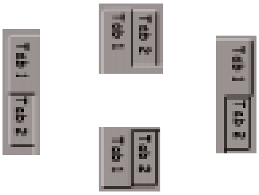
Default The text reads left to right and the shape of the tab is a horizontal rectangle.



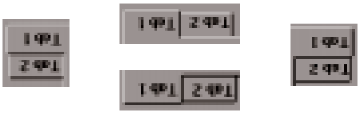
Up The text reads from bottom to top and the shape of the tab is a vertical rectangle. This adds the UP attribute to the SHEET.



Down The text reads from top to bottom and the shape of the tab is a vertical rectangle. This adds the DOWN attribute to the SHEET.



Inverted The text is upside down and the shape of the tab is a horizontal rectangle. This adds the UP and the DOWN attribute to the SHEET.



5. Set **Mode** options.

   See *Common Control Properties—Setting Control Modes*.

Extra Tab

1. Optionally, set the colors for the SHEET.

   The colors apply to all TABs on the SHEET, but can be overridden by setting colors for the individual TABs. See *Common Control Attributes—Setting the COLOR Attribute.*

2. Check the **Spread** box to resize the tabs on the TABs to fill all the available space on the SHEET.

   The resizing algorithm considers the size and orientation of the text displayed on each tab, the number of tabs, and the available space on the property sheet. This adds the SPREAD attribute to the SHEET. See the *Language Reference* for more information.

3. Check the **Wizard** box to hide the “tab” portion of the TAB controls.

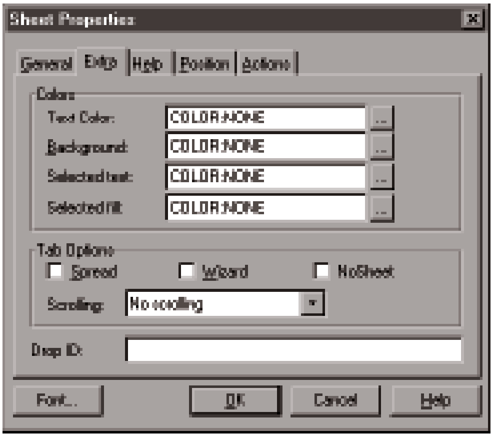
   Hiding the tabs aids in creating a wizard. A wizard is a window with a “tabless” SHEET control containing one or more TABs. You’ll need to write the code to handle the “turning of the pages”.

This adds the WIZARD attribute to the SHEET. See the *Language Reference* for more information. See *How to Create a Wizard* in the on-line help.

Tip: Do not check this box until you are finished designing the window!

4. Check the **NoSheet** box to erase the borders of the tab pages so that only the protruding selectable portion of the tab is visible.

   This has the additional effect of making tabs located above the sheet, fall to the bottom of the sheet, and making tabs located below the sheet rise to the top of the sheet. This adds the NOSHEET attribute to the SHEET. See the *Language Reference* for more information.



Sheet Properties

General | Extra | Help | Position | Actions

Colors

Text Color: COLOR:NONE ...

Background: COLOR:NONE ...

Selected text: COLOR:NONE ...

Selected fill: COLOR:NONE ...

Tab Options

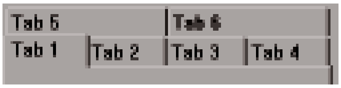
□ Spread □ Wizard □ NoSheet

Scrolling: No scrolling ▾

Drop ID:

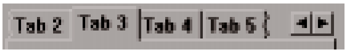
5. From the **Scrolling** drop-down list, specify tabs scrolling behavior by choosing one of the following selections:

No Scrolling The tabs do not scroll. This is the default. If necessary, the tabs are arranged in multiple rows.



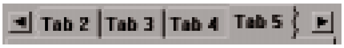
Joined Scroll Buttons

Tabs are scrollable, with adjacent scroll buttons. Adds the JOINED attribute to the SHEET. See the *Language Reference* for more information.



Spread Scroll Buttons

Tabs are scrollable, with scroll buttons at opposite ends of the sheet. Adds the HSCROLL attribute to the SHEET. See the *Language Reference* for more information.



6. In the **Drop ID** field, optionally type up to sixteen (16) comma delimited signatures.

   The **Window Formatter** adds the DROPID attribute to your control. The DROPID indicates this control is a valid target for drag and drop operations. The signature is a string constant that identifies which types of drag and drop operations are valid for the control.

   Drag and drop capability means the user can select an item in one window or control, hold down the left mouse button, drag the item to another window or control, and release the mouse button, dropping the item onto the other window or control, which can then look at the item that was dropped and do something with it.

   Implementation of this capability requires that the source control have a DRAGID attribute with a signature that matches the target's DROPID signature, and that the procedures that drive each window have appropriate source code to process the drag and drop events. See the *Language Reference* for more details and examples.

Help Tab

See *Common Control Attributes—Setting Help Attributes*.

Position Tab

See *Common Control Attribute*s*—Setting the AT Attribute*.

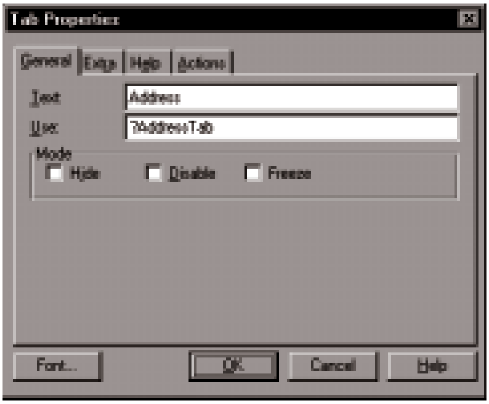
Tab Properties

The TAB structure declares a group of controls. This group is one of multiple pages of controls that may be contained within a SHEET structure. The multiple TAB structures within the SHEET structure define the pages displayed to the user. The SHEET structure's USE attribute receives the text of the TAB control selected by the user.

The Windows 95 standard to change from tab to tab is CTRL+TAB. Clarion TAB controls follow this standard, both in the development environment and in a compiled application.

Note: If you nest TABS, only the top level is controlled by CTRL+TAB.

General Tab



Text: Address

Use: ?AddressTab

Mode

□ Hide □ Disable □ Freeze

1. In the **Text** field, type a string constant.

   See *Common Control Properties—Setting the TEXT Attribute*.If the control is to display a variable, type a picture token in this field. See *Common Control Properties—The Picture Editor*.

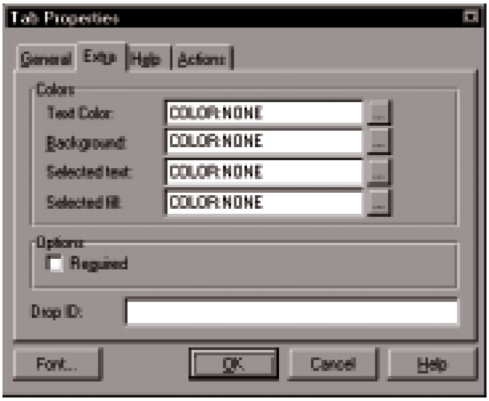
2. In the **Use** field, type a field equate label.

   The field equate label references the tab control in program statements. See *Common Control Properties—Setting the USE Attribute*.

3. Set **Mode** options.

   See *Common Control Properties—Setting Control Modes*.

Extra Tab



Colors

Text Color: COLOR:NONE

Background: COLOR:NONE

Selected text: COLOR:NONE

Selected fill: COLOR:NONE

Options

□ Required

Drop ID:

1. Optionally, set the colors for the TAB.

   The colors set here override colors for the SHEET. See *Common Control Attributes—Setting the COLOR Attribute.*

2. Check the **Required** box to enforce input to required input fields.

When checked, your program automatically checks that all input controls with the REQ attribute are neither blank nor zero.

Specify this type of tab when a window also contains an ENTRY, COMBO, SPIN or TEXT control field with the REQ attribute (or else use the INCOMPLETE function to test the input controls). When the user clicks on a tab with the REQ attribute and an input field is blank or zero, the first required control which is blank or zero receives focus.

3. In the **Drop ID** field, optionally type up to sixteen (16) comma delimited signatures.

   The **Window Formatter** adds the DROPID attribute to your control. The DROPID indicates this control is a valid target for drag and drop operations. The signature is a string constant that identifies which types of drag and drop operations are valid for the control.

   Drag and drop capability means the user can select an item in one window or control, hold down the left mouse button, drag the item to another window or control, and release the mouse button, dropping the item onto the other window or control, which can then look at the item that was dropped and do something with it.

   Implementation of this capability requires that the source control have a DRAGID attribute with a signature that matches the target's DROPID signature, and that the procedures that drive each window have appropriate source code to process the drag and drop events. See the *Language Reference* for more details and examples.

Help Tab

See *Common Control Attributes—Setting Help Attributes*.

Position

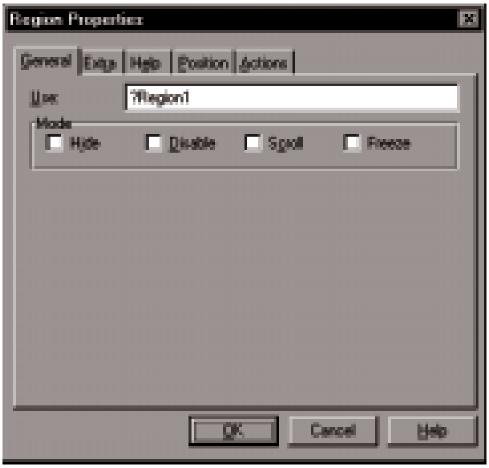
The position of the TAB is determined by the position of the parent SHEET.

Region Properties

A REGION control is simply a rectangular area of the screen. Its main purpose is to provide a reference to test whether a given event—such as a mouse event—occurred within that region.

You may give a region control color, a bevel, or provide for a cursor change when the user passes the mouse over the region.

General Tab



Use: ?Region1

Mode

□ Hide □ Disable □ Scroll □ Freeze

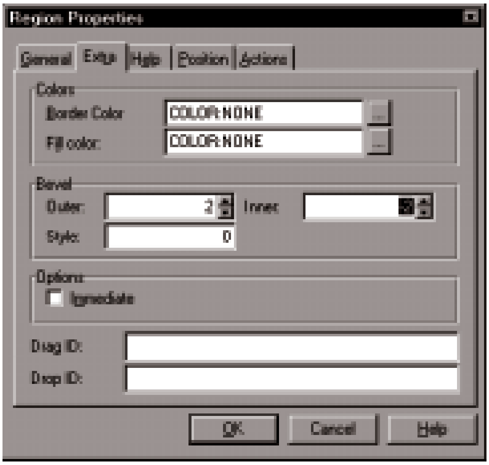
1. Place a field equate label in the **Use** field.

   The field equate label references the region in program statements. See *Common Control Properties—Setting the USE Attribute*.

2. Set **Mode** options.

   See *Common Control Properties—Setting Control Modes*.

Extra Tab



Colors

Border Color: COLOR:NONE

Fill color: COLOR:NONE

Bevel

Outer: 2 Inner: 0

Style: 0

Options

□ Immediate

Drag ID:

Drop ID:

1. Optionally, specify the **Fill** and **Border** colors.

   Type a valid color equate or press the ellipsis button to select a color. The standard **Color** dialog appears. Select a color by clicking on the color selection square, or add a custom color. See *Common Control Attributes—The Color Dialog*.

2. Optionally, set the BEVEL attribute for the REGION.

The BEVEL attribute gives a three dimensional look to the region. The region appears raised, depressed, or both.

Outer

A positive value makes the region appear raised above the plane of the window. The higher the value, the further the region is raised. A negative value makes the region appear depressed below the plane of the window. The bevel effect begins at the outer border of the region.

Inner

A positive value makes the region appear raised above the plane of the window. The higher the value, the further the region is raised. A negative value makes the region appear depressed below the plane of the window. The bevel effect begins immediately inside the outer bevel.

Style

A USHORT, whose sixteen bits define the style (but not the size) of the each of the four edges of the REGION. The STYLE attribute gives you very fine control over the bevel appearance. See the *Language Reference* for an explanation of the meaning of each bit.

3. You may add the IMM attribute to the Region control by checking the **Immediate** box.

   This generates events (EVENT:MouseIn, EVENT:MouseOut, EVENT:MouseMove) whenever the user passes the cursor over the region; however, it incurs substantial overhead at run-time, so should be used sparingly.

4. In the **Drag ID** field, optionally type up to sixteen (16) comma delimited signatures.

   The **Window Formatter** adds the DRAGID attribute to your control. The DRAGID indicates this control is a valid source for drag and drop operations. The signature is a string constant that identifies which types of drag and drop operations are valid for the control.

   Drag and drop capability means the user can select an item in one window or control, hold down the left mouse button, drag the item to another window or control, and release the mouse button, dropping the item onto the other window or control, which can then look at the item that was dropped and do something with it.

   Implementation of this capability requires that the source control have a DRAGID attribute with a signature that matches the target’s DROPID signature, and that the procedures that drive each window have appropriate source code to process the drag and drop events. See the *Language Reference* for more details and examples.

5. In the **Drop ID** field, optionally type up to sixteen (16) comma delimited signatures.

The **Window Formatter** adds the DROPID attribute to your control. The DROPID indicates this control is a valid target for drag and drop operations. The signature is a string constant that identifies which types of drag and drop operations are valid for the control. See the *Language Reference* for more details and examples.

Help Tab

See *Common Control Attributes—Setting Help Attributes*.

Position Tab

See *Common Control Attribute*s*—Setting the AT Attribute.*

Non-Interactive Controls

Non interactive controls do not accept data, but instead guide the user to other controls with text or graphics. For example:

- A string in a dialog box can provide directions for filling out the data field.
- One of the simplest graphic elements—a group box—can visually associate a group of controls, signalling the user that the entries all relate to the same thing.
- An image or graphic can do more than embellish a dialog. It can convey meaning to a process that might otherwise take many, many words.

String Properties

The String control lets you place a string constant on screen. It optionally lets you display a variable value.

> **Tip: Strings do not support multi-line text; for multi-line text (word wrap) use a Prompt control or a Text control.**

The **String Properties** dialog contains the following options.

General Tab

1. In the **Text** field, type the string constant or a picture token.

   A string constant is displayed as typed. See *Common Control Attributes—Setting the Text Attribute*.

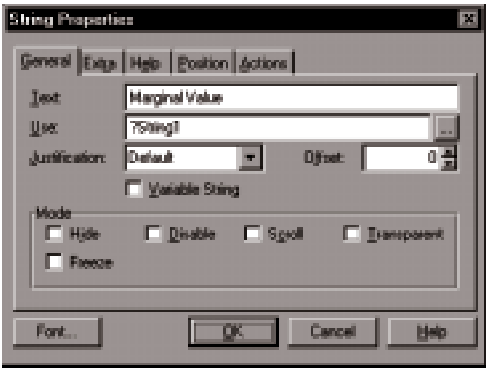
   Checking the **Variable String** box changes the **Text:** prompt to **Picture:**. A picture token is used to format a variable string for display. Press the ellipsis button (...) to define an appropriate picture token. See *Common Control Attributes—The Picture Editor.*

2. In the **Use** field, type a field equate label or the label of a variable to display.

   A field equate label references the control in source code. Press the ellipsis button (...) to select or define a variable to display. See *Common Control Attributes—Setting the USE Attribute.*

3. From the **Justification** drop-down list, choose **Left Justified**, **Centered**, **Right Justified**, **Decimal**, or **Default**.

Adds the LEFT, CENTER, RIGHT, or DECIMAL attribute to the STRING. See the *Language Reference* for details. **Left Justified**, **Centered**, and **Right Justified** position the data predictably, left, center, or right justified in the control. **Default** positions the data according to any applicable settings in the data dictionary. **Decimal** justification aligns values by their decimal points. Each justification may be offset by a distance you specify.



Text: Marginal Value
Use: ?String1
Justification: Default
Offset: 0
□ Variable String
□ Hide □ Disable □ Scroll □ Transparent
□ Freeze

4. In the **Offset** field, specify a justification offset in dialog units.

   See the Glossary for the definition of dialog units. Sets the offset value for the LEFT, RIGHT, CENTER, and DECIMAL attributes. For CENTER justification, a negative value offsets to the left of center and a positive value offsets to the right of center.

   For DECIMAL justification, a negative value offsets to the left of the decimal and a positive value offsets to the right of the decimal.

Tip: For Decimal justification, use an offset equal to 4 * (decimal places + 1)

5. Optionally check the **Variable String** box.

   This specifies that you want to display the contents of a variable in the string control. If so, place a picture in the **Picture** field, such as @s2. See *Common Control Attributes—The Picture Editor.*

6. Set **Mode** options.

   See *Common Control Attributes—Setting Control Modes* above.

Tip: When you place text on top of an IMAGE or a colored graphic such as a BOX, check the Transparent box (TRN attribute) so as not to obscure the graphic.

Extra Tab

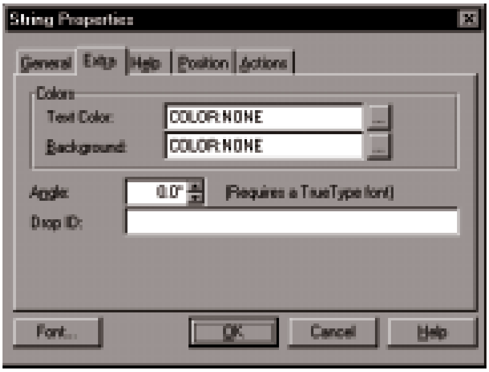
1. Optionally, select colors for the string.

   See *Common Control Attributes—Setting the COLOR Attribute* above.

2. Optionally, specify the **Angle** of the text.

   An angle other than zero requires a TrueType font. See *Common Control Properties—Setting the FONT Attribute* for more information. Adds the ANGLE attribute to the STRING. With this, you can rotate the text from its normal horizontal position through a full 360 degrees.

Tip: Make the control taller than usual to accommodate the slant of the text.



Text Color: COLOR:NONE
Background: COLOR:NONE
Angle: 0.0° (Requires a TrueType font)
Drop ID:

3. In the **Drop ID** field, optionally type up to sixteen (16) comma delimited signatures.

   The **Window Formatter** adds the DROPID attribute to your button. The DROPID indicates this button is a valid target for drag and drop operations. The signature is a string constant that identifies which types of drag and drop operations are valid for the button.

   Drag and drop capability means the user can select an item in one window or control, hold down the left mouse button, drag the item to another window or control, and release the mouse button, dropping the item onto the other window or control, which can then look at the item that was dropped and do something with it.

   Implementation of this capability requires that the source control have a DRAGID attribute with a signature that matches the target's DROPID signature, and that the procedures that drive each window have appropriate source code to process the drag and drop events. See the *Language Reference* for more details and examples.

Help Tab

See *Common Control Attributes—Setting Help Attributes*.

Position Tab

See *Common Control Attribute*s*—Setting the AT Attribute.*

Prompt Properties

The PROMPT control lets you place a string on screen which automatically provides an accelerator key to the next active control following the prompt. It is identical to the STRING control, except it has no variable capability and no angle capability; however, unlike a STRING, a PROMPT supports word wrapping (multi-line text). See *String Properties*.

Group Box Properties

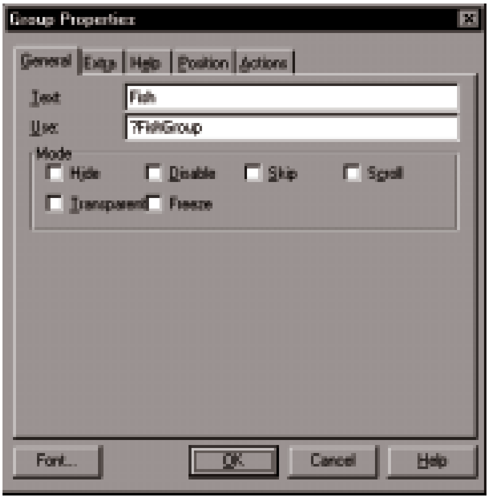
A GROUP control draws a box around other controls. It visually associates the controls for the user, and *lets you address all the controls as one entity*—making it easy, for example, to disable all at once.

The **Group Properties** dialog contains the following options.

General Tab

1. In the **Text** field, type a string constant.

   The **Text** field takes a string constant containing the prompt for the group of controls. An ampersand (&) within the text means the next character is the accelerator key for the group. See *Common Control Properties—Setting the Text Attribute*.



Group Properties

Text: Fish

Use: ?FishGroup

Mode

□ Hide □ Disable □ Skip □ Scroll

□ Transparent □ Freeze

2. In the **Use** field, type a field equate label.

   A field equate label references the control in source code. See *Common Control Properties—Setting the USE Attribute*.

3. Set **Mode** options.

   See *Common Control Properties—Setting Control Modes*.

Extra Tab

1. Optionally, set colors for the group box.

   See *Common Control Attributes—Setting the COLOR Attribute.*

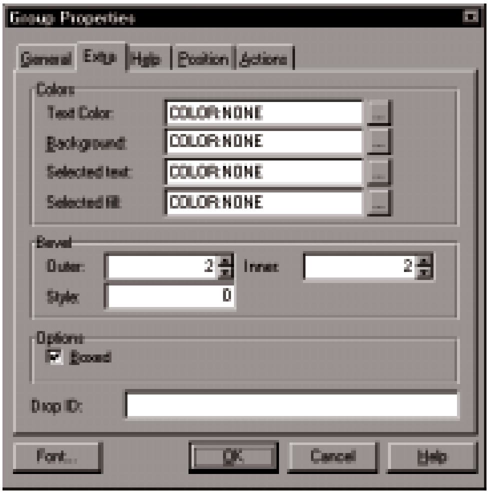
2. Optionally, set the BEVEL attribute for the GROUP.

   The BEVEL attribute gives a three dimensional look to the group box. The box appears raised, depressed, or both.

   Outer
   A positive value makes the group box appear raised above the plane of the window. The higher the value, the further the box is raised. A negative value makes the group box appear depressed below the plane of the window. The bevel effect begins at the outer border of the box.

   Inner
   A positive value makes the group box appear raised above the plane of the window. The higher the value, the further the box is raised. A negative value makes the group box appear depressed below the plane of the window. The bevel effect begins immediately inside the outer bevel.

   Style
   A USHORT, whose sixteen bits define the style (but not the size) of the each of the four edges of the GROUP. The STYLE attribute gives you very fine control over the bevel appearance. See the *Language Reference* for an explanation of the meaning of each bit.



Colors
Text Color: COLOR:NONE
Background: COLOR:NONE
Selected text: COLOR:NONE
Selected fill: COLOR:NONE

Bevel
Outer: 2
Inner: 2
Style: 0

Options
☑ Boxed

Drop ID:

3. Optionally, make the group box invisible.

By clearing the **Boxed** box, you may make the group box and its text invisible to the user. This removes the BOXED attribute from the GROUP. The group box is visible in the **Window Formatter**, but invisible in **Preview!** mode and at runtime. This creates a different effect than hiding or disabling the group. Hiding or disabling the group also hides or disables all controls within the group.

4. In the **Drop ID** field, optionally type up to sixteen (16) comma delimited signatures.

   The **Window Formatter** adds the DROPID attribute to your button. The DROPID indicates this button is a valid target for drag and drop operations. The signature is a string constant that identifies which types of drag and drop operations are valid for the button.

   Drag and drop capability means the user can select an item in one window or control, hold down the left mouse button, drag the item to another window or control, and release the mouse button, dropping the item onto the other window or control, which can then look at the item that was dropped and do something with it.

   Implementation of this capability requires that the source control have a DRAGID attribute with a signature that matches the target's DROPID signature, and that the procedures that drive each window have appropriate source code to process the drag and drop events. See the *Language Reference* for more details and examples.

Help Tab

See *Common Control Attributes—Setting Help Attributes*.

Position Tab

See *Common Control Attributes—Setting the AT Attribute.*

Progress Bar Properties

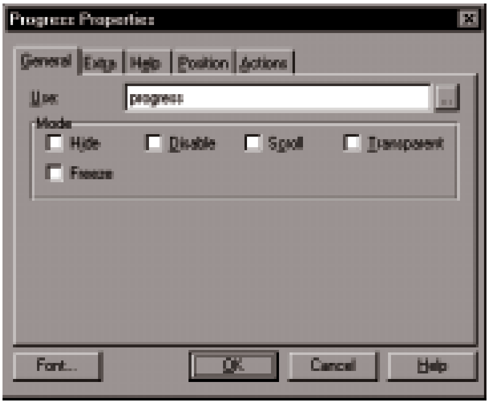
The PROGRESS control declares a control that displays a progress bar. This usually displays the current percentage of completion of a batch process by incrementally "filling" the bar as the process progresses.

The **Progress Properties** dialog contains the following options.

General Tab

1. In the **Use** field, type the label of the numeric variable that tracks the progress of your process.

The progress bar is automatically updated whenever the value in the variable changes. If the USE attribute is a field equate label, you can update the progress bar display by assigning a value (within the range defined by the RANGE attribute) to the control's PROP:Progress property. See *Common Control Properties—Setting the USE Attribute*.



Use: progress

Mode

□ Hide □ Disable □ Scroll □ Transparent

□ Freeze

2. Set **Mode** options.

   See *Common Control Properties—Setting Control Modes*.

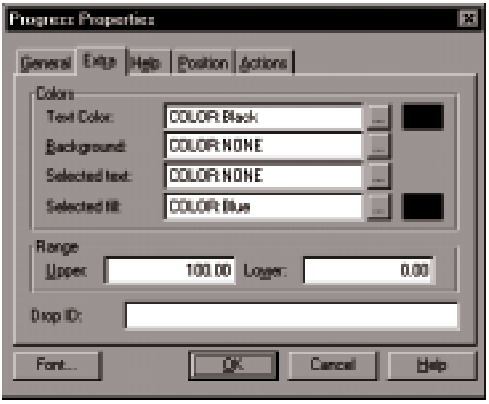
Extra Tab

1. Optionally, set colors for the progress bar.

   See *Common Control Attributes—Setting the COLOR Attribute.*

2. Specify the **Range** of values the progress bar displays.

   If omitted, the default range is from zero (0) to one hundred (100).



Colors

Text Color: COLOR:Black

Background: COLOR:NONE

Selected text: COLOR:NONE

Selected fill: COLOR:Blue

Range

Upper: 100.00 Lower: 0.00

Drop ID:

3. In the **Drop ID** field, optionally type up to sixteen (16) comma delimited signatures.

The **Window Formatter** adds the DROPID attribute to your control. The DROPID indicates this control is a valid target for drag and drop operations. The signature is a string constant that identifies which types of drag and drop operations are valid for the control.

Drag and drop capability means the user can select an item in one window or control, hold down the left mouse button, drag the item to another window or control, and release the mouse button, dropping the item onto the other window or control, which can then look at the item that was dropped and do something with it.

Implementation of this capability requires that the source control have a DRAGID attribute with a signature that matches the target's DROPID signature, and that the procedures that drive each window have appropriate source code to process the drag and drop events. See the *Language Reference* for more details and examples.

Help Tab

See *Common Control Attributes—Setting Help Attributes*.

Position Tab

See *Common Control Attributes*s*—Setting the AT Attribute*.

Image Properties

The IMAGE control lets you place bitmapped and vector images in a window. The bitmap file formats supported are .BMP, .CUR, .PCX, .GIF, .ICO and .JPG. The vector file format supported is .WMF. Clarion supports up to 16.7 million color resolution.

GIF images are substantially faster than .JPG images although they do not compress quite as well. 32-bit JPG images are significantly faster than 16-bit JPG images. Clarion does not support printing of ICO images (because Windows doesn't support it). BMP images are fast, but take lots of space because they are not compressed.

> **Tip:** **Use the PALETTE attribute on your window to ensure ample color support for your images. The PALETTE attribute specifies how many colors you want this window to use when it is the foreground window. This is applicable in hardware modes where a palette is in use and spare colors are available. See the *Language Reference* for details.**

The **Image Properties** dialog provides the following options.

General Tab

1. Select a graphics file.

   Type in a file name, or press the ellipsis (**...**) button to the right of the **File** field to select a graphics file using the standard open file dialog.

2. Place a field equate label in the **Use** field.

   The field equate label references the image in program statements. See *Common Control Attributes—Setting the USE Attribute.*

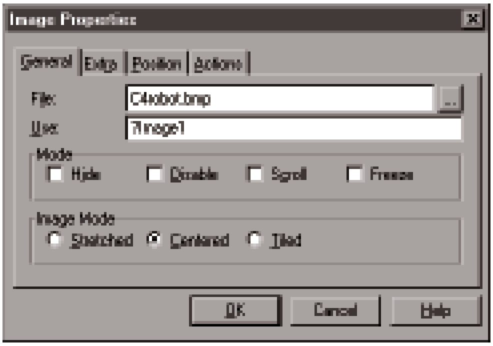
3. Select an **Image Mode**

   Choose from:

   Stretched
   The image expands (or contracts) to fill the image control.

   Centered
   The image displays at its default size, centered within the image control.

   Tiled
   The image displays at its default size and is repeated as many times as needed to fill the image control.



File: C4robot.bmp

Use: ?Image1

Mode

□ Hide □ Disable □ Scroll □ Freeze

Image Mode

○ Stretched ● Centered ○ Tiled

4. Set **Mode** options.

   See *Common Control Properties—Setting Control Modes* above.

Extra Tab

1. Optionally add scroll bars.

   Check the **Horizontal**, **Vertical**, or both check boxes to add scroll bars if the image is larger than the control size.

Position Tab

See *Common Control Attributes—Setting the AT Attribute.*

Tip: For IMAGE controls, Default displays the picture at the size it was created.

Line Properties

The LINE control lets you place a straight line in your windows. You may set the line's color, thickness and position. The Line control cannot receive focus, nor can it generate events.

The **Line Properties** dialog contains the following options.

General Tab

1. Place a field equate label in the **Use** field.

   The field equate label references the line in program statements. See *Common Control Attribute*s*—Setting the USE Attribute.*

2. Optionally, set the line's thickness.

   Type a point value in the **Line Width** spin control. The default is 1 point.

3. Set **Mode** options.

   See *Common Control Properties—Setting Control Modes*.

Extra Tab

1. Specify the **Line Color**.

   Type a valid color equate or press the ellipsis button to select a color. The standard **Color** dialog appears. Select a color by clicking on the color selection square, or add a custom color. See *Common Control Attributes—The Color Dialog*.

Tip: To heighten the "chiselled" look of a 3D window with a menu bar, place a white line control of 0 height, and FULL width, starting at point 0,0. The line sets off the gray area of the window against the menu bar.

Position Tab

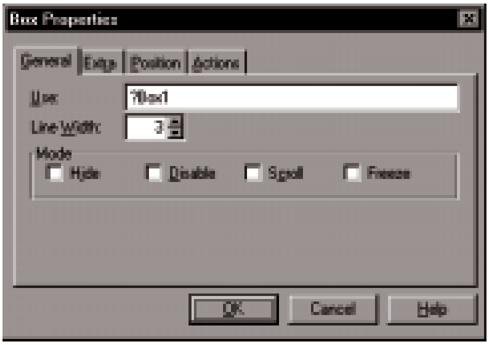
See *Common Control Attribute*s*—Setting the AT Attribute.*

Box Properties

The Box control lets you place a square or rectangle in your windows. You may fill the box with a color, and specify a border color. You may also specify it should have rounded corners. The Box control cannot receive focus, nor can it generate events.

The **Box Properties** dialog contains the following options.

General Tab



Use: ?Box1

Line Width: 3

Mode
□ Hide □ Disable □ Scroll □ Freeze

1. Place a field equate label in the **Use** field.

   The field equate label references the Box in program statements. See *Common Control Attribute*s*—Setting the USE Attribute.*

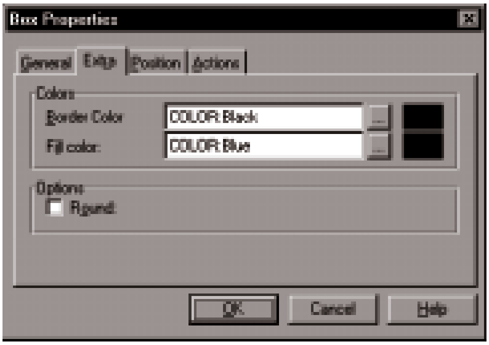
2. Optionally, set the thickness of the box's border.

   Type a point value in the **Line Width** spin control. The default is 1 point.

3. Set **Mode** options.

   See *Common Control Properties—Setting Control Modes*.

Extra Tab



Colors

Border Color: COLOR:Black

Fill color: COLOR:Blue

Options
□ Round

1. Specify the **Fill** and **Border** colors.

   Type a valid color equate or press the ellipsis button to select a color. The standard **Color** dialog appears. Select a color by clicking on the color selection square, or add a custom color. See *Common Control Attributes—The Color Dialog*.

2. Optionally specify the Box should appear with rounded corners by checking the **Rounded** box.

   The corners of the box are rounded.

Position Tab

See *Common Control Attributes—Setting the AT Attribute.*

> **Tip: While you can set the size of the box and other graphic controls by manually typing in coordinates, it is much easier to draw it directly in the Windows Formatter.**

Ellipse Properties

The ELLIPSE control lets you place a circle or ellipse in your windows. You may fill the ellipse with a color, and specify a border color. The ellipse control cannot receive focus, nor can it generate events.

The **Ellipse Properties** dialog contains the following options.

General Tab

1. Place a field equate label in the **Use** field.

   The field equate label references the ellipse in program statements. See *Common Control Attributes—Setting the USE Attribute.*

2. Optionally, set the thickness of the ellipse's border.

   Type a point value in the **Line Width** spin control. The default is 1 point.

3. Set **Mode** options.

   See *Common Control Properties—Setting Control Modes*.

Extra Tab

1. Specify the **Fill** and **Border** colors.

   Type a valid color equate or press the ellipsis button to select a color. The standard **Color** dialog appears. See *Common Control Attributes—The Color Dialog.*

Position Tab

See *Common Control Attributes—Setting the AT Attribute.*

> **Tip: While you can set the size of the ellipse and other graphic controls by manually typing in coordinates, it is much easier to drag its handles in the Window Formatter.**

Panel Properties

The PANEL control lets you place a raised or depressed rectangular panel in your windows. You may color the panel. The panel control cannot receive focus, nor can it generate events.

The **Panel Properties** dialog contains the following options.

General Tab

1. Place a field equate label in the **Use** field.

   The field equate label references the panel in program statements. See *Common Control Attribute*s*—Setting the USE Attribute.*

2. Set **Mode** options.

   See *Common Control Properties—Setting Control Modes*.

Extra Tab

1. Specify the **Fill** color.

   Type a valid color equate or press the ellipsis button to select a color. The standard **Color** dialog appears. See *Common Control Attributes—The Color Dialog.*

2. Optionally, set the BEVEL attribute for the PANEL.

   The BEVEL attribute gives a three dimensional look to the panel. The panel appears raised, depressed, or both.

   Outer A positive value makes the panel appear raised above the plane of the window. The higher the value, the further the panel is raised. A negative value makes the panel appear depressed below the plane of the window. The bevel effect begins at the outer border of the panel.

   Inner A positive value makes the panel appear raised above the plane of the window. The higher the value, the further the panel is raised. A negative value makes the panel appear depressed below the plane of the window. The bevel effect begins immediately inside the outer bevel.

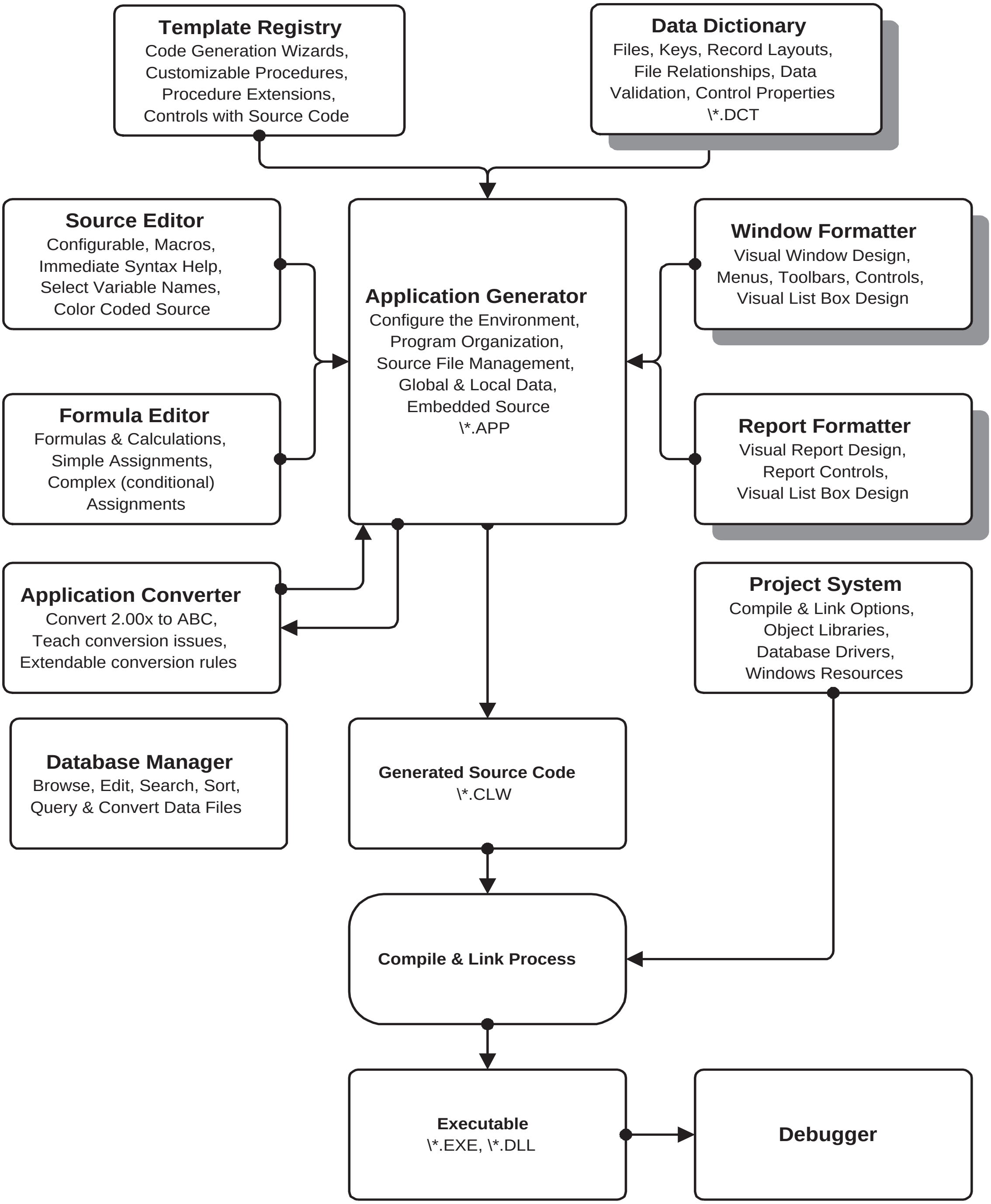
   Style A USHORT, whose sixteen bits define the style (but not the size) of the each of the four edges of the PANEL. The STYLE attribute gives you very fine control over the bevel appearance. See the *Language Reference* for an explanation of the meaning of each bit.

Position Tab

See *Common Control Attributes*—*Setting the AT Attribute*.

Tip: **While you can set the size of the panel and other graphic controls by manually typing in coordinates, it is much easier to draw it directly in the Window Formatter.**

6 - CUSTOM CONTROLS



Overview

This chapter shows you how to set custom control properties. Custom controls include both OLE and VBX controls.

Custom controls are defined outside the Clarion Development Environment and may be either interactive or non-interactive.

- OLE controls—are "containers" for linked or embedded *objects* from other programs such as Excel, Word, or PowerPoint. The programs must be registered OLE servers. The objects may be documents created by the server programs such as spreadsheets, word processing documents, or slide presentations. Or the objects may be "directors" that let the server program appear to run from within your Clarion program.
- OLE controls—are also "containers" for OCX (ActiveX) controls. OCXs are add-in controls from third party vendors. You can place these controls with the **Window Formatter** once you register the OCXs with Windows.
- VBX controls—are add-in controls from third party vendors in Visual Basic Extension format. Clarion supports VBX controls compatible with Microsoft Visual Basic 1.0. You can place these controls with the **Window Formatter** once you register the .VBX libraries with the Clarion development environment.

OLE Controls

OLE Container Overview

The OLE control lets you place OLE (Object Linking and Embedding) objects or .OCX controls in your windows. See *Object Linking and Embedding* in the *Language Reference* for a complete discussion of this topic.

OLE

By using an OLE Server, you add the full power of the server to your application with very little programming on your part. However, your end users must be licensed to use the OLE Server, and their hardware should support the substantial additional resources required by the server. For example, your Clarion application may run very efficiently on a 486 processor with 4M RAM, but if it calls Excel, PowerPoint, and/or Word, your end users must have licenses for Excel, PowerPoint and Word, and their hardware should provide reasonable performance for those programs (a Pentium with 16 to 32M RAM).

OCX

Using an OCX usually requires a lot of embedded source code or hand code. However, many OCXs are distributable to end users with no additional license fee, and most OCXs require considerably fewer resources than do typical OLE Servers.

> **Note: The Application Generator does not automatically generate source code to support any particular functionality for the OLE control. You must embed some code, or use the OLEControl template to get functionality from the OLE control; otherwise it is display only.**

Embedded Code

When you use an OLE control to access an OLE Server, at a minimum you probably should embed code to activate and deactivate the server (OCXs are automatically activated). This gives your user full access to all the functionality of the server. You can embed additional code to control exactly how the server behaves and which server functions are available to your user. Consult the server’s documentation for more information.

You may activate the server based on a button push, a menu selection, a list selection, a mouse click, etc. To activate the server, use PROP:DoVerb. For example:

```
!Activate Server in its default mode.
?OleControl{PROP:DoVerb} = 0
```

or

```
!Activate Server in open mode (in its own separate window).
?Ole{PROP:DoVerb} = -2
```

To deactivate an in-place activated server, use PROP:Deactivate. You can add a "deactivate" menu item to the OLE control and embed the deactivation code in the *Control Event Handling—Accepted* embed point for the menu item. For example:

```
?Ole{PROP:Deactivate}   !Deactivate OLE Server
```

More embedded code is generally required to implement an OCX. The code required is specific to the OCX and usually requires a thorough understanding of the OCX's operation.

OLE Control Menus

OLE controls may have MENUs, just like WINDOWs have. To define the OLE control's menu, in the **Window Formatter**, select the OLE control, then choose **Menu ➤ New Menu**. Define the menu the same way you define menus for your windows (see *Window Formatter—Menu Editor*).

The OLE control's menu merges with the window's menu. At runtime, if you activate the OLE server "in-place," the server's menus merge with the window's menu as well.

Compound Storage File v. Linked or Embedded Document

Specifying a Storage File (OPEN('Filename\!ObjectName)) is very similar to specifying a linked or embedded document (LINK('Filename')). In both cases, the OLE object is a link to an external file. However, there are some significant differences:

- The OLE Server can manipulate a linked or embedded document file independent of your application, but cannot do so with a Storage File.
- A Storage File can contain multiple objects, for example, two spreadsheets, or a spreadsheet and a word document.
- The OLE container's properties are saved in (and restored from) the Storage File, but not in the linked or embedded document.

OLE Control Properties

Use the **OLE Properties** dialog to specify how the OLE control is declared in the generated source code. This section describes each prompt in the **OLE Properties** dialog. The pages following this section provide instructions for using this dialog for specific OLE implementations.

General Tab

1. Type a field equate label in the **Use** field.

   The field equate label references the OLE control in program statements.

2. Select a sizing attribute from the **Sizing Mode** drop-down list. The **Window Formatter** adds the attribute to the OLE declaration. Choose from:

   Default Adds no sizing attribute. Zoom is the default.

   Clip The OLE declaration gets the CLIP attribute. The OLE object only displays what fits into the area defined by the OLE container control's AT attribute. If the object is larger than the control, only the top left corner displays.

   Stretch The OLE declaration gets the STRETCH attribute. The OLE object stretches to completely fill the area defined by the OLE container control's AT attribute. The object's aspect ratio is lost.

   AutoSize The OLE declaration gets the AUTOSIZE attribute. The OLE object automatically resizes when the OLE container control's AT attribute changes at runtime.

   Zoom The OLE declaration gets the ZOOM attribute. The OLE object stretches to fill the area defined by the OLE container control's AT attribute. The object's aspect ratio is maintained.

3. Check the **Compatible Mode** box to specify a compatibility mode of 1 for objects that require it (such as the Windows bitmap editor). Clear this box to let the compatibility mode default to 0.

4. Check the **32-Bit** box to tell the **Window Formatter** you want to work with 32-bit objects.

   This restricts the **Object Type** list to 32-bit objects and allows the **Window Formatter** to load 32-bit Ole Servers.

5. Use the **Control Type** group of controls to specify the object type for the OLE control declaration. Select one of the three radio buttons: **Ole, Document**, or **OCX**.

Control Type—Ole

When you select the **Ole** radio button, the **Object Type** list contains registered OLE objects. The OLE structure gets the CREATE or OPEN attribute.

Object Type
Select from a list of registered OLE objects, such as Excel Spreadsheets, Word documents, PowerPoint Slides, etc. to create or open.

Tip: **When you select an OLE server, the Window Formatter automatically loads it. This can be time consuming during the window design process. Design and draw your window first, then specify your OLE control last, or specify the server at runtime using property syntax rather than at design time.**

Storage File

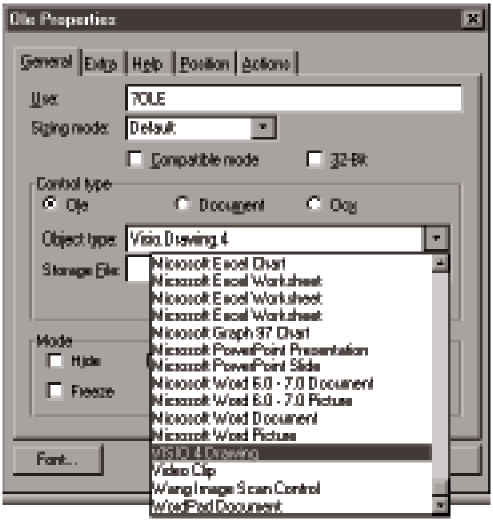
Specifies the name of an OLE Compound Storage File (.OLR) and the object within it to OPEN. Separate the filename and the object name with a backslash and exclamation point:

```
FileName\!ObjectName.
```

If this field is blank, your application creates a new OLE object (spreadsheet, slide, etc.) of the specified type.

The **Window Formatter** supplies a default value for this field when you use it to create the Compound Storage File. See *Window Formatter Menus—Popup Menu—Open*.

The OLE Server (PowerPoint, Word, etc.) can access and manipulate the object in the compound storage file, but only through your application. Therefore, use a compound storage file when you want to limit your user's access to the object.



Ole Properties

Use: ?OLE

Sizing mode: Default

□ Compatible mode □ 32-Bit

Control type: ● Ole ○ Document ○ Ocx

Object type: Visio Drawing 4

Storage File:

Mode: □ Hide □ Freeze

General
Extra
Help
Position
Actions
Microsoft Excel Chart
Microsoft Excel Worksheet
Microsoft Excel Worksheet
Microsoft Excel Worksheet
Microsoft Graph 97 Chart
Microsoft PowerPoint Presentation
Microsoft PowerPoint Slide
Microsoft Word 6.0 - 7.0 Document
Microsoft Word 6.0 - 7.0 Picture
Microsoft Word Document
Microsoft Word Picture
VISIO 4 Drawing
Video Clip
Wang Image Scan Control
WordPad Document
Font...

Tip: RIGHT-CLICK **the OLE control in the Window Formatter, then choose Open from the popup menu to activate the specified OLE server. This lets you build the object with the OLE server, and it automatically specifies a default filename\!objectname in the Storage File field.**

When the object in the storage file is opened, the saved version of the OLE container properties are reloaded.

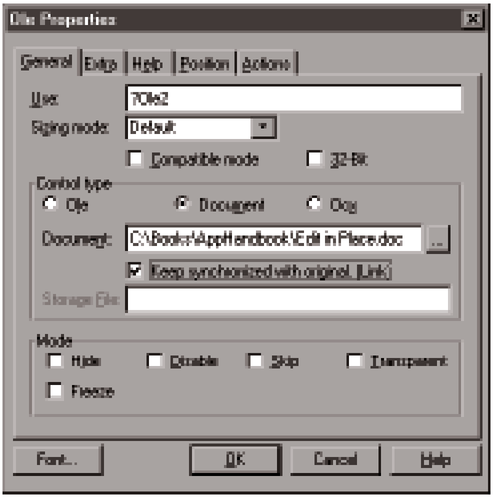
Control Type—Document

When you select the **Document** radio button, the **Document** entry box replaces the **Object Type** list, and the **Keep synchronized with original** check box appears. The OLE structure gets the DOCUMENT or LINK attribute.

Document

Type the full pathname of the document file or press the ellipsis button (...) to select the file from the Windows file dialog. A document file is a file that is associated with a specific OLE server, so the application can activate that server at runtime (e.g., MYBUDGET.XLS is associated with Excel).

If the filename has no path, the application looks for it in the current directory. The document file should be installed on the end user's machine in the specified directory.



Use: ?Ole2

Sizing mode: Default

□ Compatible mode □ 32-Bit

Control type

○ Ole ● Document ○ Ocx

Document: C:\Books\AppHandbook\Edit in Place.doc ...

☑ Keep synchronized with original. (Link)

Storage File:

Mode

□ Hide □ Disable □ Skip □ Transparent

□ Freeze

Keep synchronized with original (Link)

Check this box to add the LINK attribute (see the *Language Reference*) to the OLE structure. This tells the server to update the original document with any changes made through your application.

Clear this box to add the DOCUMENT attribute (see the *Language Reference*) to the OLE structure. This tells the server *not* to update the original document with changes made through your application.

Tip: The default DoVerb action ({PROP:DoVerb}=0) may depend on whether the object is a link (LINK) or an embed (DOCUMENT).

Storage File

Specify the name of an OLE compound storage file (.OLR) and the embedded document within it to open (a compound storage file may be specified for embedded documents (DOCUMENT) but not for linked documents (LINK)). Separate the filename and the object name with a backslash and exclamation point:

```
FileName\!ObjectName
```

The **Window Formatter** supplies a default value for this field when you use it to create the compound storage file. See *Window Formatter Menus—Popup Menu—Open*.

The OLE Server can access and manipulate the object in the compound storage file, but only through your application. Therefore, use a compound storage file when you want to limit your user's access to the document.

Tip: **RIGHT-CLICK the OLE control in the Window Formatter, then choose Open from the popup menu to activate the OLE server. This lets you build the document with the OLE server, and it automatically specifies a default filename\!object name in the Storage File field.**

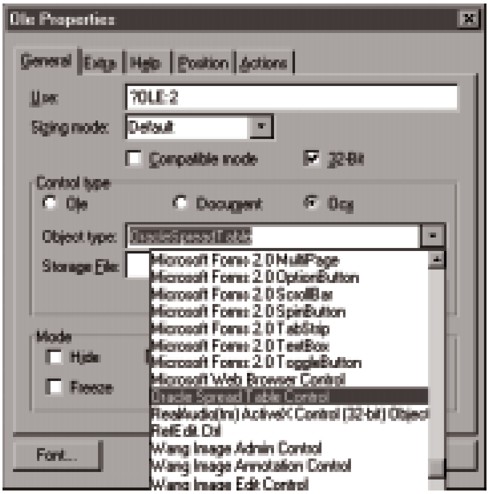
When the object is opened, the saved version of the OLE container properties are reloaded.

Control Type—OCX

When you select the t**OCX**radio button, the **Object Type** list contains registered OCX objects. The OLE declaration gets the CREATE or OPEN attribute.

Object Type

Select from a list of registered OCX objects to CREATE or OPEN.



Ole Properties

Use: ?OLE:2

Sizing mode: Default

□ Compatible mode ☑ 32-Bit

Control type

○ Ole ○ Document ● Ocx

Object type: [illegible]

Storage File:

Mode

□ Hide

□ Freeze

General
Extra
Help
Position
Actions
Microsoft Forms 2.0 MultiPage
Microsoft Forms 2.0 OptionButton
Microsoft Forms 2.0 ScrollBar
Microsoft Forms 2.0 SpinButton
Microsoft Forms 2.0 TabStrip
Microsoft Forms 2.0 TextBox
Microsoft Forms 2.0 ToggleButton
Microsoft Web Browser Control
Oracle Spread Table Control
RealAudio(tm) ActiveX Control (32-bit) Object
RefEdit.Ctrl
Wang Image Admin Control
Wang Image Annotation Control
Wang Image Edit Control
Font...

Storage File

Specify the name of an OLE compound storage file (.OLR) and the object within it to open. Separate the filename and the object name with a backslash and exclamation point:

```
FileName\!ObjectName
```

If you leave this field blank, your application creates a new OCX object of the specified type.

Tip: **RIGHT-CLICK the OLE/OCX control in the Window Formatter, then choose Custom from the popup menu to activate the Custom Properties dialog for the OCX. This records any property changes and adds them as attributes to the OLE control declaration statement.**

If a property is set that cannot be represented in text form, a storage file is created to store the property. The Window Formatter specifies a default filename\!object name in the Storage File field.

Tip: **You can create a Storage File at runtime for the OCX by issuing a**

```
?OCXControl{PROP:SaveAs}='FileName\!ObjectName'
```

while the OCX is active within your program. With this technique, you can reopen an OCX in the state the user left it!

When the object is opened, the saved version of the OLE container properties are reloaded.

6. Set **Mode** options.

   See *Common Control Properties—Setting Control Modes*.

Extra Tab

7. Optionally, set colors for the OLE control.

   Type a valid color equate or press the ellipsis button to select a color. The standard **Color** dialog appears. See *Common Control Attributes—The Color Dialog.*

Tip: **Specify a background color for the OLE control. This makes the boundaries of the control more visible and makes the control look like an action control rather than a static control.**

Help Tab

See *Common Control Attributes—Setting Help Attributes*.

Position Tab

See *Common Control Attributes—Setting the AT Attribute.*

OLE Controls with OCXs

You must register your OCX with your Windows operating system before you can use it. Consult your OCX documentation for instructions on registering the OCX.

> **Tip: Some OCX installation programs register the OCX automatically upon installation.**

In addition, many OCXs require that you use them only in accordance with a license agreement.

Generally, your program talks to the OCX with property assignments and callback functions which, among other things, capture OCX events. Therefore, you should be familiar with the OCX's properties, events, and operations. Consult your OCX documentation for this information.

The Clarion runtime library calls the callback functions whenever it needs to pass on information concerning the OCX. You must register your callback functions before the runtime library can call them. You may unregister the functions if you need to. Clarion supplies procedures for this purpose as illustrated below and in *OLE (.OCX) Custom Controls* in the *Language Reference*.

ActiveX Controls, License Files, and Compound Storage Files

ActiveX Controls and License Files

Many ActiveX controls operate on two levels: the development/design level, and a more limited end user level. Typically, a developer buys a license which allows him to operate the control at the design level and to redistribute the control at the end user level. This type of license is often enforced through an associated license file. (.LIC or .LPK). That is, the license file must be present at design time to successfully manipulate the properties of the ActiveX control during application development; but it need not (and should not) be present at runtime to support the more limited end user operation of the control. Thus, the ActiveX control (.OCX and/or .DLL) may be distributed with the application to end users, but the license file need not be distributed, and in fact, cannot legally be distributed under the terms of the typical license agreement.

Clarion and License Files

Clarion lets you implement ActiveX controls in your application with the following methods. All the methods require the presence of the license file at design time and the last method requires the presence of the license file at runtime as well; therefore you cannot use the last method for distribution to end users**.**

1) *Within* the WINDOW structure, name the ActiveX control.

   You can use the Window Formatter to populate an OLE container control (or control template), then select the ActiveX control from the **Object Type** drop-down list in the **OLE Properties** dialog.

2) *Within* the WINDOW structure, name a compound storage file (.OLR) that contains an instance of the ActiveX control.

   You can use the Window Formatter to create the compound storage file: populate an OLE container control (or control template), select the ActiveX control from the **Object Type** drop-down list in the **OLE Properties** dialog, then choose the **Custom** command on the right-click popup menu.

3) *Outside* the WINDOW structure, name a compound storage file that contains an instance of the ActiveX control. For example:

```
CODE
?CalendarObject{PROP:Open} = ‘Calendar.OLR\!MeetingSchedule’
```

4) *Outside* the WINDOW structure, name the ActiveX control. This method requires the presence of a license file at runtime—a violation of the typical license agreement; therefore you cannot use this method for distribution to end users**.**

```
CODE
?CalendarObject{PROP:Create} = ‘GraphDemoLib.GraphDemo’
```

Benefits of Compound Storage Files

Compound storage files are files that contain one or more OLE or ActiveX objects, including any special property settings for those objects such as fonts, colors, sizes, initial values, etc. Microsoft established the standard format for these files, which in the Clarion implementation have a file extension of .OLR (by convention only).

You can use compound storage files to reduce and reuse the code needed to implement ActiveX objects in your application. In addition, if you always use compound storage files to implement your ActiveX controls, you will never need to distribute license files to end users in violation of your license agreement.

Compound storage files reduce code because the Clarion Window Formatter automatically creates compound storage files for OLE and ActiveX objects. You can save the custom properties of a specific object by visually manipulating the object (such as a spreadsheet or a calendar control) with its own powerful and easy to use methods, rather than laboriously hand coding property assignments. For example, you can format an Excel spreadsheet using Excel menu commands, toolbar buttons, etc., then save the spreadsheet in a compound storage file, then open the spreadsheet in its current state from the compound storage file. Without the compound storage file, you would have to issue a series of Excel property assignments to the Excel spreadsheet object to initialize it to the desired state.

Compound storage files reuse code because you can reference a single compound storage file many times within a single application or from multiple applications. The property settings saved within the compound storage file are reused with each reference.

VBX Controls

VBX controls are "add-in" controls sold by many third party vendors. These perform a wide variety of tasks, from sliders and gauge controls to TWAIN image capture add-ins. The **Window Formatter** lets you directly place these controls once you "register" the .VBX libraries.

The specific VBX control format Clarion supports is the Microsoft Visual Basic control format, normally given the .VBX extension. There is one important limitation:

- Clarion supports .VBX properties *compatible with Microsoft Visual Basic 1.0*. VBX controls which require VB 2.0 or higher are incompatible.

  This is in line with other non-Visual Basic platforms, such as the Microsoft Foundation Classes v. 2.0. The biggest difference between level one and level two or higher .VBX's is that the latter contain "hooks" into the MS Access database engine which ships with Visual Basic 2.x and higher. The level number refers to the VB version number.

Tip: If the vendor description of a .VBX doesn't specifically state whether the control is designed for Visual Basic 1, you can immediately identify a level two or higher control if they identify it as a "data bound" control.

VBX control libraries usually require a license file (*.LIC or *.DEM) before you can add the control to your applications. The library vendor provides the file when you buy the library. When you distribute the application to your end users, you distribute the .VBX file only, not the license file.

Additionally, when you ship the .VBX file to your end users, follow the library vendor's instructions as to where to place the .VBX control file(s).

Registering .VBXs

Before you can place a VBX control in a window, you must register the .VBX file which contains it. To do so:

1. Install the .VBX and any associated .dlls and license files into a directory in the system path.

   Some .VBX vendors install their .VBX's to the Windows System directory, while others prefer private directories. When you install a .VBX library to your hard drive, make a note of where you put it so you can locate it with the Open File dialog.

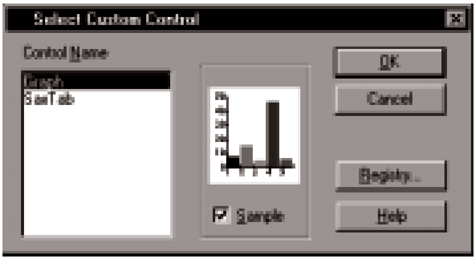
2. From Clarion's main menu, choose **Setup ➤ VBX Custom Control Registry**.

3. Press the **Add** button in the **VBX Custom Control Registry** dialog box.
4. Navigate to the .VBX file and select it in the **Add Custom Control** dialog, then press **OK**.
5. Press **OK** to close the **VBX Custom Control Registry** dialog.

Adding VBX Controls to a Window or Report

1. From the **Window Formatter** or the **Report Formatter**, select the VBX Control tool, or choose **Control ➤ VBX Control**, then CLICK in the window or report.

   This opens the **Select Custom Control** dialog. This dialog lets you select controls from the **VBX Custom Control Registry.** Highlight the control you want. When you highlight a control, if the **Sample** box is checked, the dialog displays the control with its default settings.



2. Press the **OK** button to return to the **Window Formatter.**
3. RIGHT-CLICK the VBX control then select **Properties** from the popup menu.

VBX Properties

The **VBX Control Properties** dialog contains the following prompts.

1. Optionally type a label for the control in the **Text** field.

   If the control supports a label, it will appear as part of the control. In practice, most controls will require you to specify a title label as a Visual Basic Control property (see *VBX Operation*).

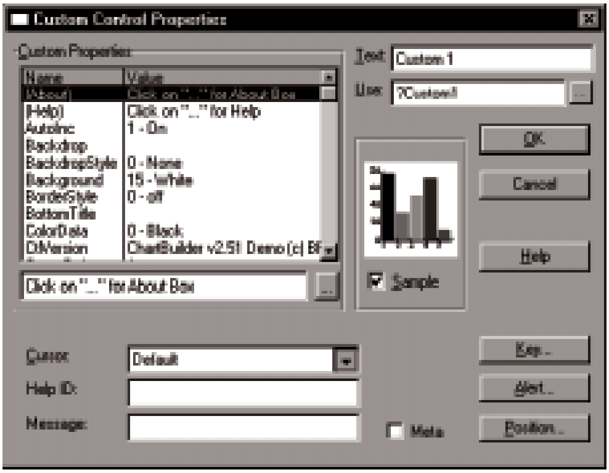
2. Type a field equate label or variable name in the **Use** field.

   The variable will nominally receive the value of the control. If the control accepts user entry, you will more likely retrieve the value entered by the user by accessing a Visual Basic Control property (see *VBX Operation*).

   .VBX's also generate a specific event (EVENT:vbxevent). The event represents a string message sent from the .VBX to the Clarion application. You can examine the event (see *VBX Operation*).

3. In the **Custom Properties** entry field, type start-up properties for the control.

   The **Custom Properties** list appears at the left of the dialog. It displays the Visual Basic Control properties and their default values. If you enter a start-up value in the dialog, the **Window Formatter** automatically adds it to the Clarion language statement that places the control in the window.



Name	Value
(About)	Click on "..." for About Box
(Help)	Click on "..." for Help
AutoInc	1 - On
Backdrop	
BackdropStyle	0 - None
Background	15 - White
BorderStyle	0 - off
BottomTitle	
ColorData	0 - Black
DllVersion	ChartBuilder v2.51 Demo (c) BF

Click on "..." for About Box

Text: Custom 1

Use: ?Custom1

☑ Sample

Cursor: Default

Help ID:

Message:

□ Meta

   When you highlight a Visual Basic Control property in the list, either an edit box or a drop-down list appears below the property list. Type a value or variable in the edit box or choose from the drop-down list.

   The documentation from the .VBX vendor should describe the Visual Basic Control properties you may set. Read on to find out how to change these properties at runtime and how to retrieve user input from the VBX control (see *VBX Operation*).

4. Optionally, specify the appearance of the cursor, the Help ID, and Message text.

   See *Common Control Attributes—Setting Help Attributes*.

5. Optionally, specify the KEY and ALRT attributes.

   See *Common Control Attributes—Setting the KEY Attribute* and *Setting the ALRT Attribute.*

6. Optionally, press the Position button to specify the AT attribute, plus any Mode attributes.

   See *Common Control Attributes—Setting the AT Attribute* and *Setting Control Modes.*

7. Optionally, check the **Sample** box to display the control with default settings.

8. Optionally, check the **Meta** box to generate a Windows metafile (.WMF) for reports.

When adding a .VBX control to a report, this specifies that the print engine generates a metafile (.WMF) to represent the control. This metafile may be previewed and printed just like any other report generated metafiles.

VBX Operation

The .VBX file acts as a mostly self-contained external library. When the application loads it into memory, you can exchange information between the application and the custom control with the properties. The Visual Basic Control properties are a message map.

The .VBX properties are the most common means by which a non-VB application uses a VBX’s functionality. Think of a property as a variable which both the application and the VBX can access.

If both the application and the VBX monitor the property, they can use it to signal each other. When the value of the property changes, it is a signal that something may need to be done. Each VBX has its own properties. You find out what properties are available by reading the VBX Vendor’s documentation.

For example, assume a VBX has a property called ‘CellColor,’ which indicates the background color of a grid cell. If the application wants to know what the current color is, it retrieves the value in the property called ‘CellColor.’ Usually, it works the other way, too. If the application changes the value of ‘CellColor’ from blue to red, then the VBX updates the window control and changes the color.

Tip: The Visual Basic Control properties are usually documented with a leading dot. Drop the dot when accessing it from the Clarion application.

VBXs and Property Syntax

The section above notes how to set the start-up properties for a control with the **Window Formatter**. At other times you’ll want to alter the properties at runtime, and of course, retrieve values after user input.

❏ To alter properties at runtime, use the property syntax. Access the control’s properties by referring to the specific property in quotes:

```
?vbx{ ‘VBProperty’ } = value
```

❏ To retrieve the current value of a Visual Basic Control property, use the property syntax this way:

```
value = ?vbx{ ‘VBProperty’ }
```

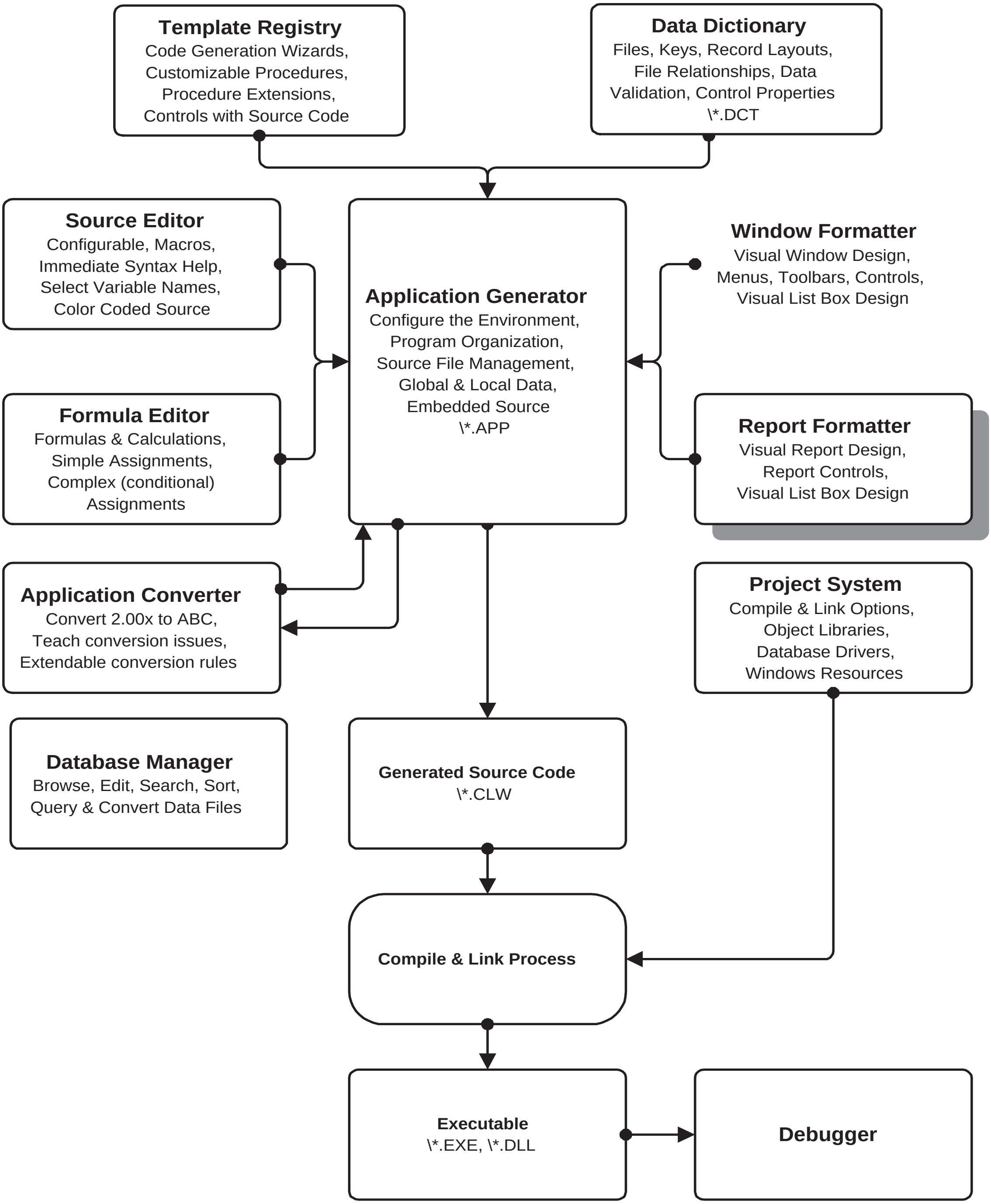
VBX Events

Besides properties, the other “channel” by which the .VBX “talks” to your application is with events. A .VBX might trigger an event, for example, if the end user double clicks on a particular part of it. When the event occurs, the .VBX generates a string (up to 255 characters) naming the event. The .VBX vendor’s documentation lists the possible events the control may generate.

Your application can examine the event, and take appropriate action by interrogating PROP:VBXevent. When working with the Application Generator, you place code similar to the example below at the embed point labelled “Control Event Handling, (VBXevent).” For example:

```
SomeString = ?vbx{PROP:VBXevent}
IF SomeString = ‘UserWantsToDoX’
  SomeProcedure
END
```

7 - REPORT FORMATTER


Template Registry
Code Generation Wizards, Customizable Procedures, Procedure Extensions, Controls with Source Code
Data Dictionary
Files, Keys, Record Layouts, File Relationships, Data Validation, Control Properties
*.DCT
Source Editor
Configurable, Macros, Immediate Syntax Help, Select Variable Names, Color Coded Source
Application Generator
Configure the Environment, Program Organization, Source File Management, Global & Local Data, Embedded Source
*.APP
Window Formatter
Visual Window Design, Menus, Toolbars, Controls, Visual List Box Design
Formula Editor
Formulas & Calculations, Simple Assignments, Complex (conditional) Assignments
Report Formatter
Visual Report Design, Report Controls, Visual List Box Design
Application Converter
Convert 2.00x to ABC, Teach conversion issues, Extendable conversion rules
Project System
Compile & Link Options, Object Libraries, Database Drivers, Windows Resources
Database Manager
Browse, Edit, Search, Sort, Query & Convert Data Files
Generated Source Code
*.CLW
Compile & Link Process
Executable
*.EXE, *.DLL
Debugger

Overview

Using the **Report Formatter**, you *visually* lay out your application’s reports. The **Report Formatter** automatically generates and places all the code structures necessary to produce the reports. Preview the reports without actually generating any code or data.

This chapter provides an overview of Clarion’s page oriented print engine, and a comprehensive treatment of each feature of the **Report Formatter**.

Clarion has many powerful reporting features and we want to systematically cover all of them. However, you the developer, probably have a particular report you need to produce by yesterday! To read about those features that will help you produce your particular report right now, turn to the *Creating Reports* chapter.

For even more examples of how to create a variety of report effects and features, see the *Creating Reports* chapter in the *Learning Clarion* book.

Clarion’s Report Engine

Before learning how to create a report using the **Report Formatter**, it is important to understand how Clarion executes a report—in other words, the division of labor between the print engine and your source code, plus the order in which the print engine processes the different sections of your report.

Usually, your source code need only contain the REPORT data structure, plus the appropriate file I/O and a PRINT statement. The REPORT data structure is generated for you by the **Report Formatter**, and the Application Generator handles the file I/O and the PRINT statement, that is, unless you prefer to hand code.

REPORT Structures

The REPORT data structure contains all the information necessary for formatting and printing each report page. A REPORT data structure may contain five sub-structures: FORM, HEADER, DETAIL, FOOTER, and BREAK. Each BREAK structure can contain it’s own HEADER, nested BREAK, DETAIL, and FOOTER.

These structure names carry traditional positioning connotations, however, Clarion provides complete flexibility in positioning the FORM, page HEADER, and page FOOTER structures. Further, BREAKs, break HEADERs, break FOOTERs, and DETAIL structures can print anywhere within the detail print area, an area defined by the REPORT’s AT attribute. The REPORT’s AT attribute, and the detail print area may be defined with the **Report Properties** dialog, or the **Report Formatter’s Page Layout View**.

Processing Sequence

What you should remember about these structures is not so much where they are printed (they may be printed anywhere you place them) but *when* they are composed, that is, the *order* in which they are composed when your report prints.

Once you know the order in which the report sections are composed at print time, you can use them better. Page placement does *not* affect the order in which the print engine composes the sections of the report. You can place a page FOOTER *above* a DETAIL, and because the page FOOTER is composed *after* the report engine processes all the DETAILs on the page, you can place a page total at the *top* of the page!

The following procedure illustrates the order in which each report structure is composed. Assume a report containing one BREAK structure, with a DETAIL section inside it:

1. The print engine composes the FORM, but does not send it to the print spooler yet. The print engine composes the FORM only once.
2. The print engine composes the page HEADER.
3. The print engine composes the group HEADER for the first group.
4. The print engine composes the DETAIL section as many times as necessary to fill the first page.

   If a BREAK *occurs on the page*:

5. The print engine composes the group FOOTER for the first group.
6. The print engine composes the group HEADER for the next group.
7. The print engine composes the DETAIL section for the next group of records, continuously checking for more group BREAKs.

   At the bottom of the page:

8. The print engine checks for widows, increments the page count, and checks the next page for orphans.
9. The print engine composes the page FOOTER.
10. The print engine *sends the entire page* to the print spooler.
11. Except for step 1, the print engine repeats this procedure for all additional pages.

Report Formatter Interface

Opening the Report Formatter

You can access the **Report Formatter** from the Application Generator or from the Text Editor.

To open the Report Formatter from the Application Generator

1. From the **Application Tree,** RIGHT-CLICK the report then choose **Report** from the popup menu (or from the **Procedure Properties** dialog press the **Report** button).

   This opens the **Report Formatter**. You're now ready to define the report section by section.

> **Tip: When you open the Report Formatter with the Application Generator, you have full access to the data dictionary "short cuts"—the Populate Field toolbox, the Populate menu, the Dictionary Fields tool, etc.**

To open the Report Formatter from the Text Editor

1. Open a source code document.
2. Locate a blank line in the data section and place the cursor there.

   This is where the **Report Formatter** places the Clarion language REPORT structure.
3. Choose **Edit ➤ Format Structure** from the menu.

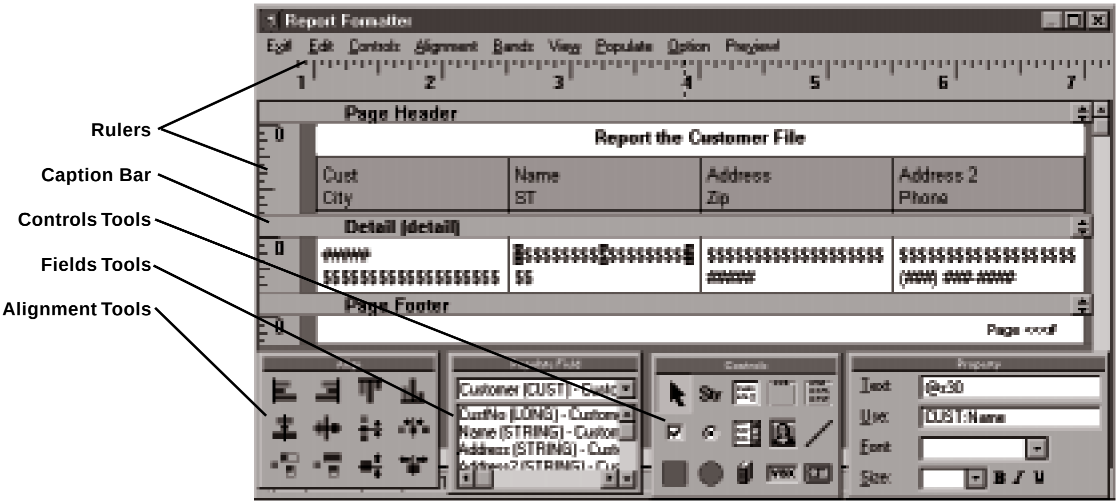
   You may also press the keyboard accelerator, CTRL+F. This opens the **New Structure** dialog.
4. From the **New Structure** dialog, choose *Report (portrait)* or *Report (landscape)*.

   This opens the **Report Formatter**. You're now ready to define the report.

> **Tip: To edit an existing report from the Text Editor, open the source code file and place the cursor on any line within the REPORT structure, then choose Edit ➤ Format Structure from the menu, or press ctrl+f.**

Band View

When you first open the **Report Formatter**, your report appears in Band View. That is, each REPORT section (HEADER, BREAK, DETAIL, FOOTER, and FORM) appears in a separate "band" inside the window. This is true even though the report sections may actually overlap when printed.



Rulers

To quickly determine each section's position, look at the rulers. The horizontal (X axis) ruler shows the position relative to the left edge of the page. The vertical (Y axis) rulers show the positioning relative to the top of each band. The measurement units are set in the **Report Properties** dialog (see *Report Formatter Structures and Properties*).

Caption Bars

Each report section has it's own caption bar. Each caption bar displays the band type and an expand/contract button at the far right. Break section caption bars also display the name of the variable the section breaks on.

Show/Hide Button ⇕

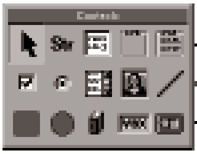
To expand or contract the report band, click on the expand/contract button at the far right of its caption bar.

Report Formatter Toolboxes

Controls Toolbox

The **Controls** Toolbox works much like a palette of drawing tools, such as the toolbox in the Windows Paintbrush accessory. Simply CLICK on an icon in the toolbox, then CLICK inside the band you wish to add the control to. The upper left hand corner of the control is placed at the intersection of the cursor crosshair when you CLICK the mouse.

The **Controls** Toolbox appears by default when you start the **Report Formatter**. Hide or re-display the **Controls** toolbox by choosing **Option ➤ Show Toolbox.** Resize the toolbox by placing the cursor on its border, then CLICK and DRAG. Move the toolbox by CLICKING and DRAGGING its caption bar.



No Tool	**String**	**Text Box**	**Group Box**	**Option Box**
Check Box	**Radio Button**	**List Box**	**Image**	**Line**
Box	**Ellipse**	**Field**	**VBX**	**Template**

All the controls in the toolbox are available from the **Controls** menu and the **Populate** menu.

Tip: **Position the cursor over any tool and wait for half a second. A tool tip appears telling you the type of control this tool creates.**

Drops the currently selected control tool.

Places a STRING control on the report under construction. See *Controls and Their Properties—String Properties*.

Places a TEXT control on the report under construction. See *Controls and Their Properties—Controls and Their Properties—Text Properties.*

Places a GROUP control (group box) on the report under construction. See *Controls and Their Properties—Group Box Properties*.

Places an OPTION control (OPTION structure, which appears as a group box with radio buttons) on the report under construction. See *Controls and Their Properties—Option Box Properties*.

Places a CHECK control on the report under construction. See *Controls and Their Properties—Check Box Properties*.

Places a RADIO control on the report under construction. See *Controls and Their Properties—Radio Button Properties*.

Places a LIST control on the report under construction. See *Controls and Their Properties—Creating List Boxes*.

Places an IMAGE control (graphic image) on the report under construction. See *Controls and Their Properties—Image Properties*.

Places a LINE control on the report under construction. See *Controls and Their Properties—Line Properties*.

Places a BOX control on the report under construction. See *Controls and Their Properties—Box Properties*.

Places an ELLIPSE control on the report under construction. See *Controls and Their Properties—Ellipse Properties*.

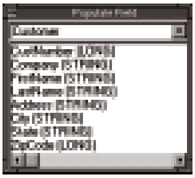
Places a control associated with a Data Dictionary field or memory variable on the report under construction.

Places a VBX control (Visual Basic custom control) on the report under construction. See *Custom Controls—VBX Controls*.

Places a control template on the report under construction (adds one or more controls, *along with associated source code*). See *Control Templates* in the *Application Handbook* for more information.

Populate Field Toolbox

The **Report Formatter** contains a floating **Populate Field** toolbox. This toolbox lets you quickly "populate" a report with fields from your data dictionary files or with memory variables.



Customer

CustNumber (LONG)
Company (STRING)
FirstName (STRING)
LastName (STRING)
Address (STRING)
City (STRING)
State (STRING)
ZipCode (LONG)

To populate a field, choose a file or variable scope from the drop-down list, then DOUBLE-CLICK the field or variable you want to appear on your report. The field is automatically aligned. To place the field manually, CLICK *once* on the field, then CLICK in the desired location. The type of control (string, check box, radio button, etc.) is determined by the settings for this particular field in the Data Dictionary.

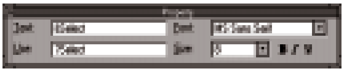
Display or hide the **Populate Field** toolbox by choosing **Option ➤ Fieldbox.** Resize the **Populate Field** toolbox by placing the cursor on the border of the box. When the cursor changes to a double headed arrow, CLICK and DRAG. Move the toolbox by CLICKING and DRAGGING it's caption bar.

You may also populate a report with fields from your files by using the **Populate** menu or by using the Dictionary Fields tool in the **Controls** toolbox.

Property Toolbox

The **Report Formatter's Property** toolbox lets you quickly specify the appearance and content of the *text* on each control within the report. Control the typeface, size, style, and content of all your report text using standard word processor buttons and drop-down lists.

In the **Text** field, type the control's text or a picture token. See *Controls and Their Properties—The Picture Editor* for more information.

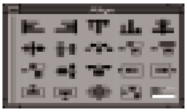


Display or hide the **Property** toolbox by choosing **Option ➤ Show Propertybox.** Resize the **Property** toolbox by placing the cursor on the border of the box. When the cursor changes to a double headed arrow, CLICK and DRAG. Move the toolbox by CLICKING and DRAGGING it's caption bar.

Align Toolbox

The **Report Formatter's Align** toolbox lets you quickly, professionally, and precisely align the controls in your report. Select the controls to align (CTRL+CLICK selects multiple controls), then click on the appropriate alignment tool. All the alignment actions are also available from the **Alignment** menu.

Tip: For most alignment functions, the first control(s) selected (blue handles) are aligned with the last control selected (red handles). That is, the last control selected is the anchor control. It doesn't move, the others do.



Align Left	**Align Right**	**Align Top**	**Align Bottom**
Align Vertical	**Align Horizontal**	**Spread Vertical**	**Spread Horizontal**
Center Vertical	**Center Horizontal**	**Same Size**	**Same Height**

Display or hide the **Align** toolbox by choosing **Option ➤ Show Alignbox.** Resize the **Align** toolbox by placing the cursor on the border of the box. When the cursor changes to a double headed arrow, CLICK and DRAG. Move the toolbox by CLICKING and DRAGGING it's caption bar.

Tip: Position the cursor over any tool and wait for half a second. A tool tip appears telling you the type of alignment this tool will accomplish.

Aligns the left borders of the selected controls with the left border of the last control selected (red handles).

Aligns the right borders of the selected controls with the right border of the last control selected (red handles).

Aligns the top borders of the selected controls with the top border of the last control selected (red handles).

Aligns the bottom borders of the selected controls with the bottom border of the last control selected (red handles).

Along a vertical axis, aligns the centers of the selected controls with the center of the last control selected (red handles).

Along a horizontal axis, aligns the centers of the selected controls with the center of the last control selected (red handles).

Equalizes the vertical space between the selected controls.

Equalizes the horizontal space between the selected controls.

Makes all selected controls the same height and width as the last control selected (red handles).

Makes all selected controls the same height as the last control selected (red handles).

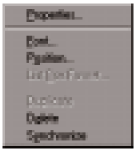
As a group (relative positions of selected controls don't change), centers the selected controls vertically within the report band.

As a group (relative positions of selected controls don't change), centers the selected controls horizontally within the report band.

Report Formatter Menus

Popup Menu

Access the popup menu by RIGHT-CLICKING a band or a control. The popup menu on the **Report Formatter** lets you manipulate and customize the report sections, *and* the controls within them, depending on which is selected.



- ❏ To select a *control*, place the cursor on the control then RIGHT-CLICK.
- ❏ To select a *report section*, place the cursor anywhere on the band, but *not* on a control, then RIGHT-CLICK.

Tip: Many of the popup menu commands are also available on the Report Formatter Edit menu.

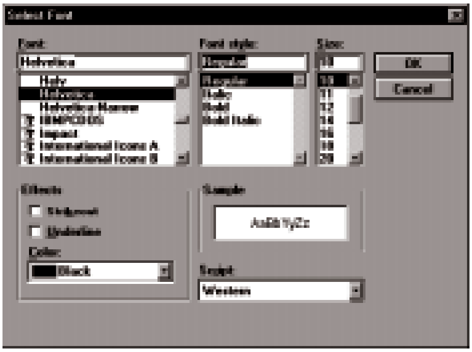
Following is a description of the popup menu choices:

Properties

Opens the properties dialog for the selected report section or control. See the discussion of the selected item for more information on it's properties.

Font

Calls the **Select Font** dialog which lets you select the font (typeface), size, style (such as bold or italic), color, and font effects (underline and strikeout) for all the controls in the report section. You may override the section font by setting a different font in the Properties dialog for any specific control. As you choose options, the dialog displays a sample of the selected font.



Font: Helvetica

Font style: Regular

Size: 10

Color: Black

Script: Western

Position

Opens the properties dialog to the **Position** tab. See the discussion of the selected item for more information on it's positioning. Generally, you may also visually position items by CLICKING and DRAGGING, and by using the Alignment tools.

List Box Format

Specify the appearance and functionality of a list box control. See *Window Formatter—List Box Formatter* for more information.

Duplicate

Copies the currently selected control to the cursor position.

Delete

Deletes the selected control or report band. Alternatively, press the DELETE key.

Synchronize

Applies the control attributes specified in the Data Dictionary to the selected control, or if a band is selected, to all the controls in the band. The attributes are applied as specified in the **Synchronization** tab of the **Application Options** dialog. See *Application Generator—Configuring the Application Generator*.

Edit Menu

The **Edit** menu in the **Report Formatter** lets you manipulate and customize the report and the controls in the report.

> **Tip: Many of the Edit menu commands are also available on the popup menu that you access by right-clicking on the control or the report band.**

Next Band

Selects the next report band in sequence.

Delete Band

Deletes the selected report band. Alternatively, press the DELETE key. This deletes the band *and* all controls in it.

Report Properties

Opens the **Report Properties** dialog.

Selected Properties

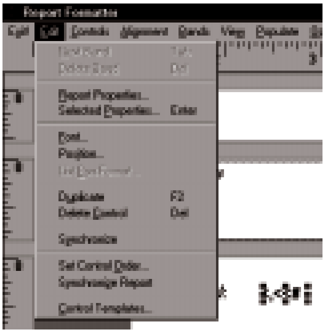
Opens the properties dialog for the selected section or control. See the discussion of the selected item for more information on it's properties.

Font

Calls the **Select Font** dialog which lets you select the font (typeface), size, style (such as bold or italic), color, and font effects (underline and strikeout) for all the controls in the report section. You may override the section font by setting a different font in the Properties dialog for any specific control. As you choose options, the dialog displays a sample of the selected font.

Position

Opens the properties dialog to the **Position** tab. See the discussion of the selected item for more information on it's positioning. You may visually position items by CLICKING and DRAGGING, and by using the Alignment tools.



List Box Format

Specify the appearance and functionality of a list box control. See *Window Formatter—List Box Formatter* for more information.

Duplicate

Copies the currently selected control to the current mouse cursor position.

Delete Control

Deletes the selected control. Alternatively, press the DELETE key.

Synchronize

Applies the control attributes specified in the Data Dictionary to the selected control, or if a band is selected, to all the controls in the band. The attributes are applied as specified in the **Synchronization** tab of the **Application Options** dialog. See *Application Generator—Configuring the Application Generator*.

Set Control Order

Opens the **Order Control** dialog, which displays all controls on the report in a hierarchical list. Reorder the controls by selecting a control and pressing the ▲ and ▼ buttons to move the control up or down within the list.

Tip: When overlapping one control over another, for example, text over a box, choose Edit ➤ Set Control Order to ensure the underlying control is printed before the overlying control; otherwise the overlying control may be obscured.

Synchronize Report

Applies the control attributes specified in the Data Dictionary to the selected to all the controls in the report. The attributes are applied as specified in the **Synchronization** tab of the **Application Options** dialog. See *Application Generator—Configuring the Application Generator*.

Control Templates

To add Extension templates to the procedure, or to edit the Control and Extension template prompts for the procedure, choose the **Control Templates** command. This opens the **Extension and Control Templates** dialog where you can add Extension templates, and you can edit either Control or Extension templates. See *Control Templates* in the *Application Handbook* for more information.

Controls Menu

The **Controls** menu lists the same controls that appear in the Controls Toolbox, except Control Template and Dictionary Fields (See *Populate Menu*). Executing a command from the **Controls** menu is identical to clicking on the corresponding toolbox icon. The menu serves as a convenience. See *Report Formatter Toolboxes*. Also see *Controls and Their Properties*.

Alignment Menu

The **Alignment** menu lists the same Alignment tools that appear in the Align Toolbox. Executing a command from the **Alignment** menu is identical to clicking on the corresponding toolbox icon (see *Report Formatter Toolboxes*). The menu provides the following additional options.

Make Same Width

Makes all selected controls the same width as the last control selected (red handles).

Bands Menu

The Bands menu lets you add new report sections or bands to your report. Choose from the following:

Page Header

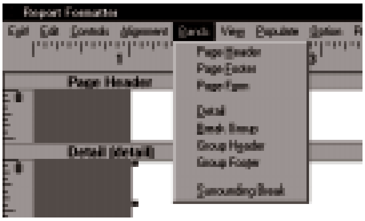
Adds a page header to your report. The header is the first section of the page to be composed. Typically, you place the report title, graphics and other "introductory" elements in the header.

Page Footer

Adds a page footer to your report. The footer is the last section of the page to be composed. Typically, you place a page number and page totals in the footer.

Page Form

Adds a "preprinted" form to your report. The form is composed once and remains constant from page to page. Typically, you display "overlays" or fixed data such as graphics and field labels in the form, then print the variable data in other bands.



Detail

Adds detail lines to your report The DETAIL structure is the "body" of the report. It contains the lowest level of data to be printed.

Break Group

Adds a new detail, break, group header and group footer to your report. Place the crosshair where you want the new group of bands to appear, then CLICK. This opens the **Break Properties** dialog. Specify the variable to break on then press **OK.**

Group Header

Adds a group header to an existing break. Place the crosshair on the caption bar of the break you wish to modify, then CLICK. This opens the **Page/Group Header Properties** dialog.

Group Footer

Adds a group footer band to an existing break. Place the crosshair on the caption bar of the break you wish to modify, then CLICK. This opens the **Page/Group Footer Properties** dialog.

Surrounding Break

Adds a break around an existing detail. Place the crosshair on the detail you want to break on, then CLICK. This opens the **Break Properties** dialog. Specify the variable to break on then press **OK.**

View Menu

The **View** menu lets you toggle between **Band View** and **Page Layout View**, plus hide or display all bands at once.

Page Layout View

Accurately displays the positioning of report sections on a sample page. Lets you move and resize the page header, page footer, form, and print detail area by CLICKING and DRAGGING. See *Specifying Report Margins* for instructions on using the **Page Layout View**.

Band View

Displays each report section in a separate band with no overlapping. Lets you add controls to your report and set their properties.

Expand Bands

Collapses or expands all the bands at once.

Populate Menu

The **Populate Menu** appears in the **Report Formatter** only when the Application Generator is active. It places an appropriate control in the report to display the selected data dictionary field or memory variable.

Dictionary Field

Lets you place a variable string control tied to a data dictionary field or a memory variable. When you CLICK in the report, the **Select Field** dialog opens. Select a field or variable, then CLICK in the report.

If you pre-formatted the field on the **Report** tab of the **Field Properties** dialog (for example, specifying a text control), the control you specified is placed, otherwise a string is placed.

Multiple Fields

Same as **Dictionary Field**, but in a repetitive fashion. Press the **Cancel** button after you have placed the last field.

Control Template

Adds one or more controls to your report, along with associated source code. See *Control Templates* in the *Application Handbook*.

Option Menu

The **Option** menu lets you display and hide the various **Report Formatter** toolboxes, zoom in and out, and apply grid behavior.

Zoom In

Magnifies the “view” in Preview mode.

Zoom Out

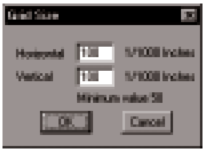
Reduces the “view” in Preview mode.

Snap to Grid

To turn grid snap on or off, choose the **Snap to Grid** command. Grid snap forces the upper left corner of new controls to align with a dot grid in the report. The grid is not printed; it is a design tool only.

Grid Size

To set the size of the grid units, choose the **Grid Size** command. You may enter different values for the X and Y axes.



Grid Size

Horizontal 100 1/1000 inches

Vertical 100 1/1000 inches

Minimum value: 50

To set the width and height spacing between the grid dots, enter values in the **Width** and **Height** fields in the **Grid Size** dialog. The values are in the measurement unit specified in the **Report Properties** dialog.

Show Toolbox

Displays or hides the Controls toolbox.

Show Alignbox

Displays or hides the Align toolbox.

Show Propertybox

Displays or hides the Property toolbox.

Show Fieldbox

Displays or hides the Populate Field toolbox.

Preview!

Preview! lets you try out various report formats without actually compiling and running the report. You can quickly “preview” alternative layouts for DETAILs, BREAKs, HEADERs, and FOOTERs, and you can see the effects of the page breaking options you have chosen.

The **Report Formatter** supplies test data in the format you specify. Fonts, sizes, colors, and positions of report controls are all displayed.

To simulate a report similar to the one your user will see

1. Choose the **Preview!** command.

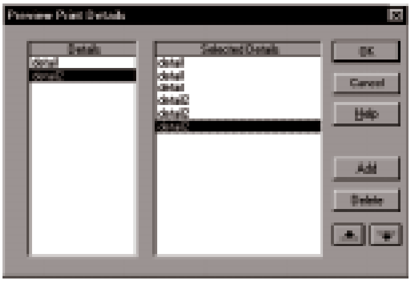
   This opens the **Preview Print Details** dialog. This dialog lets you generate “filler” data for your report. The data have no values, but serve as placeholders, so you can get a feel for the appearance of your finished report.

If you have more than one DETAIL, highlight one of them on the left side of the dialog.

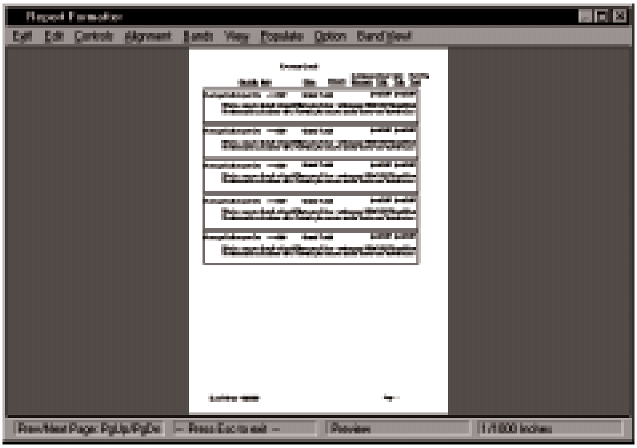
2. Press the **Add** button to generate filler data.

   Generate as many as you need. Some reports will have only one record per page, others will have many records. You can add enough records to overflow the page and preview the page breaking behavior of your report.

   You can even mix two or more DETAILs. Use the up and down buttons to rearrange the DETAIL placeholders.



3. Press the **OK** button to preview the report.



Preview page HEADERs and group HEADERs.

Preview DETAILs and page breaking behavior.

Preview typeface, size, style and color.

Preview Page Orientation.

4. Choose **Option** ➤ **Zoom in** for a magnified view.

5. To exit **Preview!** mode, press ESC, or press **Band View!**

   The **Report Formatter** reverts to Band View.

Report Structures and Properties

The REPORT structure contains all the information necessary to format and print each report page. Following is an example of a REPORT structure with empty headers, footer, and form, a break on “CustNumber,” and several variable strings in the detail. This structure was generated by the **Report Formatter**.

```
Report    REPORT,AT(1000,2000,6000,7000),PRE(RPT),FONT(‘Arial’,10,,),THOUS
            HEADER,AT(1000,1000,6000,1000)
            END
CustBreak   BREAK(CUS:CustNumber)
              HEADER,AT(,,,1000)
              END
Detail        DETAIL
                STRING(@n4),AT(125,52),USE(CUS:CustNumber)
                STRING(@S20),AT(125,208),USE(CUS:Company)
                STRING(@S20),AT(125,365),USE(CUS:Address)
                STRING(@S20),AT(125,531),USE(CUS:City)
                STRING(@S2),AT(125,688),USE(CUS:State)
                STRING(@n5),AT(125,844),USE(CUS:ZipCode)
              END
            END
            FOOTER,AT(1000,9000,6000,1000)
            END
            FORM,AT(1000,1000,6000,9000)
            END
          END
```

Report Properties

There are *two* **Report Properties** dialogs. One of these dialogs is associated with the Report Procedure template. This dialog is accessed with the **Report Properties** button in the **Procedure Properties** dialog and is discussed in the *Application Handbook*. See *Procedure Templates—Report Template*.

The other **Report Properties** dialog is associated with *every* report, whether or not the Report Procedure template is used. This dialog is accessed from the **Report Formatter** by choosing **Edit ➤ Report Properties**, and is discussed here. This dialog lets you set up basic report options, including page orientation, measurement units, detail print area, and paper size. We recommend setting these options *before* laying out other parts of your report.

The **Report Properties** dialog provides the following options.

General

Job Name

Names the print job, as listed in the Windows Print Manager application.

Label

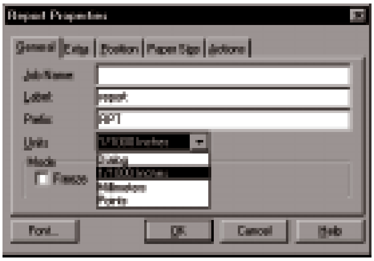
Type a valid Clarion label to name the REPORT data structure.

Prefix

Specifies the label prefix for the REPORT structure.

Units

Specifies the default measurement for all controls placed in the report. Choose Dialog Units, thousandths of Inches, Millimeters or Points.



Report Properties

Job Name:

Label: report

Prefix: RPT

Units: 1/1000 Inches

Mode

☐ Freeze

Freeze

"Freezes" all the controls on the report, so that subsequent data dictionary changes are not applied. You can override the #Freeze attribute for all controls, or for individual controls. See *Application Generator—Configuring the Application Generator*.

Extra

Preview

Specifies the name of a QUEUE which stores the filenames for the metafiles (*.WMF) generated for print preview. See the PREVIEW attribute in the *Language Reference*.

If you are using the Report Procedure template, simply check the **Print Preview** box in the other **Report Properties** dialog, and leave this entry blank.

Colors

Specifies a Text or Background color for the report. See *Window Formatter—Common Control Attributes—Setting the COLOR Attribute.*

Position

Defines the location and size of the report's *detail print area*, by filling in the REPORT's AT attribute. All DETAILS, group HEADERs, and group FOOTERs print *within the detail print area* at an offset *relative to the previous item printed*.

The measurement units for these values are specified on the **General** tab.

Top Left Corner

To set a precise starting point for your detail print area relative to the top left corner of the paper, specify Top Left Corner coordinates with this dialog. In effect, this establishes the *top and left margin* for your report detail print area. These settings may also be done visually by dragging the detail print area and its borders in the **Report Formatter's Page Layout View**.

Width/Height

To set the size of the detail print area, choose from the following options for the **Width** and **Height**.

Note: When changing a report from portrait to landscape, or vice versa, you should also change the width and/or height values in this dialog.

Default

Sets a value based on the Paper Size.

Fixed

To set a specific size, mark the Fixed choices.

Paper Size

Paper Size

Choose from over 40 standard sizes, or choose **Other** to specify a custom size.

Width

Specifies a custom paper width in units specified on the **General** tab.

Height

Specifies a custom paper height in units specified on the **General** tab.

Landscape

Specifies text-to-paper orientation. Landscape (checked box) aligns the report text parallel with the *longest* paper edges (the report is wider than it is tall). Portrait (cleared box) aligns the report text with the *shortest* paper edges (the report is taller than it is wide).

Tip: Before changing the paper size or orientation on an existing report, move controls and resize the FORM, HEADERs, FOOTERs, etc. to fit within the new paper dimensions. Otherwise, these items will fall outside the boundaries accessible by the Report Formatter.

Actions

Files

Access the File Schematic for the report.

Embeds

Access the **Embedded Source** dialog for the report. See *Application Generator—Embedded Source Code*.

Font

To set the default font for all controls in the report, press the **Font** button, then choose the font and style in the **Select Font** dialog. You may override the default by setting a different font in the **Properties** dialog for any specific control. The options you choose in the dialog become the parameters for the FONT attribute. As you choose options, the dialog displays a sample of the formatting.

Form

To specify constant text or graphics which print on every page, place it in the FORM. The print engine composes the FORM at the beginning of the print job; it does not update it with each new page. Therefore, the FORM is not suitable for holding variable data, or even a page number. You can, however, print fields from a control file, if you wish to print the same field contents on every page.

The form usually overlaps the other sections and may be used as a layer, to hold graphic frames or "preprinted" material into which the data from the other sections fit. You might use lines and boxes, for example, to divide the DETAIL into compartments, grouping data for the user. You may even create a 'greenbar' effect by alternating gray or light green color blocks. Another use for the FORM is to simulate a watermark.

Tip: **For best results when using a drawing tool to create a watermark, on, for example, a 300 DPI printer, set the fill for the watermark element to 10% gray, or light gray. At higher printing resolutions, try 20% gray.**

To add a form to your report, choose **Bands ➤ Page Form**.

Form Properties

RIGHT-CLICK on the form band, then choose **Properties** from the popup menu. The **Form Properties** dialog provides the following options.

General

Use Type a field equate label to reference the FORM in your source code.

Freeze

"Freezes" all the controls on the band, so that subsequent data dictionary changes are not applied. You can override the #Freeze attribute for all controls, or for individual controls. See *Application Generator—Setting Application Options*.

Extra

Colors

Specifies a Text or Background color for the band. See *Window Formatter—Common Control Attributes—Setting the COLOR Attribute.*

Position

Lets you set the location and size of the form by filling in its AT attribute. The measurement units for these values are specified on the **General** tab of the **Report Properties** dialog.

Top Left Corner

To set a precise starting point for your form relative to the top left corner of the paper, specify **Top Left Corner** coordinates with this dialog. In effect, this establishes the top and left margins for your form. These settings may also be accomplished visually by dragging the form and it's borders in the **Report Formatter's Page Layout View.**

Width/Height

To set the size of the page form, choose from the following options for **Width** and **Height**.

Default

Sets a value based on the Paper Size.

Fixed

To set a specific size, mark the Fixed choices.

> **Tip: When changing a report from portrait to landscape, or vice versa, you should change the width and/or height in this dialog.**

Actions

Files

Access the File Schematic for the report.

Embeds

Access the **Embedded Source** dialog for the report. See *Application Generator—Embedded Source Code*.

Font

Calls the **Select Font** dialog which lets you select the font (typeface), size, style (such as bold or italic), color, and font effects (underline and strikeout) for all the controls in the report section. You may override the section font by setting a different font in the Properties dialog for any specific control. As you choose options, the dialog displays a sample of the selected font.

Page Header

To specify text and data to *compose* at the start of each *page*, place it in the page HEADER. Remember, the page header may be physically positioned anywhere on the page, not just at the top.

Typically, the HEADER includes a report title and the page number. It is also a useful place to display your company logo. To add a page header to your report, choose **Bands** ➤ **Page Header**.

Page HEADER Properties

RIGHT-CLICK on the header band, then choose **Properties** from the popup menu. The **Header Properties** dialog provides the following options.

General

Use Type a field equate label to reference the page HEADER in executable code.

Freeze
“Freezes” all the controls on the band, so that subsequent data dictionary changes are not applied. You can override the #Freeze attribute for all controls, or for individual controls. See *Application Generator—Setting Application Options*.

Extra

Colors
Specifies a Text or Background color for the band. See *Window Formatter—Common Control Attributes—Setting the COLOR Attribute.*

Alone
Has no effect on a Page Header. See Group Header.

Absolute
Has no effect on a Page Header. See Group Header.

Position

Lets you set the location and size of the page header, by filling in its AT attribute. The measurement units for these values are specified on the **General** tab of the **Report Properties** dialog.

Top Left Corner
To set a precise starting point for your page header relative to the top left corner of the paper, specify **Top Left Corner** coordinates with this dialog. In effect, this establishes the top and left margins for your header. These settings may also be accomplished visually by dragging the header and it's borders in the **Report Formatter's Page Layout View.**

Width/Height
To set the size of the page header, choose from the following options for **Width** and **Height**.

Default
Sets a value based on the Paper Size.

Fixed
To set a specific size, mark the Fixed choices.

> **Tip: When changing a report from portrait to landscape, or vice versa, you should change the width and/or height in this dialog.**

Actions

Files
Access the File Schematic for the report.

Embeds
Access the **Embedded Source** dialog for the report. See *Application Generator—Embedded Source Code*.

Font

Calls the **Select Font** dialog which lets you select the font (typeface), size, style (such as bold or italic), color, and font effects (underline and strikeout) for all the controls in the report section. You may override the section font by setting a different font in the Properties dialog for any specific control. As you choose options, the dialog box displays a sample of the selected font.

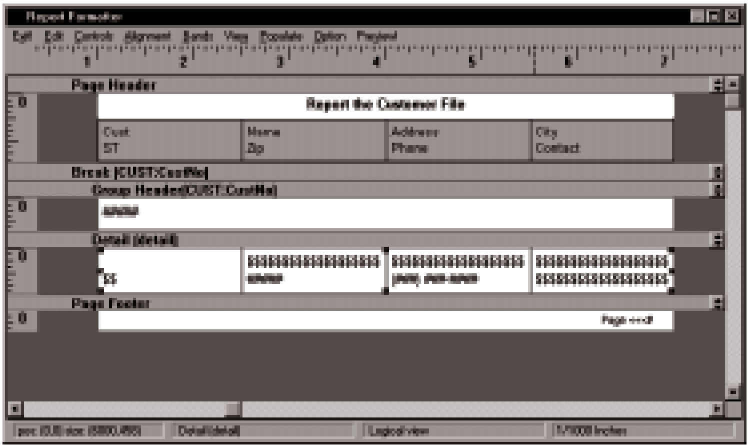
Group Breaks

Group breaks provide a means of grouping report data into sections and optionally displaying subheadings, subtotals, or other information associated with the group. Each group consists of a set of records all sharing the same value in the BREAK field.

When the value of the break variable changes, the old group ends and a new group begins. Ending a group means that the last DETAIL in the group is processed and the group FOOTER is composed. Beginning a group means that the group HEADER is composed and the first DETAIL in the new group is processed. In order to produce meaningful groups, the records must be sorted on the same fields the BREAKs are declared on. See *Specifying Sort Order* and *Specifying Nested Group Breaks* in the *Creating Reports* chapter.

Within a report, you may visually separate these groups with spaces, subtotals, headers, or other summary information, either above the group, below the group, or both. Displaying summary information for a group is accomplished by placing text or graphic controls in a group HEADER or FOOTER.

A BREAK structure may contain most of the same elements as a REPORT structure: group HEADERs, DETAILs, group FOOTERs, and BREAKs. Thus breaks may be nested, giving several levels of record grouping. The **Report Formatter** displays group breaks in an indented outline structure which lets you easily visualize nested group breaks.



Group BREAK Properties

This dialog lets you edit the properties of the group BREAK. RIGHT-CLICK on the break band, then choose **Properties** from the popup menu. The **Break Properties** dialog provides the following options.

General

Label

Type a valid Clarion label, naming the BREAK structure.

Variable
Press the ellipsis (...) button to select a break field. When the value of this variable changes, the old group ends and a new group begins.

Use Type a field equate label to reference the BREAK in source code.

Freeze
"Freezes" all the controls on the band, so that subsequent data dictionary changes are not applied. You can override the #Freeze attribute for all controls, or for individual controls. See *Application Generator—Configuring the Application Generator*.

Actions

Files
Access the File Schematic for the report.

Embeds
Access the **Embedded Source** dialog for the report. See *Application Generator—Embedded Source Code*.

Tip: If the break variable is a global or local variable, you must be sure that the executable code updates its value, so that it can generate a group break.

Group Header

Though they print on the page at the same time, the print engine composes the group HEADER before the group DETAIL. The group HEADER is a good place to identify the group, but not a good place to display group totals (because the group hasn't been processed yet).

Group HEADER Properties

This dialog lets you edit the properties of the group HEADER. RIGHT-CLICK on the header band, then choose **Properties** from the popup menu. The **Header Properties** dialog provides the following options.

General

Use Type a field equate label to reference the group HEADER in executable code.

Page Before
To print the group HEADER structure on a new page, check this box. This sets the PAGEBEFORE attribute. The print engine generates a page break before printing the page header.

The page number automatically increments, unless you reset it.

To reset the page number to a value you specify, type the value in the corresponding **New Page No:** field.

Page After

To print the group HEADER, then force a new page, check this box. This sets the PAGEAFTER attribute. The print engine generates a page break immediately after printing the page header.

The page number automatically increments, unless you reset it. To reset the page number to a value you specify, type the value in the corresponding **New Page No:** field.

With Prior

To prevent 'orphan' elements in a printout, type a numeric value in the this field. This sets the WITHPRIOR attribute. An 'orphaned' print element is the last element in a group and is separated from the rest of the group by a page break.

The value specifies the number of preceding group items that must appear on the same page as the last item.

With Next

To prevent 'widow' elements in a printout, type a value in this field. This sets the WITHNEXT attribute. A 'widowed' print element is the first element in a group and is separated from the rest of the group by a page break.

The value specifies the number of subsequent group items that must appear on the same page as the first item.

Freeze

"Freezes" all the controls on the band, so that subsequent data dictionary changes are not applied. You can override the #Freeze attribute for all controls, or for individual controls. See *Application Generator—Setting Application Options*.

Extra

Colors

Specifies a Text or Background color for the band. See *Window Formatter—Common Control Attributes—Setting the COLOR Attribute.*

Alone

Specifies that the group HEADER section always prints alone on a page, with no form, page header, or page footer. This adds the ALONE attribute to the structure which is useful for printing title pages and grand totals.

Absolute

Specifies that the group HEADER section always prints at the same fixed position on the page. This adds the ABSOLUTE attribute to the structure. See *Position* immediately below.

Tip: These settings are useful when your report has more than one DETAIL. DETAILs may be printed conditionally and may be used to print one-time only pages (title page or grand total page).

Position

Sets the location and size of the group header by specifying its AT attribute. The location is an offset *relative to the top left corner of the detail print area,* or to the last item printed in the detail print area. The measurement units for these boxes are specified on the **General** tab of the **Report Properties** dialog.

Top Left Corner

To set a precise starting point for your group header relative to the top left corner of the print detail area, or relative to the last item printed in the print detail area, specify **Top Left Corner** coordinates with this dialog.

Width/Height

To set the size of the page header, choose from the following options for **Width** and **Height**.

Default

Sets a value based on the Paper Size and print detail area (see also *Report Properties*)

Fixed

To set a specific position and size, mark the Fixed choices.

Tip: When changing a report from portrait to landscape, or vice versa, you should change the width and/or height in this dialog.

Actions

Files

Access the File Schematic for the report.

Embeds

Access the **Embedded Source** dialog for the report. See *Application Generator—Embedded Source Code*.

Font

Calls the **Select Font** dialog which lets you select the font (typeface), size, style (such as bold or italic), color, and font effects (underline and strikeout) for all the controls in the report section. You may override the section font by setting a different font in the Properties dialog for any specific control. As you choose options, the dialog displays a sample of the selected font.

Detail

To specify the data for the body of the report, place it in a DETAIL. This is typically where lowest level information is printed. High level, duplicate, or summary information is better suited to HEADER or FOOTER structures. To add a DETAIL, choose **Bands ➤ Detail.**

Note that a report may have multiple DETAILs that can be printed conditionally (see *Procedure Templates—Report Procedure* in the *Application Handbook*). This is useful for printing one-time only pages such as title pages or grand total pages (see *Creating Totals and Calculated Fields* in the *Creating Reports* chapter). You can also use embedded source statements to control which DETAIL to print at run-time. Each DETAIL structure requires its own PRINT statement.

The DETAIL prints within the print detail area defined by the REPORT's AT attribute. Additionally, any the group HEADERs, group FOOTERs, or group BREAKs also print inside the detail print area.

Tip: For best automatic handling when it comes to placing structures on the page, nest your DETAIL inside all other structures. For example, if you have two BREAK structures, one nested in the other, delete all DETAIL structures except the one nested inside the innermost BREAK.

DETAIL Properties

RIGHT-CLICK on the detail band, then choose **Properties** from the popup menu. The **Detail Properties** dialog provides the following options.

General

Label
Type a valid Clarion label, naming the DETAIL structure.

Use Type a field equate label to reference the DETAIL in your source code.

Page Before
To print the DETAIL structure on a new page, check this box. This sets the PAGEBEFORE attribute. The print engine generates a page break before printing the DETAIL.

The page number automatically increments, unless you reset it. To reset the page number to a value you specify, type the value in the corresponding **New Page No:** field.

Page After
To print the DETAIL, then force a new page, check this box. This sets the PAGEAFTER attribute. The print engine generates a page break immediately after printing the DETAIL.

The page number automatically increments, unless you reset it. To reset the page number to a value you specify, type the value in the corresponding **New Page No:** field.

Tip: To print a separate page for each record, place the variable strings and/or controls you wish in the DETAIL, and check the PAGEAFTER box in the Detail Properties dialog.

With Prior

To prevent 'orphan' elements in a printout, type a numeric value in the this field. This sets the WITHPRIOR attribute. An 'orphaned' print element is the last element in a group and is separated from the rest of the group by a page break.

The value specifies the number of preceding group items that must appear on the same page as the last item.

Tip: When placing subtotals or totals in a DETAIL, use the WITHPRIOR attribute to ensure they print with at least one member of the column above it when a page break occurs.

With Next

To prevent 'widow' elements in a printout, type a value in this field. This sets the WITHNEXT attribute. A 'widowed' print element is the first element in a group and is separated from the rest of the group by a page break.

The value specifies the number of subsequent group items that must appear on the same page as the first item.

Freeze

"Freezes" all the controls on the band, so that subsequent data dictionary changes are not applied. You can override the #Freeze attribute for all controls, or for individual controls. See *Application Generator—Configuring the Application Generator*.

Extra

Colors

Specifies a Text or Background color for the band. See *Window Formatter—Common Control Attributes—Setting the COLOR Attribute.*

Alone

Specifies that the DETAIL always prints alone on a page, with no form, page header, or page footer. This adds the ALONE attribute to the structure.

Absolute

Specifies that the DETAIL always prints at the same fixed position on the page. This adds the ABSOLUTE attribute to the structure. See *Position* immediately below.

Tip: **These settings are useful when your report has more than one DETAIL. DETAILs may be printed conditionally and may be used to print one-time only pages (title page or grand total page).**

Position

Lets you set the location and size of the detail by specifying the AT attribute. The location is an offset *relative to the top left corner of the detail print area,* or to the last item printed in the detail print area. The measurement units for these boxes are specified on the **General** tab of the **Report Properties** dialog.

Top Left Corner
To set a precise starting point for your detail relative to the top left corner of the print detail area, or relative to the last item printed in the print detail area, specify **Top Left Corner** coordinates with this dialog.

Width/Height
To set the size of the detail, choose from the following options for **Width** and **Height**.

Default
Sets a value based on the Paper Size and print detail area (see also *Report Properties*)

Fixed
To set a specific position and size, mark the Fixed choices.

Tip: **When changing a report from portrait to landscape, or vice versa, you should change the width and/or height in this dialog.**

Actions

Files
Access the File Schematic for the report.

Embeds
Access the **Embedded Source** dialog for the report. See *Application Generator—Embedded Source Code*.

Font

Calls the **Select Font** dialog which lets you select the font (typeface), size, style (such as bold or italic), color, and font effects (underline and strikeout) for all the controls in the report section. You may override the section font by setting a different font in the Properties dialog for any specific control. As you choose options, the dialog displays a sample of the selected font.

Group Footer

The group FOOTER, is composed after the group DETAIL. This is a good place to display group totals or counts.

Group Footer Properties

This dialog lets you edit the properties of the group FOOTER. RIGHT-CLICK on the footer band, then choose **Properties** from the popup menu. The **Footer Properties** dialog provides the following options.

General

Use Type a field equate label to reference the group FOOTER in your source code.

Page Before

To print the group FOOTER structure on a new page, check this box. This sets the PAGEBEFORE attribute. The print engine generates a page break before printing the group FOOTER.

The page number automatically increments, unless you reset it. To reset the page number to a value you specify, type the value in the corresponding **New Page No:** field.

Page After

To print the group FOOTER, then force a new page, check this box. This sets the PAGEAFTER attribute. The print engine generates a page break immediately after printing the group FOOTER.

The page number automatically increments, unless you reset it. To reset the page number to a value you specify, type the value in the corresponding **New Page No:** field.

With Prior

To prevent ‘orphan’ elements in a printout, type a numeric value in the this field. This sets the WITHPRIOR attribute. An ‘orphaned’ print element is the last element in a group and is separated from the rest of the group by a page break.

The value specifies the number of preceding group items that must appear on the same page as the last item.

With Next

To prevent ‘widow’ elements in a printout, type a value in this field. This sets the WITHNEXT attribute. A ‘widowed’ print element is the first element in a group and is separated from the rest of the group by a page break.

The value specifies the number of subsequent group items that must appear on the same page as the first item.

Freeze

"Freezes" all the controls on the band, so that subsequent data dictionary changes are not applied. You can override the #Freeze attribute for all controls, or for individual controls. See *Application Generator—Setting Application Options*.

Extra

Colors

Specifies a Text or Background color for the band. See *Window Formatter—Common Control Attributes—Setting the COLOR Attribute.*

Alone

Specifies that the group FOOTER always prints alone on a page, with no form, page header, or page footer. This adds the ALONE attribute to the structure.

Absolute

Specifies that the group FOOTER always prints at the same fixed position on the page. This adds the ABSOLUTE attribute to the structure. See *Position* immediately below.

Position

Lets you set the location and size of the group FOOTER by specifying the AT attribute. The location is an offset *relative to the top left corner of the detail print area*, or to the last item printed in the detail print area. The measurement units for these boxes are specified on the **General** tab of the **Report Properties** dialog.

Top Left Corner

To set a precise starting point for your group footer relative to the top left corner of the print detail area, or relative to the last item printed in the print detail area, specify **Top Left Corner** coordinates with this dialog.

Width/Height

To set the size of the page footer, choose from the following options for **Width** and **Height**.

Default

Sets a value based on the Paper Size and detail print area (see also *Report Properties*)

Fixed

To set a specific position and size, mark the Fixed choices.

Tip: When changing a report from portrait to landscape, or vice versa, you should change the width and/or height in this dialog.

Actions

Files
Access the File Schematic for the report.

Embeds
Access the **Embedded Source** dialog for the report. See *Application Generator—Embedded Source Code*.

Font

Calls the **Select Font** dialog which lets you select the font (typeface), size, style (such as bold or italic), color, and font effects (underline and strikeout) for all the controls in the report section. You may override the section font by setting a different font in the Properties dialog for any specific control. As you choose options, the dialog displays a sample of the selected font.

Page Footer

To specify text and data to generate at the end of each *page*, place it in the page FOOTER. Remember, the page footer may be physically positioned anywhere on the page, not just at the bottom. Typically, the page FOOTER includes totals, page numbers, print dates, etc. To add a page footer to your report, choose **Bands ➤ Page Footer**.

Page FOOTER Properties

RIGHT-CLICK on the footer band, then choose **Properties** from the popup menu. The **Footer Properties** dialog provides the following options.

General

Use Type a field equate label to reference the page FOOTER in executable code.

Freeze
"Freezes" all the controls on the band, so that subsequent data dictionary changes are not applied. You can override the #Freeze attribute for all controls, or for individual controls. See *Application Generator—Setting Application Options*.

Extra

Colors
Specifies a Text or Background color for the band. See *Window Formatter—Common Control Attributes—Setting the COLOR Attribute.*

Alone
Has no effect on a Page Footer. See *Group Footer*.

Absolute

Has no effect on a Page Footer. See *Group Footer*.

Position

Lets you set the location and size of the page footer, by filling in its AT attribute. The measurement units for these boxes are specified on the **General** tab of the **Report Properties** dialog.

Top Left Corner

To set a precise starting point for your page footer relative to the top left corner of the paper, specify **Top Left Corner** coordinates with this dialog. These settings may also be accomplished visually by dragging the footer and it's borders in the **Report Formatter's Page Layout View.**

Width/Height

To set the size of the page footer, choose from the following options for the **Width** and **Height**.

Default

Sets a value based on the Paper Size.

Fixed

To set a specific size, mark the Fixed choices.

Tip: When changing a report from portrait to landscape, or vice versa, you should change the width and/or height in this dialog.

Actions

Files

Access the File Schematic for the report.

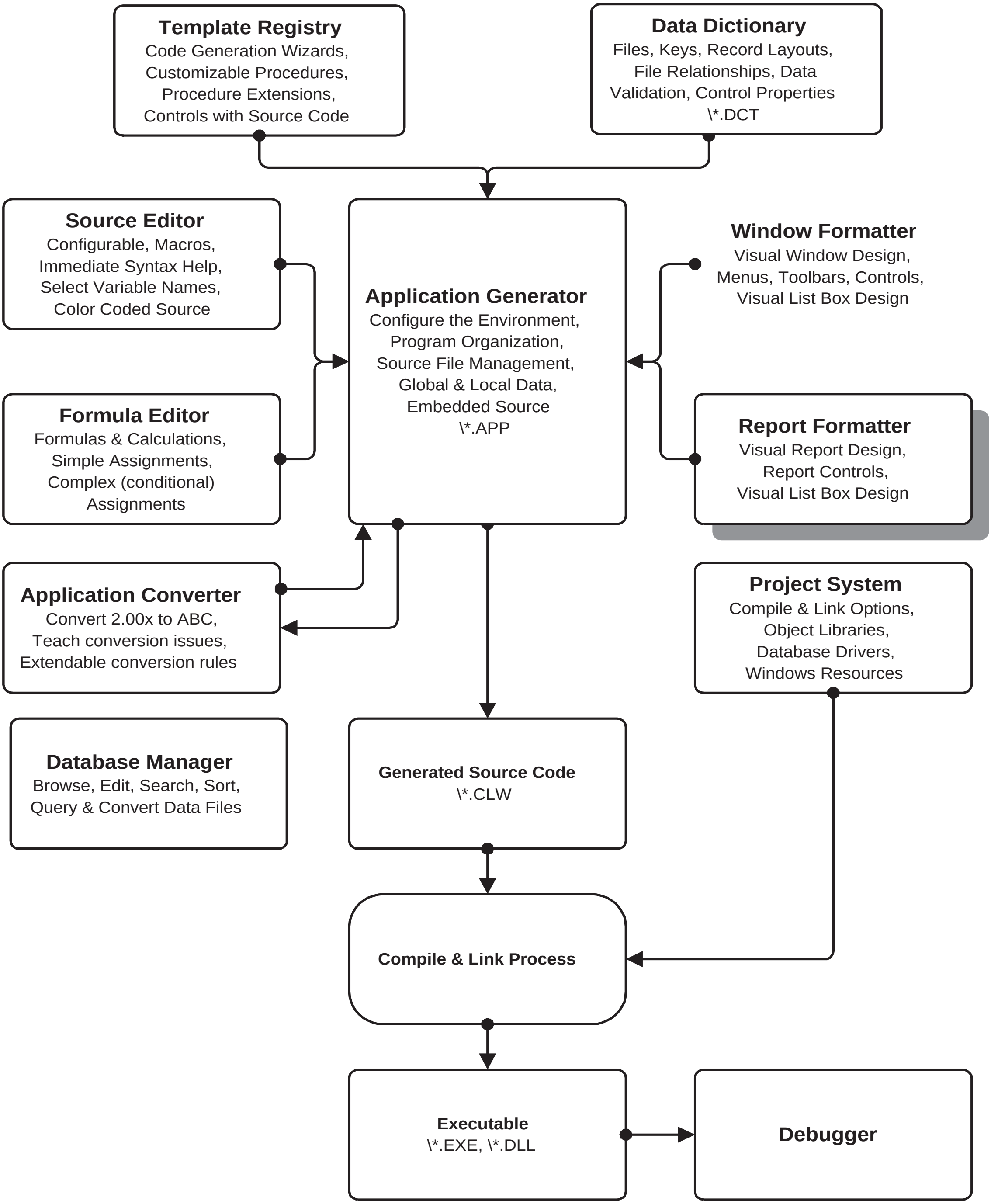
Embeds

Access the **Embedded Source** dialog for the report. See *Application Generator—Embedded Source Code*.

Font

Calls the **Select Font** dialog which lets you select the font (typeface), size, style (such as bold or italic), color, and font effects (underline and strikeout) for all the controls in the report section. You may override the section font by setting a different font in the Properties dialog for any specific control. As you choose options, the dialog displays a sample of the selected font.

8 - CREATING REPORTS


Template Registry
Code Generation Wizards, Customizable Procedures, Procedure Extensions, Controls with Source Code
Data Dictionary
Files, Keys, Record Layouts, File Relationships, Data Validation, Control Properties *.DCT
Source Editor
Configurable, Macros, Immediate Syntax Help, Select Variable Names, Color Coded Source
Application Generator
Configure the Environment, Program Organization, Source File Management, Global & Local Data, Embedded Source *.APP
Window Formatter
Visual Window Design, Menus, Toolbars, Controls, Visual List Box Design
Formula Editor
Formulas & Calculations, Simple Assignments, Complex (conditional) Assignments
Report Formatter
Visual Report Design, Report Controls, Visual List Box Design
Application Converter
Convert 2.00x to ABC, Teach conversion issues, Extendable conversion rules
Project System
Compile & Link Options, Object Libraries, Database Drivers, Windows Resources
Database Manager
Browse, Edit, Search, Sort, Query & Convert Data Files
Generated Source Code
*.CLW
Compile & Link Process
Executable
*.EXE, *.DLL
Debugger

About This Chapter

Clarion has many powerful reporting features and we want to systematically cover all of them. However, you the developer, probably have a particular report you need to produce by yesterday! You may only want to read about those features that will help you produce your particular report right now. Therefore, this chapter provides a "how to" reference for common report effects and functions such as sorting, totaling, page breaking, etc.

This chapter provides a list of report functions and effects that developers are frequently or occasionally asked to provide. It provides a general description of how to create specific report effects, plus a reference to those parts of the **Report Formatter** that are needed to produce the particular effect.

This chapter also suggests a proper sequence for building reports. For example, we discuss the features that define files and keys and establish page size, orientation, and margins *before* the features that lay out other parts of the report.

The list of effects is not comprehensive, but instead attempts to launch you quickly into doing a variety of report functions. Once you understand how to create these basic effects, you will be well on your way to creating truly dazzling special effects for your reports.

For more examples of how to create a variety of report effects and features, see the *Creating Reports* chapter in the *Learning Clarion* book. For systematic coverage of all Report Formatter tools and features, see the *Report Formatter* chapter in this book.

Common Reporting Tasks

Creating the Report Procedure

If you have not defined a report procedure, you may do so by opening the **Application Tree** dialog and pressing the INSERT key, or choosing **Procedure ➤ New** from the menu. When the **Select Procedure Type** dialog opens, clear the **Use Procedure Wizard** box then select **Report**. This creates a report procedure and opens its **Procedure Properties** dialog. You should specify the report's files and keys as described immediately below.

Alternatively, you may use the Report Wizard to help you define your report procedure. See the *Wizards and Utility Templates* in the *Application Handbook*.

Specifying Files

The first logical step in preparing a report is to specify the files and keys your report procedure uses. You can do this from the **Procedure Properties** dialog, before even starting the **Report Formatter**.

The files you specify determine which data dictionary fields can appear on your report. You may specify more than one file to report on, that is, a primary file plus secondary related files and even other unrelated files. See *Dictionary Editor—Adding or Modifying Relationships*.

To specify the files for your report

1. From the **Application Tree** dialog, DOUBLE-CLICK on the report procedure.

   This opens the **Procedure Properties** dialog.

2. Press the **Files** button.

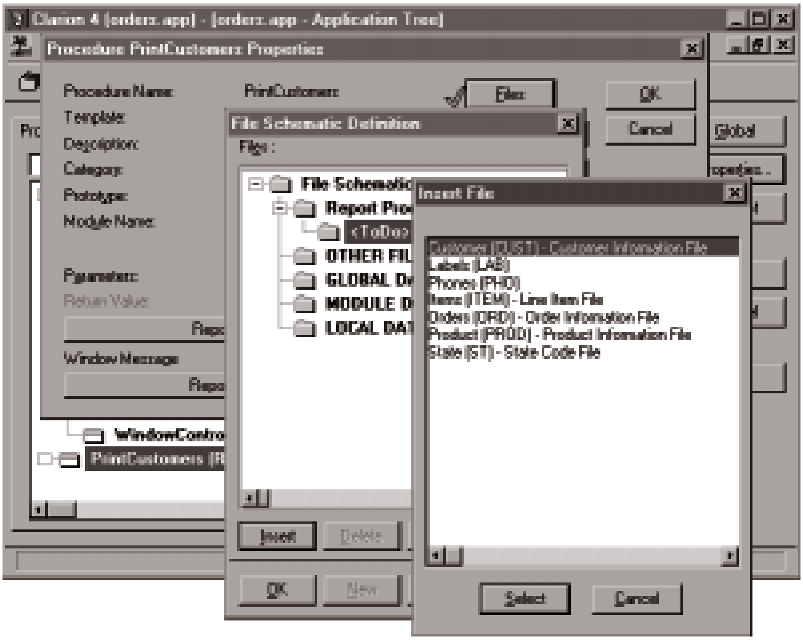
   This opens the **File Schematic Definition** dialog. Use this dialog to tell the Application Generator which files and keys your report procedure uses.

3. DOUBLE-CLICK the <ToDo> item for your report procedure.

   This opens the **Insert File** dialog.

4. DOUBLE-CLICK the file you wish to report from.

   This sets the file and closes the **Insert File** dialog. The <ToDo> item is replaced by the primary file you chose.



Clarion 4 (orders.app) - [orders.app - Application Tree]

Procedure PrintCustomers Properties

Procedure Name: PrintCustomers

File Schematic Definition

Insert File

Customer (CUST) - Customer Information File
Labels (LAB)
Phones (PHO)
Items (ITEM) - Line Item File
Orders (ORD) - Order Information File
Product (PROD) - Product Information File
State (ST) - State Code File

Printing From More Than One File (Adding Secondary Files)

1. From the **File Schematic Definition** dialog, DOUBLE-CLICK the primary file for your report procedure.

   This opens te **Insert File** dialog, listing only those files related to the primary file.

2. DOUBLE-CLICK the additional file you wish to report from.

   This sets the file and closes the **Insert File** dialog. Repeat these steps for any other related files. When adding files, you may also DOUBLE-CLICK a secondary file, then choose from files related only to the secondary file.

   All the fields from the primary file and the secondary files may be displayed on your report, and they are available for sorting and filtering as well.

Specifying Keys (Sort Order)

The keys (or indexes) you specify for your report procedure will determine the sequence in which the report items appear. The keys should also determine break variables and nesting order of any group breaks.

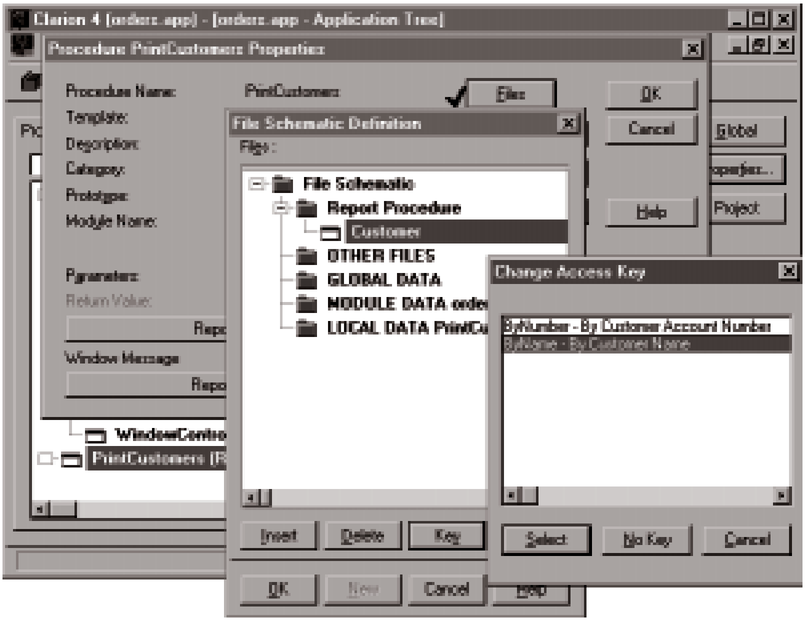
Adding secondary files (see above) to your procedure also gives you another logical field to break on—that is, the common fields linking the two files.

> **Tip:** **The ABC Report Template lets you specify other sort fields in addition to the key you set in the File Schematic Definition dialog. See *Procedure Templates—Report* in the *Application Handbook*.**

To specify the keys for your report

1. From the **File Schematic Definition** dialog, press the **Key** button, to specify which key is used for this procedure.

   This opens the **Change Access Key** dialog.



Clarion 4 (orders.app) - [orders.app - Application Tree]

Procedure PrintCustomers Properties

Procedure Name: PrintCustomers

File Schematic Definition

- File Schematic
  - Report Procedure
    - Customer
  - OTHER FILES
  - GLOBAL DATA
  - MODULE DATA orde
  - LOCAL DATA PrintCu

Change Access Key

ByNumber - By Customer Account Number
ByName - By Customer Name

2. DOUBLE-CLICK the key you want for this report.

   This sets the key and closes the **Change Access Key** dialog.

3. Press the **OK** button to close the **File Schematic Definition** dialog.

Specifying Which Records to Print (Range Limits & Filters)

You don’t always want to print all the records in a file. Often, you want to specify a subset of records to print. You can specify a subset of records to print by using Range Limits, Filters, or both.

You should use Range Limits whenever possible to specify your record selection, because Range Limits use file keys and are therefore very fast.

When Range Limits cannot provide a narrow enough selection criteria, you can add Filters to restrict your report to precisely the records you need.

To specify Range Limits and Filters, open the **Procedure Properties** dialog for your report, then press the **Report Properties** button. See *Procedure Templates—Report Template* in the *Application Handbook* for specific instructions on setting Range Limits and Filters.

Specifying Paper Size and Orientation

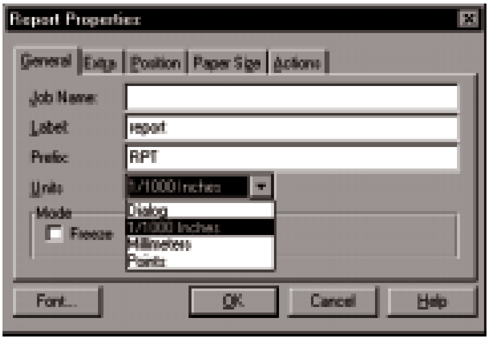
Paper size and text-to-paper orientation should be carefully thought out and established early in the report development process. Changing these settings after the report fields, headings, etc. have been laid out usually results in redoing much of the layout work.

To set the paper size and orientation, choose **Edit ➤ Report Properties**, then select the **Paper Size** tab. See *Report Formatter—Report Properties*.

Measurement Unit

To change the page measurement unit, choose **Edit ➤ Report Properties**, then select the **General** tab. Select dialog units, thousandths of inches, millimeters, or points from the **Units** drop-down list.

Dialog units (see the *Glossary*) are defined as one-quarter the average character width by one-eighth the average character height. The size of a dialog unit depends on the size of the default font for the report. This measurement is based on the font specified in the FONT attribute of the report or on the printer’s default font.



Report Properties

Job Name:

Label: report

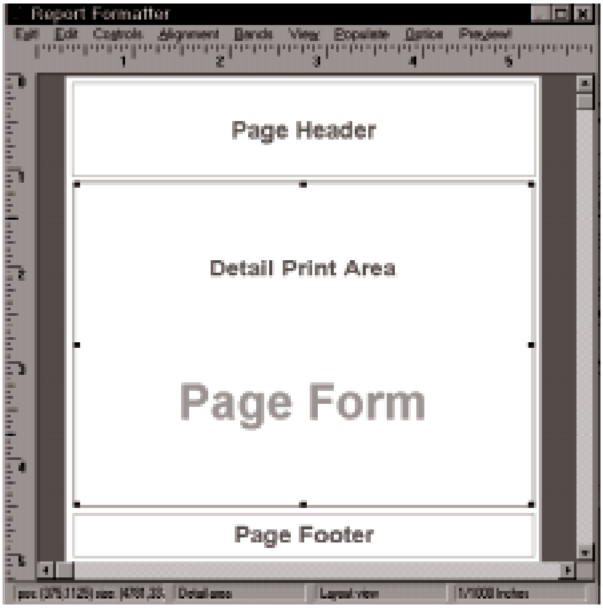
Prefix: RPT

Units: 1/1000 Inches

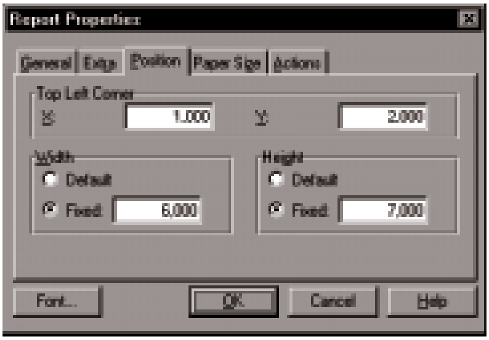
Specifying Report Margins

Margins for the various report sections (FORMs, page HEADERs, page FOOTERs, and the detail print area) are all set *independently* of each other, therefore, there is no one margin setting that applies to the entire report. You may specify these margins visually with the **Report Formatter's Page Layout View.** Please note that *the boundaries of each of these structures may overlap*.

To use the **Page Layout View** effectively, you should maximize the window so you can see as much of the report at one time as possible. Then use the TAB key to select the report section to reposition (the **Report Formatter** displays the selected section in the status bar at the bottom of the window). Finally, set the position of the selected section by clicking and dragging its interior, or position one or two edges of the selected section by clicking and dragging one of its handles.



You may also specify the margins of the FORM, page HEADER, page FOOTER, and detail print area on the **Position** tab of the respective **Form Properties, Page/Group Header Properties, Page/Group Footer Properties,** and **Report Properties** dialogs.



Top Left Corner
X: 1.000 Y: 2.000

Width
○ Default
● Fixed: 6.000

Height
○ Default
● Fixed: 7.000

The Detail Print Area

The detail print area is defined by the REPORT structure’s AT attribute. The four rules you should understand and remember about the detail print area are:

- Every DETAIL, group HEADER, and group FOOTER is printed within the boundaries of the detail print area.
- The boundaries of the detail print area may be set with the **Report Formatter’s Page Layout View** or with the **Position** tab of the **Report Properties** dialog.
- Each item within the detail print area is printed *relative to the previous item printed.*
- The *relative* position of items printed within the detail print area may be set with the **Position** tab of the item’s **Properties** dialog, or by dragging the item’s handles in Band View.

Positioning and Alignment

The AT Attribute

Use the **Position** tab to set the AT attribute for various report structures. The AT attribute on print structures performs two different functions, depending upon the structure on which it is placed.

When placed on a FORM, page HEADER, or page FOOTER, the AT attribute defines the position and size of the structure. The position specified by the *x* and *y* parameters is *relative to the top left corner of the page.*

When placed on a DETAIL, a BREAK, a group HEADER, or group FOOTER, the structure is printed relative to the previous item printed, according to the following rules (unless the ABSOLUTE attribute is present):

- The width and height parameters of the AT attribute specify the *minimum* size of the structure.
- The structure is actually printed at the *next available position within the detail print area*.
- The position specified by the x and y parameters of the structure's AT attribute is an *offset* from the next available print position within the detail print area.
- The first item is printed at the top left corner of the detail print area (at the offset specified by its AT attribute).
- Next and subsequent items are printed relative to the ending position of the previous item:

  If there is room to print the next item beside the previous item, it is printed there.

  If not, it is printed below the previous item.

Precise Positioning and Alignment

There are four major tools to help you precisely align your report data: Grid Snap, alignment tools, the **Position** tab of the respective properties dialogs, and constrained dragging. Grid Snap is discussed in *Report Formatter—Menus*. The alignment tools are discussed above in *Report Formatter—Toolboxes*.

Tip: **For super precise positioning, use the Position tab of the respective properties dialogs. When you position a structure on screen, the smallest unit you can move it is usually 1/96 inch. However, the Position tab lets you specify position by thousandths of inches.**

Constrained Dragging

Press and hold the SHIFT key while dragging a control to limit the control's movement to a single axis. That is, SHIFT + DRAG moves a control either horizontally or vertically, but not both.

Skipping Blank Lines

Use the SKIP attribute on a STRING control to suppress printing an empty string. That is, an empty STRING with SKIP attribute on a line by itself takes up no vertical space on the page.

For labels or other reports that may have an empty STRING on the same line as non-empty STRINGS, you can populate a series of local STRING variables, such as AddressField1 and AddressField2, on your report rather than the optional database fields. Then embed source code such as:

```
IF MBR:Address2                    !If 2nd address line not empty
  AddressField1 = MBR:Address2     !assign it to the report field
  AddressField2 = CLIP(MBR:City)&', '&MBR:State&' '&MBR:Zip
ELSE                               !If 2nd address line is empty
  AddressField1 = CLIP(MBR:City)&', '&MBR:State&' '&MBR:Zip
  AddressField2 = ''               !skip it
END
```

Alternatively, you can populate a TEXT control on your report whose USE variable is a STRING that is long enough to contain the entire series of optional and required database fields. Then embed source code such as:

```
TextField = CLIP(FirstName)&' '&MBR:LastName&'<13,10>'!name + return
TextField = CLIP(TextField)&MBR:Address1&'<13,10>'    !address + return
IF MBR:Address2                                       !if address2 exists
  TextField = CLIP(TextField)&MBR:Address2&'<13,10>'  !address2 + return
END
TextField = CLIP(TextField)&CLIP(MBR:City)&', '&MBR:State&' '&MBR:Zip
```

Handling Different Size Memos

Use the RESIZE attribute on a TEXT control to allow the control (and the report band) to expand to accommodate all of the text. That is, with the RESIZE attribute, you need not make the report band or the TEXT control as large as the largest data item to be printed. Make the controls as small as the smallest item and the report engine expands the controls for larger items.

Specifying a "Pre-printed" Form

In the **Report Formatter's Band View**, place text or graphic controls in the form band. This specifies constant text or graphics which prints on every page. This underlying "form" prints independently of and "underneath" all other items on the report.

See *Form* in the *Report Formatter* chapter.

Specifying Page Headers and Footers

In the **Report Formatter's Band View**, place text or graphic controls in the page header band or the page footer band. Use the **Bands** menu to add bands as required.

Page headers are composed *prior to* details and breaks. Page footers are composed *after* details and breaks. Both HEADERs and FOOTERs may be located anywhere on the page.

The data printed in the page headers and footers may be constant or variable. Page headers and footers are typically used to present report titles, fixed column headings, page numbers, print dates, company logos, etc.

See *Report Formatter—Page Header* and *Page Footer*. See also *Report Formatter—Using the Bands Menu.*

Specifying Column Headers and Report Titles

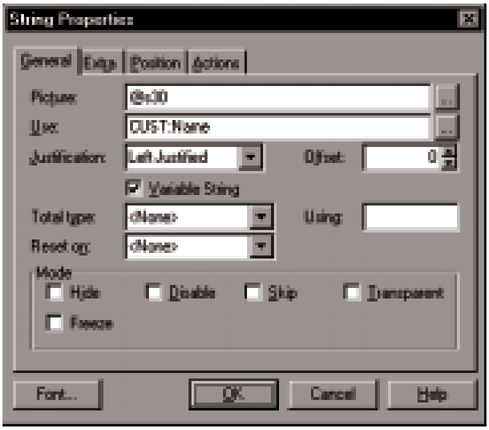
Static strings are good for printing column headings, report titles, and any other text that doesn't change. You will usually want to place your titles and headings in the page header or the page form. If you need to print titles or headings that change, see *Specifying Fields to Print*.

See *Report Formatter—Toolboxes.* See also *Controls and Their Properties—String Properties*.

To place a static string control in your report

1. Select the String tool from the Controls toolbox, or choose **Controls ➤ String**.
2. CLICK in the band that displays the title (usually the page header).

   The **Report Formatter** places a STRING control in the report structure. The center of the crosshair positions the upper left corner of the string.

   Picture: @s30
   Use: CUST:Name
   Justification: Left Justified
   Offset: 0
   ☑ Variable String
   Total type: <None>
   Using:
   Reset on: <None>
   □ Hide □ Disable □ Skip □ Transparent
   □ Freeze

3. DOUBLE-CLICK the string control, or RIGHT-CLICK the control then choose **Properties** from the popup menu.
4. Type the text to display in the **Text** field.

   No quotation marks are needed.
5. Press the **Font** button to specify the typeface, size, color, and style of the text.

Tip: Alternatively you may specify the text and it's font with the Property toolbox. Choose Option ➤ Show Propertybox.

6. Specify the text justification with the **Justification** drop-down list.

For special effects

1. Select the **Extra** tab to set text color and angle.
2. Press the **OK** button to close the **String Properties** dialog.

Specifying Fields to Print (variable text)

Variable string controls are the basic unit for printing variable data in the report. Variable strings are also used for displaying *totals and other calculated fields*.

By using the USE variable, the report procedure accesses the memory variable or data dictionary field you want to print. The **Report Formatter** formats the data according to the picture token you specify.

See *Report Formatter—Toolboxes* and *Menus.* See also *Controls and Their Properties—String Properties.*

To place a variable string control in your report

1. Select the Dictionary Fields tool from the controls toolbox, or highlight a field in the Populate Field toolbox (choose **Option ➤ Show Fieldbox**).

   Using the toolbox lets you place a data dictionary field only, without ever leaving the **Report Formatter**. Using the Dictionary Fields toolbox lets you select from data dictionary fields *and* memory variables using the **Select Fields** dialog.

2. CLICK in the band which will contain the variable string (usually the detail, group header or group footer).

   If you are using the Dictionary Fields tool, select your variable using the **Select Fields** dialog. The center of the crosshair positions the upper left corner of the string.

   The **Report Formatter** places a STRING control in the REPORT data structure with a variable as the string's **Use** attribute, and sets the following string properties: the **Variable String** box is checked, a **Picture** token and a **Justification** value is provided based on the data dictionary information for the selected field or variable.

Alternatively, you can place a string control, then set the string properties manually to accomplish the same result: check the **Variable String** box, type the variable name in the **Use** field, type a picture token in the **Picture** field, and select a **Justification** value from the drop-down list.

3. Press the **Font** button to specify the typeface, size, color, and style of the text.

Tip: Alternatively you may specify the text and it's font with the Property toolbox. Choose Option ➤ Show Propertybox.

For special effects

4. Select the **Extra** tab to set text color and angle.
5. Optionally, select a total type from the **Total Type** drop-down list to create sums, averages, tallies (counts), page numbers, etc. See *Creating Totals*.
6. Press the **OK** button to close the **String Properties** dialog.

Specifying Group Breaks

In order to have meaningful groups or subtotals in your report, the report records must be sorted properly. See *Specifying Sort Order*. Also see *Report Formatter—Group Breaks.*

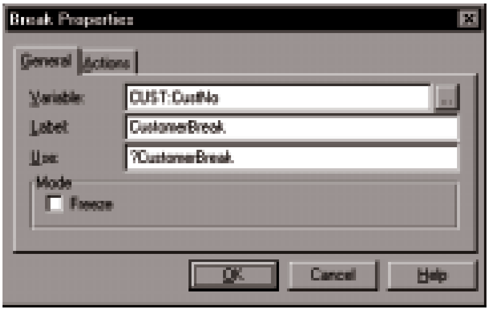
To create a group break

1. In the **Report Formatter**, choose **Bands ➤ Surrounding Break**.
2. Move the cursor over the detail band, then, when the cursor changes to a crosshair, CLICK in the DETAIL band.

   This opens the **Break Properties** dialog.
3. In the **Variable** field, press the ellipsis (...) button to select a break variable from the **Select Field** dialog.

   At runtime, when the variable's value changes, a new group begins.
4. In the **Label** field, type a valid Clarion label to use as a label for the BREAK structure.

   This label may be referenced by the RESET and TALLY attributes.



Variable: CUST:CustNo

Label: CustomerBreak

Use: ?CustomerBreak

Mode

□ Freeze

5. In the **Use** field, type a field equate label to reference the BREAK in your source code.
6. Press the **OK** button.

   This inserts the group BREAK. When the break variable changes, the report composes the group FOOTER and the next group HEADER defined for the break.

Specifying Group Headers and Footers

The print engine composes the group HEADER before the group DETAIL. The group HEADER is a good place to identify the group, for example, with a static string saying “Customer:” followed by a variable string displaying the customer name field.

The print engine composes the group FOOTER after the group DETAIL. The group FOOTER is a good place to summarize the group, for example, with a static string saying “Total:” followed by a variable string displaying a sum. See *Creating Totals*. See also *Report Formatter—Group Breaks—Group Headers* and *Group Footers.*

1. First, define a group break as described above.
2. Choose **Bands ➤ Group Header** from the menu to define a group HEADER for the BREAK.
3. Move the cursor over the break band; when the cursor changes to a crosshair, CLICK in the BREAK caption bar.

   This opens the **Page/Group Header Properties** dialog. Specify a field equate label and any special page breaking behavior. See *Specifying Page Breaks* below.
4. Press the **OK** button.

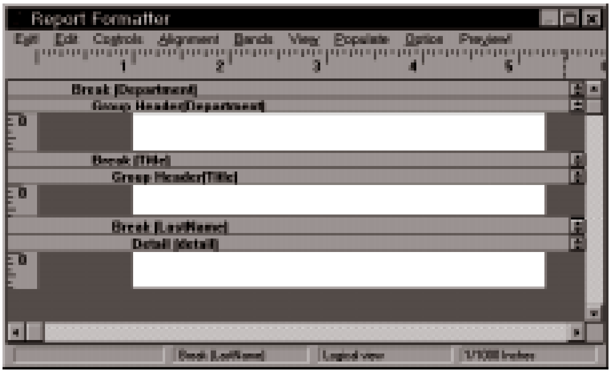
   This inserts the group HEADER band. You may place controls here just as in any other report band. Group footers are added similarly, using **Bands ➤ Group Footer** from the menu.

Specifying Nested Group Breaks

See *Specifying Sort Order* and *Specifying Group Breaks* above.

A nested break is created the same way as the first break. The second break may be nested “within” the first break by placing the second break on the detail inside the first break. Alternatively, the second break may be added “outside” the first break by placing the second break on the first break.

When establishing nested breaks, you should coordinate the nesting order of the breaks with the sort order of the files being processed. For example, if you have chosen a key composed of the Department, Title, and LastName fields, then you could similarly break on LastName, within Title, within Department.



You can review your key’s component fields in the Data Dictionary’s **Field/ Key Definitions** dialog.

Adding secondary files to your procedure also gives you another logical field to break on: that is, the common fields linking the two files.

Specifying Page Breaking Behavior

Edit the properties of the group HEADERs, the DETAILs, and/or the group FOOTERs. There are several options available. The options are Page Before, Page After, With Next, and With Prior.

See *Report Formatter—Group Header, Group Detail, Group Footer*.

Creating Totals and Calculated Fields

See *Specifying Fields to Print* above.

A total field is simply a variable STRING control with the SUM attribute added. The AVE, CNT, MAX and MIN attributes similarly create averages, counts (tallies), maximum, and minimum fields. These attributes may be added by choosing from the **Total type** drop-down list in the **String Properties** dialog.

In addition, you can precisely control the totalling behavior by specifying the RESET attribute from the **Reset on** drop-down list and by specifying the TALLY attribute from the **Tallies** list on the **Extra** tab. Finally, you can get ultimate control by specifying a variable to receive intermediate values for the SUM, AVE, CNT, MAX, and MIN operations with the **Using** field. You can perform custom calculations in embedded source using the intermediate variable to get any result you need.

> **Tip:** **You cannot automatically calculate intermediate group level totals with Clarion's Report Procedure templates and STRINGs. For example, you cannot add together two group level totals to create a third total.This type of calculation requires manual tracking of group breaks.**

In general, you place a total field in a page or group FOOTER so it can total the records from the beginning of the report, from the beginning of the page, or from the beginning of the BREAK group. However, you can also place a total field in a DETAIL structure to provide a running total. A tally (CNT) field in the DETAIL can number the records as they appear on the report.

To specify a total field

1. Place a variable string control as described in *Specifying Fields to Print*.

   For example, if you want total sales from an order entry system, select the data dictionary field containing the sales value.

2. Then DOUBLE-CLICK on the string control to open the **String Properties** dialog.

3. Choose a total type from the **Total type** drop-down list. Choose from *Sum, Average, Minimum, Maximum, Count***,** and *Page No***.**

4. Optionally, use the **Reset on** drop-down list to reset the total to zero before each page or before each break.

   For example, if your report is page-based so you need a total for each page, select *Page* from the **Reset on** drop-down list. If you want totals per customer and you have defined a break on customer, select the customer break from the **Reset on** drop-down list. If you want a grand total, select <*None*> from the **Reset on** drop-down list. See *Grand Totals* below.

5. Optionally, use the **Tallies** list on the **Extra** tab to specify when the calculation occurs.

By default, the specified calculation (SUM, CNT, MAX, MIN, AVG) occurs each time a report DETAIL is printed. This is true even if the calculated field is not in the DETAIL band and even if the calculated field is not in the lowest level child record of a multi-level file relationship.

In other words, use the **Tallies** list to limit the occurrence of the calculation to the printing of higher level breaks. For example, in a report that prints a three level file relationship (Customer>Order>Item), where the field to total resides in the middle level (Order) and the report prints all three levels, select the OrderBreak in the **Tallies** list to increment the total each time the Order BREAK prints rather than each time the Item DETAIL prints.

Tip: Don't choose the same break for both Reset on and Tallies. If you do, the result is always zero.

6. Optionally, use the **Using** field to specify a variable to receive intermediate values for the SUM, AVE, CNT, MAX, or MIN operation.

   In effect, this gives you access to the Report Engine's totalling capabilities. You can use the specified variable within your own custom calculations in embedded source code. Please note the Report Engine only writes to the intermediate variable, it does not read or reuse the variable for subsequent calculations. Therefore, you cannot alter a total generated by the Report Engine, however, you can HIDE the field calculated by the report Engine and replace it with your own custom calculated field.

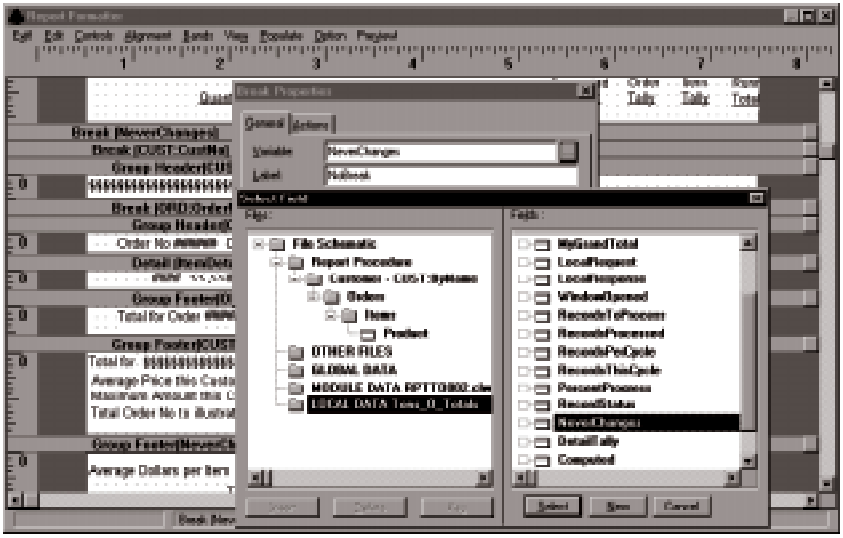
Sub-totals and Page Totals

Sub-totals are created simply by placing a total field within a page or group footer, and reseting the total to zero at the beginning of each page or group. Use the **Reset** drop-down list in the **String Properties** dialog to reset the total to zero before each new page or group.

Grand Totals

In effect, grand totals are simply sub-totals based on a break variable that never changes.

To create a grand total for your report, add a group break on a variable that doesn't change throughout the report. Any other group breaks should be nested within this grand total group break.



Next add a footer to the grand total group break. Finally, add the sub-total to the group footer as described above. Do not reset the total to zero, that is, select <*None*> from the **Reset on** drop-down list in the **String Properties** dialog.

Alternatively, you can create a separate DETAIL to calculate the grand total:

1. Add a new Detail band outside of any BREAK structures in the report.
2. Place your grand total fields on the new Detail band. Be sure to set Reset on to <None> and specify the detail or break structures on which to tally the total in the Tallies list on the Extra tab.
3. Suppress the Detail band from automatically printing:

   On the **Band Properties** dialog, name a field equate label as the detail structure's USE attribute.

   On the **Procedure Properties** for the Report, press the Report Properties button, then select the Detail Filters tab. Highlight the grand total Detail band then press the **Properties** button. Then, in the **Filter** field, type *False* to suppress automatic printing.
4. Embed the following code to explicitly print your grand total detail band:

   Clarion Templates	Before Print Preview
   ABC Templates	ReportManager Method Executable Code Section
   AskPreview()—Priority 1300

```
PRINT(RPT:MyGrandTotalDetailBand)
```

Row Totals

Displaying a row total (or any other calculation) requires two steps: assigning a value to a variable, then displaying the variable value in a string control (see *Specifying Fields to Print* above).

You can assign the value to the variable with embedded source code (see *Embedded Source Code* in the *Application Generator* chapter) or with the Formula Editor (see the *Formula Editor* chapter). There is a good example of this process in the *Creating Reports—An Invoice Report* section of the *Learning Clarion* book.

Whether you use the Formula Editor or embedded source, the key point to know is that the assignment should be done just *prior* to the PRINT statement. This table shows which embed point to use for each alternative:

Formula	Before Print Detail
Clarion Templates	Before Printing Detail Section
ABC Templates	Process Manager Method Executable Code Section TakeRecord(),BYTE

Page Numbers

Controlling/Resetting Page Numbers

See *Report Structures and Properties—Group Header, Group Detail, Group Footer*.

Edit the properties of the group HEADERs, the DETAILs, and/or the group FOOTERs. There are several options available. The options are Page Before, Page After, With Next, With Prior, and New Page No.

Displaying Page Numbers

Use the ReportPageNumber template (see *Control Templates* in the *Application Handbook*) or create your own page number control as follows:

1. Choose **Controls ➤ String** or select the string tool from the Controls toolbox, then CLICK in the page header or footer band.
2. DOUBLE-CLICK on the string control you just placed.

   This opens the **String Properties** dialog.
3. Check the **Variable String** box.
4. In the **Picture** field of the **String Properties** dialog, type @pPage <<#p.

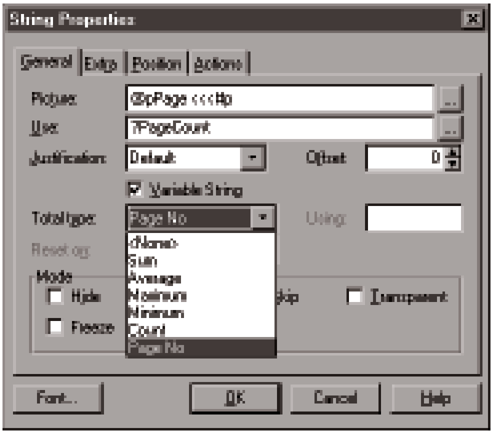
   This picture token suppresses leading zeroes and displays the page number following the word “Page.”

5. In the **Use** field, type a field equate label to reference the string in source code.

   For example, type *?PageCount*. This need not reference a variable. The Report Engine generates its own variable to hold the page number.

6. From the **Total type** drop-down list, choose **Page No**.

7. Press the **OK** button to close the **String Properties** dialog.



Picture: @pPage <<<#p

Use: ?PageCount

Justification: Default

Offset: 0

☑ Variable String

Total type: Page No

Using:

□ Hide

□ Freeze

□ Transparent

Displaying Print Dates

To place a "Date Printed" in a report

Use the ReportDateStamp template (see *Control Templates* in the *Application Handbook*) or create your own date stamp control as follows:

1. Choose **Controls ➤ String**, then CLICK in the appropriate report band.

   For a print date you generally use the page header or page footer.

2. DOUBLE-CLICK on the string control you just placed.

   This opens the **String Properties** dialog.

3. In the **Text** field, type a date picture (@d1, for example), or press the ellipsis button to use the Picture Editor

4. In the **Use** field, press the ellipsis (...) button to create a local variable called *PrintDate*.

   This opens the **Select Field** dialog.

5. Highlight Local Data, then press the **New** button.

   This opens the **New Field Properties** dialog.

6. In the **Name** field, type *PrintDate*.

7. Choose **Long** from the **Data Type** drop-down list.

8. Press **OK** to close the **New Field Properties** dialog.
9. Press **OK** to close the **String Properties** dialog.
10. Press **Exit!** to exit the **Report Formatter**.

To assign a value of TODAY() to PrintDate

1. CLICK the **Embeds** button in the **Procedure Properties** dialog.
2. In the **Embedded Source** dialog, DOUBLE-CLICK the following embed point:

Clarion Templates	Procedure Setup
ABC Templates	WindowManager Method Executable Code Section Init()

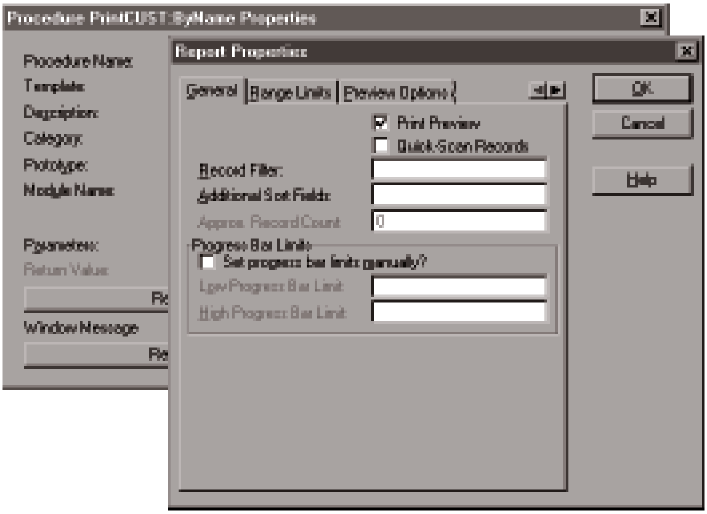
3. In the **Select embed type** dialog, DOUBLE-CLICK **SOURCE**.
4. In the Text Editor type:

```
PrintDate = TODAY()
```

5. Press **Exit!** and save your embedded source.

Implementing Print Preview

Generate a report procedure with the Report template or the Report Wizard (see *Wizards and Utility Templates* and *Procedure Templates* in the *Application Handbook*). In the report procedure's **Procedure Properties** dialog, press the **Report Properties** button, then check the **Print Preview** box in the **Report Properties** dialog (see *Report Properties* above).



☑ Print Preview

☐ Quick-Scan Records

Approx. Record Count: 0

☐ Set progress bar limits manually?

If you prefer to hand code your print preview process, see PREVIEW in the *Language Reference*, for more information and examples.

Printing Labels (Dynamically)

Printing labels simply means printing a multi-column report so that the report rows and columns match up with the commercial label forms you use.

In the real world, different users will have different label papers at different times, so ideally, the user should be able to specify the dimensions of the label paper at run-time, and the label report should fit the specified label paper.

❑ *To print dynamically sized labels:*

◆ Add a Labels file to your data dictionary.

◆ Create Browse and Update procedures for the Labels file.

◆ Create a Label Report procedure.

◆ Add the Label file to the Label Report File Schematic.

◆ Embed code to do the run-time resizing.

Add a Labels File to Your Data Dictionary

The file may use whichever file system you prefer, and should contain the following fields. No keys are required, but using LabelType as a unique key provides alphabetic sorting and prevents duplicate entries. See *Dictionary Editor—Adding Files to the Dictionary* and *Adding or Modifying Fields*.

Name	Type	Length	Initial Value
LabelType	String	15	
PageWidth	Decimal	5,2	
PageHeight	Decimal	5,2	
LabelWidth	Decimal	5,2	
LabelHeight	Decimal	5,2	
TopMargin	Decimal	5,2	
LeftMargin	Decimal	5,2	
FontSize	Byte		11

Create a Browse and an Update Form for the Label File

The fastest way to build these procedures is to use the procedure Wizard. See *Wizards and Utility Templates* in the *Application Handbook.*

1. From the Application Tree, press the INSERT key.
2. In the **New Procedure** dialog, type a name (*BrowseLabels*) then press **OK**.
3. In the **Select Procedure Type** dialog, check the **Use Procedure Wizard** box, then select **Browse**

4. Answer the Wizard's questions:

   Choose the Labels file. *Do* allow the user to update records. *Do* provide a "Select" button. The Wizard builds both procedures, then opens the **Procedure Properties** dialog for the Browse procedure.

5. Press the **OK** button to exit the **Procedure Properties** dialog.

Create a Label Report Procedure

Your report should observe these conventions. The report fields should have Height and Width of **Default** and there should be only one field per line. Thus if you need to display FirstName and LastName on the same line, you should concatenate these fields into a single field.

1. Create a report for your address file.

   Use the Report Wizard if you want, but don't worry about formatting yet. Just make sure the report contains all the name and address fields you need for your labels.

   Call the label report as you would any other report. See *Application Generator—Adding a Procedure to Your Application*, and *Window Formatter—Creating Your Application's Menu*.

2. From the Application Tree, RIGHT-CLICK your report procedure and choose **Report** from the popup menu.

3. Delete all report sections, except the Detail section.

   CLICK on the section's caption bar then press DELETE.

4. For report lines with more than one field, concatenate the fields and remove trailing spaces.

   This step improves the appearance of your labels by removing unnecessary spaces. This step is also required to support the embedded code recommended below.

   Follow the steps described in the *How to CLIP and Concatenate Name Fields* of the How Do I...? topic in the on-line help.

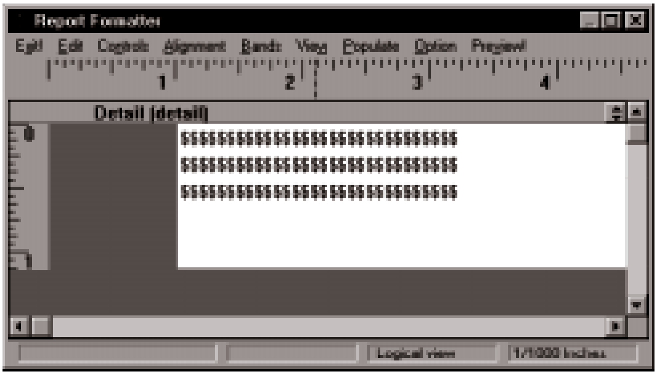
5. Set the properties for each report field.

Tip: Use the Property Toolbox to identify the fields: Choose Option ➤ Propertybox.

   Arrange your name and address fields in a vertical format, that is, one field below another, starting at the top left corner of the Detail band. Use the Alignment tools for precise alignment. See *Specifying Fields to Print* above.

   Remember to set the Height and Width of each field to **Default** on the **Position** tab.

   Your report should look similar to the illustration.



6. Specify a field equate label (USE attribute) for the Detail section.

   CLICK on the detail then type *?detail* in thePropertybox **Use** field.

7. **Exit!** the **Report Formatter** and save your changes.

Add the Label File to the Report File Schematic

This step ensures that the report procedure opens the Label file and therefore has access to the Label file's record buffer. The embedded code below uses the position and size values in this record buffer to assign run-time positions and sizes to the label report.

1. From the Application Tree, RIGHT-CLICK your label report procedure, and choose **Properties** from the popup menu.
2. From the **Procedure Properties** dialog, press the **Files** button.
3. From the **File Schematic** dialog, DOUBLE-CLICK **Other Files**.
4. From the **Insert File** dialog, DOUBLE-CLICK **Labels**.
5. Press **OK** to exit the dialogs.

Embed Run-time Resizing Code

Now we will embed code at three points in the report procedure:

- Beginning of Procedure—call the BrowseLabels procedure to allow the user to specify the label paper to use.
- After Opening Report—call the ResizeTheReport routine to assign the label sizes specified by the user.
- Procedure Routines—the ResizeTheReport routine.

1. From the Application Tree, RIGHT-CLICK your label report procedure then choose **Embeds** from the popup menu.
2. From the **Embedded Source** dialog, DOUBLE-CLICK the following embed point:

Clarion Templates	Beginning of Procedure, After Opening Files
ABC Templates	WindowManager Method Executable Code Section Init()—Priority 7800

3. From the **Select Embed Type** dialog, DOUBLE-CLICK SOURCE and type the following statements:

```
  !Allow Run-time Specification of Label Paper
  GlobalRequest = SelectRecord    !enable select button
  BrowseLabels                    !call BrowseLabels procedure
```

4. Exit the Text Editor and save your changes.
5. From the **Embedded Source** dialog, DOUBLE-CLICK the following embed point:

Clarion Templates	After Opening Report
ABC Templates	WindowManager Method Executable Code Section Open()—Priority 7800

6. From the **Select Embed Type** dialog, DOUBLE-CLICK SOURCE and type the following statements.

```
  DO ResizeTheReport
```

7. Exit the Text Editor and save your changes.
8. From the **Embedded Source** dialog, DOUBLE-CLICK the *Procedure Routines* embed point (ABC and Clarion Templates).
9. From the **Select Embed Type** dialog, DOUBLE-CLICK SOURCE and type the following statements.

   If you specified a label for your REPORT structure, change the report labels below to match your label.

```
ResizeTheReport   ROUTINE
!!==============================================================
!! Modify the report to fit specified label paper.
!! Assumes Report Measurement Unit is 1/1000 of an inch.
!!==============================================================

  SETTARGET(report)  !Make report the target for property assignments

  !!==========================================================
  !! Define the REPORT's AT attribute (ie detail print area).
  !! The detail print area should be the size of the entire page.
  report{PROP:Width}  = LAB:PageWidth  * 1000
  report{PROP:Height} = LAB:PageHeight * 1000
  !! Adjust the margins to center the address text within each label.
  report{PROP:Xpos}   = LAB:LeftMargin * 1000
  report{PROP:Ypos}   = LAB:TopMargin  * 1000

  !!==========================================================
  !! Define the DETAIL's AT attribute.
  !! The DETAIL should be the size of each individual label.
  ?detail{PROP:Width}  = LAB:LabelWidth  * 1000
  ?detail{PROP:Height} = LAB:LabelHeight * 1000
```

```
!!===========================================================
!! Adjust font size and vertical position of the report fields.
LOOP Fld#=FIRSTFIELD() TO LASTFIELD()       !Process all report fields
  Fld#{PROP:Fontsize} = LAB:FontSize        !Set font size
  IF Fld# > FIRSTFIELD()                    !Skip the first field,
    Fld#{PROP:Ypos}=BottomOfPreviousField#  ! reposition the rest
  END
  BottomOfPreviousField#=Fld#{PROP:Ypos}+(16*LAB:FontSize)
END
SETTARGET()               !Reset default target to the active window.
                EXIT      !Exit the ROUTINE.
```

That's it! Make and run your application. When you run your label report, you will have the opportunity to input the dimensions of your various label papers. The dimensions are stored in the Label file where they are available for your selection or modification each time you print labels.

Printing One Record per Page

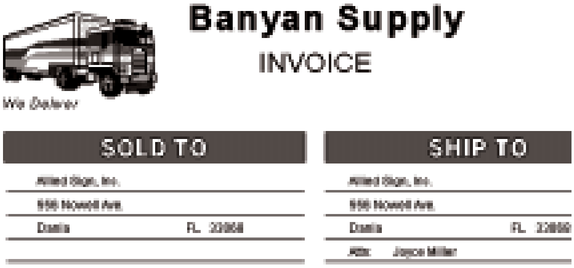
To print a separate page for each record, check the PAGEAFTER box in the **Detail Properties** dialog. See *Report Structures and Properties—Detail*.

Printing Mail-Merge Documents

For a mailmerge document, you usually place the name and address fields in the HEADER, then reserve the DETAIL for a multi-line text control. See *Page Header* above. See *Printing Multi-Line Text* below. The ClubMgr example application also contains a simple mailmerge.

Printing Graphics

Graphic controls embellish the report and guide the reader's eye to the data. The following controls allow you to add pictures, lines, and shapes to your report. See the *Controls and Their Properties* chapter for more information on the following controls.



Banyan Supply

INVOICE

We Deliver

SOLD TO	SHIP TO
Allied Sign, Inc.	Allied Sign, Inc.
556 Howell Ave.	556 Howell Ave.
Dania FL 33068	Dania FL 33069
	Attn: Joyce Miller

Image

Most likely you will wish to place an image, such as a logo, in a HEADER. You may choose any of the graphic file formats supported for window controls; however, printing large images, especially .JPG files may present problems for some printers.

The most important consideration when placing an image is its size—Clarion automatically resizes the image to fit the control size. This may introduce distortion. WMF files distort the least, however, the simplest way to prevent distortion is to preserve the height to width ratio of your images.

To size a 640 x 480 pixel graphic, for example, determine its height-to-width ratio, which is 4:3. Plan an image box in the same ratio—for example, 2000 x 1500 thousandths, which represents a 2 inch by 1.5 inch box on the page.

Tip: Whenever possible, use vectorized graphics such as the Windows Metafile Format (*.WMF). When you need to shrink or stretch them, their appearance is less subject to distortion than a bitmap.

To place an IMAGE control in your report:

1. Select the Image control from the Controls toolbox, or choose **Controls ➤ Image**.
2. CLICK in the band where you want to place the control.

   The **Report Formatter** places an IMAGE control in the report structure. The center of the crosshair positions the upper left corner of the control.
3. DOUBLE-CLICK the control you just placed.

   This opens the **Image Properties** dialog.

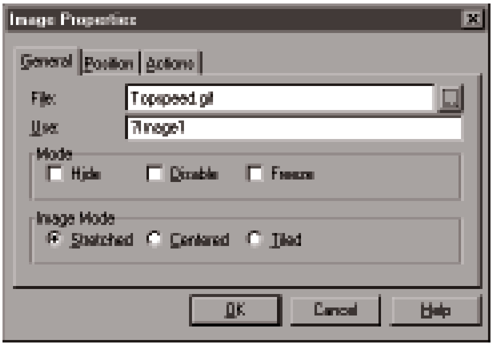


Image Properties

File: Topspeed.gif
Use: ?Image1
Mode: ☐ Hide ☐ Disable ☐ Freeze
Image Mode: ● Stretched ○ Centered ○ Tiled

4. In the **File** field, type the fully qualified image file name, or press the ellipsis (**...**) button to select the file from theWindows file dialog.

Clarion automatically links the image file into the executable when the file is explicitly named in the control, so you don't have to ship the image file separately.

5. In the **Use** field, type a field equate label to refer to the image control in source code.
6. Select the **Position** tab.
7. Type the correct image size in the fixed **Width** and **Height** fields.

   You may also resize the image by CLICKING and DRAGGING its handles.
8. Press **OK**.

Line

Lines are the simplest means of visually separating sections or fields within your report. To place a line:

1. Select the Line tool from the Controls toolbox, or choose **Controls ➤ Line**—the cursor changes to a crosshair.
2. CLICK in the band in which you want to place the line.

   The center of the crosshair positions the left end point. The **Report Formatter** places a LINE control in the report structure. Relocate and resize the line by dragging its handles.
3. DOUBLE-CLICK the line you just placed.

   This opens the **Line Properties** dialog.
4. In the **Use** field, type a field equate label to refer to the line control in source code.
5. Select the **Position** tab to specify exact coordinates for the line.

   To specify a horizontal line, be sure to check the **Fixed** box in the **Height** group, and type a zero (0) in the box next to it. The height is the measure of the vertical distance between the origin and the end point; for a horizontal line, this is equal to zero. In the **Width** group, type the length of the line in the **Fixed** box.

   To specify a vertical line, check the **Fixed** box in the **Width** group, and type a zero (0) in the box next to it. The width is the measure of the horizontal distance between the origin and the end point; for a vertical line, this is equal to zero. In the **Height** group, type the length of the line in the **Fixed** box.

   To specify a horizontal or vertical line the full width or height of the section, check the **Full** option

Tip: To create a horizontal divider line useful for splitting a HEADER from the DETAIL section, for example, check Full Width, and set the Fixed Height to zero.

6. To specify a line color, press the **Line Color** button on the **Extra** tab, then choose a color from the **Line Color** dialog.

Box

You can highlight a report field by placing a gray box underneath it. You can frame an entire area of a report by placing a box with no fill around it.

> **Tip:** **When overlapping one control over another, choose Edit ➤ Set Control Order to ensure the underlying control is printed before the overlying control; otherwise the overlying control may be obscured.**

To place a box:

1. Select the Box tool from the Controls toolbox, or choose **Controls ➤ Box**.
2. CLICK in the band in which you want to place the box.

   The center of the crosshair positions the upper left corner of the box. When you click, the **Report Formatter** places a BOX control in the report structure. Resize and relocate the box by dragging its handles or its interior.
3. DOUBLE-CLICK the box you just placed.

   This opens the **Box Properties** dialog.
4. In the **Use** field, type a field equate label to refer to the control in source code.
5. Select the **Extra** tab.
6. Set the COLOR attributes.

   If you want a solid box filled with color, type a valid color equate in the check **Fill Color** box or press the ellipsis (...) button to choose a color from the **Fill Color** dialog.

   If you want a colored border, type a valid color equate in the check **Border Color** box or press the ellipsis (...) button to choose a color from the **Border Color** dialog.
7. If you want rounded corners for the box, check the **Round** box to set the ROUND attribute.
8. To set the size of the box by typing in coordinates, select the **Position** tab.

   Type the measurements you wish in the **Fixed Width** and **Fixed Height** boxes.

> **Tip:** **To create a border or 'frame' around the whole report, place a box in the Form band. Be sure the FORM is the full size of the page. Create a box with a border but no fill, and set the width and height to Full.**

9. Press the **OK** button to close the **Box Properties** dialog.

Ellipse

When placing an ELLIPSE in a report, follow the same procedures as for placing a box.

Printing Multi-line Text with Word-wrap

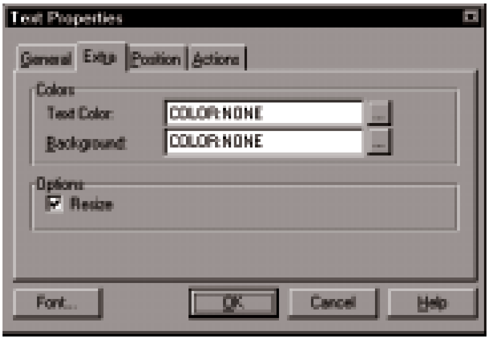
A multi-line text control can print a long string (such as a MEMO), automatically word-wrapping and printing as many lines as the MEMO's contents requires.

1. Select the Text tool from the Controls toolbox, or choose **Controls ➤ Text Field**.
2. CLICK in the band which will hold the control.

   The center of the crosshair positions the upper left corner of the control. The **Select Field** dialog appears. Use this dialog to select (or create) the data dictionary field or memory variable to print.
3. Press the **Select** button.

   The **Report Formatter** places a TEXT control in the report structure. Resize the text box by using the mouse to drag its handles.
4. RIGHT-CLICK the text control and choose **Properties** from the popup menu.
5. Select the **Extra** tab, then check the **Resize** box.

   At run-time, the text expands downward, expanding the detail if necessary, to contain the entire text of the memo! If necessary, the text flows onto the following page. The RESIZE attribute only adjusts the height of the TEXT and the DETAIL.



Text Color: COLOR:NONE
Background: COLOR:NONE
☑ Resize

Reports That Look Like Windows

The **Report Formatter** gives you the ability to print on the page virtually anything you can put on the screen. This includes specialized controls such as list boxes, check boxes, group boxes, radio buttons, etc.

In most cases, setting control properties for a report is identical to setting control properties for a window. See the *Controls and Their Properties* chapter for more information on each of the following controls.

Option Box

You may print an OPTION structure within your report. This appears on the page exactly as it does on screen—as an option box. You place an option structure on the page only to hold radio buttons. You may hide the box structure so that only the buttons print on the page.

1. Select the Option Box tool from the Controls toolbox, or choose **Controls ➤ Option Box**.
2. CLICK in the band in which you want to place the OPTION structure.

   The center of the crosshair positions the upper left corner of the box. The **Report Formatter** places an OPTION structure within the report structure. Resize and relocate the option box by dragging its handles or its interior.
3. DOUBLE-CLICK the option box you just placed.

   This opens the **Option Properties** dialog.
4. In the **Text** field, type a caption for the option box.

   If you choose not to hide the option box when printing, the caption appears at the upper left border of the box, just as it does on screen.
5. In the **Use** field, type a field equate label to refer to the control in source code.
6. Select the **Extra** tab.
7. Clear the **Boxed** box to hide the box, but not the radio buttons.
8. Press **OK**.

   You must add each radio button separately, placing them in the OPTION box.

Radio Button

Placing RADIO buttons in a printed report provides a visual aid to the user by showing the selected value as well as all the possible values for the field.

Before you place the radio buttons in the report, you must first place an OPTION structure, by using the **Controls ➤ Option Box** command. The

RADIO button must be placed inside the option box representing the OPTION structure. If you attempt to place a radio button without an OPTION structure, the Development Environment displays an error message.

1. Place an option box.
2. Select the Radio Button tool from the Controls toolbox, or choose **Controls ➤ Radio Button**.
3. CLICK inside the option box you just placed.

   The center of the crosshair positions the upper left corner of the radio button. The **Report Formatter** places an RADIO control within the OPTION structure.
4. DOUBLE-CLICK the radio button you just placed.

   This opens the **Radio Button Properties** dialog.
5. In the **Text** field, type a caption for the radio button.

   The caption appears beside the radio button, just as it does on screen.
6. In the **Use** field, type a field equate label to refer to the control in source code.

   The radio button automatically turns 'on' or 'off' according to the value of the variable specified in the OPTION box's USE attribute.
7. Press **OK**.

Check Box

The check box (CHECK control) provides an attractive way to display a yes/no choice for a field—the alternative might be an entire column that repeats "one," "yes," or even ".T." for each record.

The printed check box looks similar to an on screen check box. To place the check box:

1. Select the Check Box tool from the Controls palette, or choose **Controls ➤ Check Box**.
2. CLICK inside the band in which you want to place the check box.

   The center of the crosshair positions the upper left corner of the check box. The **Select Field** dialog appears. Use this dialog to select (or create) the data dictionary field or memory variable displayed by this control. This should be a numeric variable which turns the check box on or off. A value of zero indicates the box is unchecked; any other value, checked.
3. Press the **Select** button.

   The **Report Formatter** places a CHECK structure within the report structure.

4. DOUBLE-CLICK the control, or RIGHT-CLICK the control and choose **Properties** from the popup menu.

   This opens the **Check Box Properties** dialog.

5. In the **Text** field, type a caption for the check box.

   The caption appears beside the check box, just as it does on screen.

6. Press **OK**.

Group Box

The primary reason for placing a group box in a report is to make a group of controls on paper resemble their appearance on screen.

To place the GROUP control:

1. Select the Group Box tool from the Controls toolbox, or choose **Controls ➤ Group Box**.

2. CLICK inside the band in which you want to place the Group Box.

   The center of the crosshair positions the upper left corner of the group box. The **Report Formatter** places a GROUP structure within the report structure.

3. DOUBLE-CLICK the control, or RIGHT-CLICK the control and choose **Properties** from the popup menu.

   This opens the **Group Properties** dialog.

4. In the **Text** field, type a caption for the group box.

   This appears at the upper left border of the group box when the report prints, provided you check the **Boxed** box.

5. In the **Use** field, type a field equate label to refer to the control in source code.

6. Select the **Extra** tab.

7. Clear the **Boxed** box to hide the box, but not the internal controls.

8. Press **OK**.

9. Add additional controls to the group.

List Box

When the data you require for the report exists in a QUEUE, you may place a listbox in the report. The list box that appears on the page is similar to the LIST control that appears on screen, though it will obviously not have the same functionality—the printed page does not support scroll bars, for example.

Because a queue is required for a listbox, and because Clarion’s report template does not normally use a queue, several embedded source statements must be added to a template generated report procedure in order to use a listbox. Alternatively, you may hand code the report procedure or use a report template that uses a queue.

At print time, the first time the code cycles through the report, it prints the LIST's header and the first item in the queue. Each additional cycle prints the next item of the queue without repeating the header. The LIST's header is automatically repeated for applicable page breaks and group breaks.

To place a listbox in your template generated report:

1. Select the List Box tool from the Controls toolbox, or choose **Controls ➤ List Box**.
2. CLICK in the band which should contain the control.

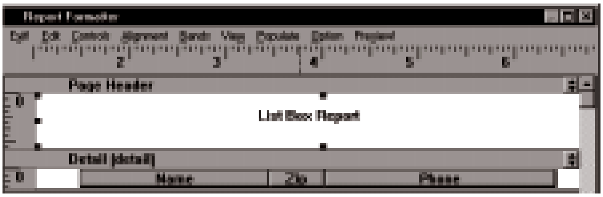
   The center of the crosshair positions the upper left corner. When you click, the **List Box Formatter** appears. Use the **List Box Formatter** just as though you were designing a list box for the screen. See the *Window Formatter—List Box Formatter*. When you are finished, press the **OK** button.

> **Tip: Each listbox item prints as a "selected" item with the specified selected item colors. The default Windows colors for a selected item are "reverse video" (white-on-black). To produce normal black-on-white printing, use the List Box Formatter to set the Selected Text for each column to COLOR:Black and the Selected Background to COLOR:White.**

3. Make the list box one (1) row high (about .220 inches).

   Drag the bottom handle until only the list box headings are visible. This eliminates empty space between rows.
4. Make the detail band the same height as the list box.

   Drag the list box to the top of the band or set its Y coordinate to zero (0), then drag the bottom handle of the band to the bottom of the list box. This eliminates empty space between rows. Your report should look something like this illustration.

5. DOUBLE-CLICK the list box to open the **List Properties** dialog.
6. In the **Use** field, type a field equate label to refer to the control in source code.

7. In the **From** field, type the label of the queue the listbox displays, or press the ellipsis (...) button to select or define the queue.
8. Press the **OK** button to close the **List Properties** dialog.

Embeds Required:

Generally speaking, the embedded source statements need to do two things: declare the QUEUE and load the queue buffer with data.

1. Declare a QUEUE in the following embed point:

Clarion Templates	Data Section, Before Report Declaration
ABC Templates	Data Section, Before Report Declaration

```
CustomerView    VIEW(Customer)      !Template generated view
                  PROJECT(CUST:Name)
                  PROJECT(CUST:Phone)
                  PROJECT(CUST:Zip)
                END

CusQ    QUEUE,PRE                   !embedded QUEUE declaration
Name      LIKE(CUST:Name)
State     LIKE(CUST:Phone)
ZIP       LIKE(CUST:Zip)
        END
```

2. Load the queue in the following embed point:

Clarion Templates	Before Printing Detail Section
ABC Templates	ProcessManager Method Executable Code Section Next(BYTE ProcessRecords=True),BYTE Priority 7500

```
CusQ.Name   = CUST:Name             !copy VIEW buffer to QUEUE buffer
CusQ.Phone  = CUST:Phone
CusQ.Zip    = CUST:Zip
```

Custom Controls

You may place a .VBX CUSTOM control in your report. There are a number of custom control libraries available which are very suitable for reports—including graphs and other visual elements. To place the control:

1. Select the VBX tool from the Controls toolbox, or choose **Controls ➤ Custom Control**.
2. CLICK inside the band which will hold the control.

   The center of the crosshair positions the upper left corner of the custom control. When you CLICK, the **Select Custom Control** dialog appears. Use this dialog to select a custom control.
3. Press the **OK** button.

   The **Report Formatter** places a CUSTOM control within the report structure. Resize and relocate the custom control by dragging its handles or its interior.

4. DOUBLE-CLICK the control you just placed.

   This opens the **Custom Control Properties** dialog.

5. In the **Text** field, type a caption for the control.

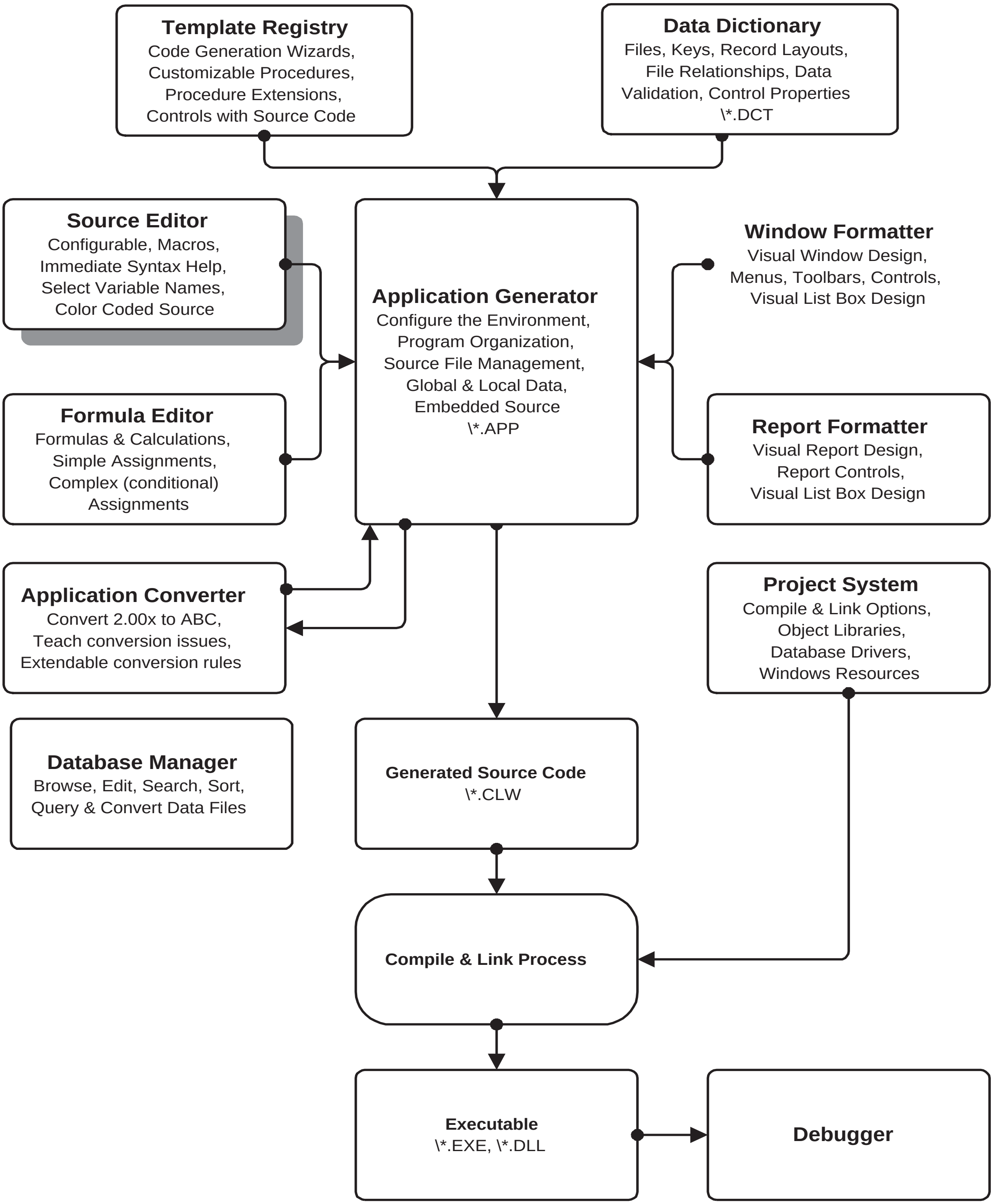
   The .VBX control may or may not display its title on the page, depending on the .VBX you use.

6. In the **Use** field, type the name of a variable.

   The variable type depends on the .VBX control. The variable value is passed to the .VBX control. Please see the *Controls and Their Properties* chapter for more details.

7. Optionally, check the **Meta** box to print the control as a metafile (.WMF).

9 - TEXT EDITOR



About This Chapter

This chapter introduces the Text Editor. If you allow the Application Generator to write most of your source code, you will probably only use the Text Editor to write your embedded source code (see *Application Generator—Embedded Source Code*). If you write your source code “from scratch,” you will probably use the Text Editor extensively to create and manage your code. The Text Editor features the following to help you accomplish either purpose:

- Multiple Document Windows, in which you may edit as many documents as your system allows.
- Color coded syntax highlighting, which makes reading individual code lines easier. The color coding is fully customizable.
- Always available Search and Replace.
- Auto-indent, to make reading and writing code easier.
- Next Error and Previous Error locator.
- Current cursor position (row and column), displays on the status bar.
- Configuration options to customize the Text Editor to fit your needs.

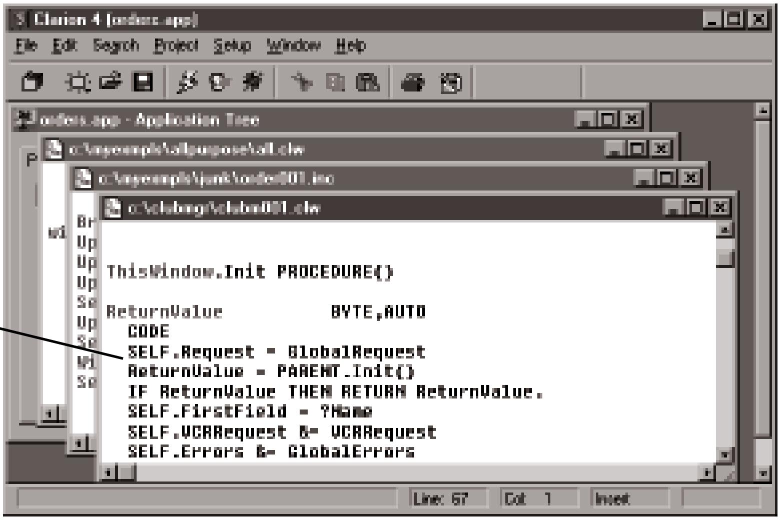
Opening the Text Editor

Anytime you view a source code document with Clarion, you use the Text Editor. Here are several ways to open a source code document:

- ❑ Choose **File ➤ New ➤ Source** tab to open the **New** dialog. Navigate to your source directory and fill in the name of your new file in this standard dialog. Then press the **Save** button. This opens a blank source code document.
- ❑ Choose **File ➤ Open**, select the **Files of type** Clarion source, then DOUBLE-CLICK a source code file in the standard **Open File** dialog.
- ❑ Use the **File ➤ Pick** command to view your most recently edited files. Select the **Source** tab, highlight a source code file, then press the **Select** button.
- ❑ In the **Project Editor** dialog, highlight a source code (.CLW) file, then press the **Edit** button. The **Edit** button is only enabled for hand coded projects.
- ❑ After a compile that generates errors, press the **Edit Errors** button.
- ❑ In the **Application Tree** dialog, RIGHT-CLICK a procedure, then choose **Source** to open the Embeditor (Text Editor in a special embedded source mode).
- ❑ In the **Application Tree** dialog, RIGHT-CLICK a procedure, then choose **Module** to edit a generated source module.

Managing Text Editor Windows

Each source code file appears in a separate document window. This section provides a summary of actions you can take to change the layout of these windows:


Clarion 4 (orders.app)
File Edit Search Project Setup Window Help
orders.app - Application Tree
c:\myexmpls\allpurpose\all.clw
c:\myexmpls\junk\order001.inc
c:\clubmg\clubm001.clw
Line: 67
Col: 1
Insert

```
ThisWindow.Init PROCEDURE()

ReturnValue          BYTE,AUTO
  CODE
  SELF.Request = GlobalRequest
  ReturnValue = PARENT.Init()
  IF ReturnValue THEN RETURN ReturnValue.
  SELF.FirstField = ?Name
  SELF.VCRRequest &= VCRRequest
  SELF.Errors &= GlobalErrors
```

The Editor's document window features color coded syntax highlighting.

Close a window

Choose **File ➤ Close** from the main menu, or choose **Close** from the window's system menu, or double click the window's system menu, or press CTRL+F4.

Activate a window

Click anywhere within the window, or select the document name from the **Window** menu. Alternatively, press CTRL+F6, or CTRL+TAB until the window you wish is active.

Move a window

Drag the document window's title bar with the mouse. Alternatively, choose **Move** from the window's system menu, then use the cursor keys, then press ENTER to set the window in place.

Resize a window

Drag its border with the mouse. Alternatively, choose **Size** from the window's system menu, use the cursor keys to resize, then press ENTER to resume editing.

Maximize a window

Press the maximize button on the document window's title bar; or choose **Maximize** from the window's system menu.

Iconize a window

Press the minimize button on the document window's title bar; or choose **Minimize** from the window's system menu.

Restore iconized

To restore an iconized document window, double click the document window icon; or choose **Restore** from the icon’s system menu.

Cycle to next window

To switch to the next window, press CTRL+F6 or CTRL+TAB.

Tile the windows

To arrange all open document windows side by side, choose **Window ➤ Tile vertically** or **Window ➤ Tile horizontally** from the main menu. This provides easy access to documents, as in the illustration below.

Cascade windows

To arrange all open document windows so that the title bars are all visible, choose **Window ➤ Cascade** from the main menu.

Using the Text Editor Tools

Typing and editing source code with the Text Editor is similar to typing documents with most word processor program. Type the code as if you were typing at a typewriter, then use the various Text Editor tools and commands to rearrange, duplicate, and modify your code.

File Menu

The Text Editor **File** menu has the following special file-oriented commands.

Print
To print the file, choose the **Print** command. This opens the standard print dialog so you can one or more copies or one or more pages of the file.

Save All
To save all open source files, choose this command.

Import File
Calls the Open File dialog, allowing you to insert the contents of a file into the open file at the insertion point.

Export Block
Saves the selected text in a new source code document under a new name which you specify. If the file exists, the Text Editor overwrites the file; if the file does not exist, the Text Editor creates it.

Edit Menu

To use an editing command, such as **Cut** or **Copy**, highlight the text you wish the command to act on, then choose the command from the menu or the toolbar.

When you wish to insert text, click with the I-Beam cursor at the place you wish to insert text, then type or **Paste** the new text.

The **Edit** menu features the primary editing commands. The following sections detail the commands on the **Edit** menu:

Undo
To undo the most recent editing action, choose the **Undo** command. This menu item changes to which action will be undone. Should you type a line of text, the menu item will show that you may **Undo Line Edits**. Should you delete a line of code, it will allow you to **Undo Block Delete**.

Certain commands cannot be undone, such as a File Save, or a Replace All.

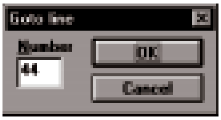
Cut To delete the highlighted text from the document and hold it in the clipboard, choose the **Cut** command. The keyboard accelerator is CTRL+X. The toolbar button with a scissors icon also activates this command.

Copy
To copy the highlighted text and hold it in the clipboard, use the **Copy** command. The keyboard accelerator is CTRL+C. The toolbar button with the overlapping pages icon also activates this command.

Paste
To place the contents of the clipboard (text only) into the document at the insertion point use the **Paste** command. The keyboard accelerator is CTRL+V. The toolbar button with the page on clipboard icon also activates this command.

Select All
To highlight all the text in the document so that the next editing command affects the entire document, choose the **Select All** command. The keyboard accelerator is CTRL+.

Goto Line
To jump to a specific source code line to edit, choose **Goto Line**. The keyboard accelerator is CTRL+G.


Goto line
Number: 44

The Text Editor places the insertion point in the first column of the line number you type in the dialog box. The status bar reflects the current line and column numbers for the insertion point position.

Goto Next Error
To move the insertion point to the next compiler error, choose this command (or CTRL+). The Editor places the cursor at the part of the statement where it detected the error. This command is only enabled following a compile which generated errors.

Goto Previous Error
To move the insertion point to the previous location at which the source code generated a compiler error, choose this command (or CTRL-). The Editor places the cursor at the part of the statement where it detected the error. This command is only enabled following a compile which generated errors.

Set/Clear Tabstop
Places or removes a custom tab stop at the insertion point.

Duplicate Line

To duplicate the entire line and insert the copy on the next line, choose this command or press CTRL+2. The original line need not be highlighted; simply position the cursor anywhere on the line.

Toggle Case

To change the case of next character following the insertion point, choose this command or press CTRL+/. A lower case letter becomes upper case, and vice versa.

Delete Line

To delete the entire line on which the insertion point is located, choose this command, or press CTRL+Y.

Delete Word

To delete the word following the insertion point, choose this command, or press CTRL+T.

Format Structure

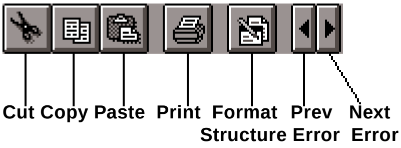
Think of this as 'visually editing' a window or report. Just place the insertion point on any line within the structure, and choose this command, or press CTRL+F. The toolbar button with the pencil and paper icon also activates this command. The Window Formatter (or Report Formatter) displays a visual representation of the structure, ready for editing.

When you exit the Window Formatter (or Report Formatter), your source code reflects the changes you made. This provides seamless interaction at the source code level with the visual design tools.

You may also place the insertion point on a blank line, then call the Window Formatter to create a new structure. When you return to the Text Editor, the source code document will contain the new structure you created with the Window Formatter (or Report Formatter). Be sure to place the structure in the data section of the program.

Tool Bar

The Text Editor toolbar provides quick access to the most frequently used edit commands: Cut, Copy, Paste, and Format Structure. These commands are the same commands accessed with the edit menu. Additionally there is a print button to call the standard windows print dialogs. CLICK these buttons to quickly access your favorite commands.



Populate Field Toolbox

The Text Editor **Populate Field** toolbox provides quick access your data dictionary fields and memory variables. Place your insertion point, then DOUBLE-CLICK a field or variable in the **Populate Field** toolbox. The Text Editor enters the fully qualified variable name at the insertion point!

This toolbox is available when you are embedding source code. See *Application Generator—Embedded Source Code*.

Search Menu

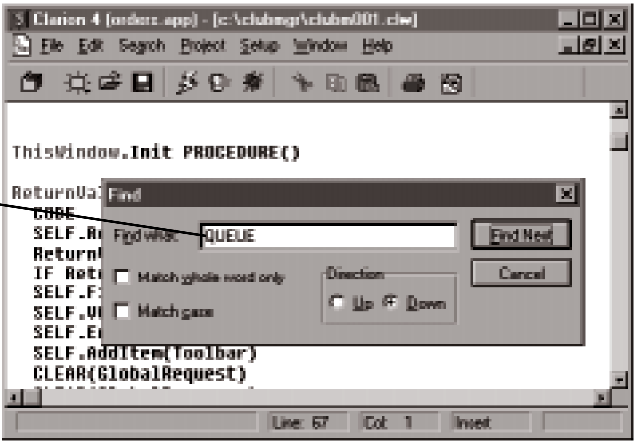
The **Search** menu makes it easy to find and change text in your source code documents. You may search for specific text, change single or multiple occurrences of text throughout the document, or simply highlight a variable, then jump to the next occurrence of it in the code.

The commands on the **Search** menu are:

Find

To find the next occurrence of a word, type it in the **Find** dialog and press the **Find Next** button. The keyboard accelerator is ALT+F3.

The **Find** dialog is modeless. This means that the dialog will remain on screen so that you may easily search again.



Clarion 4 [orders.app] - [c:\clubmgr\clubm001.clw]

```
ThisWindow.Init PROCEDURE()

ReturnVa
  CODE
  SELF.R
  Return
  IF Ret
  SELF.F
  SELF.V
  SELF.E
  SELF.AddItem(Toolbar)
  CLEAR(GlobalRequest)
```

Find

Find what: QUEUE

□ Match whole word only

□ Match case

Direction: ○ Up ● Down

Line: 67 Col: 1 Insert

The Find dialog, searching for "QUEUE."

1. In the **Find What** field, type the text to search for.

   The default contents of the **Find What** field is the last text searched for.

2. Optionally check the **Match whole word only** box, the **Match case** box, or both.

   For example if you search for ‘find’ with **Match whole word only**, you will not get ‘findings.’ If you search for ‘Find’ with **Match case** you will not get ‘find.’

3. Specify whether to search upwards or downwards.

4. Press the **Find Next** button to start the search.

Replace

To change a specific text string, type the original text and the replacement text in the **Replace** dialog. You may make the changes one at a time, throughout a selected text block, or throughout the entire document.

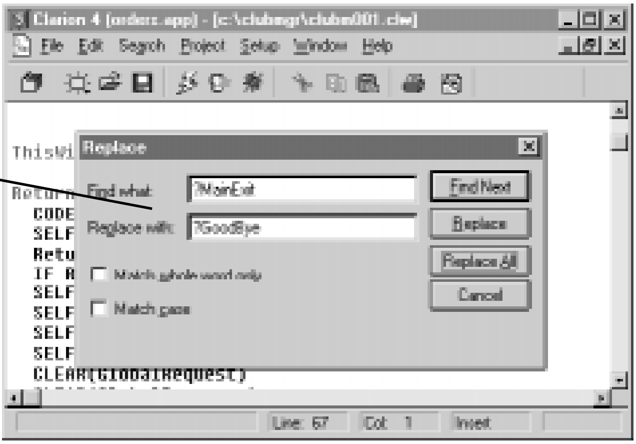
The **Replace** dialog is modeless. This means that the dialog will remain on screen so that you may easily search and replace again.

1. In the **Find What** field, type the original text to search for.

   The default contents of the **Find What** field is the last text searched for.

2. In the **Replace with** field, type the replacement text.

   The default contents of the **Replace with** field is the previous replacement text.



Clarion 4 (orders.app) - [c:\clubmgr\clubm001.clw]

Find what: ?MainExit

Replace with: ?GoodBye

□ Match whole word only

□ Match case

The Replace dialog — changing the name of an equate from ?MainExit to ?GoodBye.

3. Optionally check the **Match whole word only** box, the **Match case** box, or both.

   For example if you search for ‘find’ with **Match whole word only**, you will not get ‘findings.’ If you search for ‘Find’ with **Match case** you will not get ‘find.’

4. Press the **Find Next** button to display the next occurrence of the original text and stop before changing it.

The dialog will ask you to confirm the change.

5. Press the **Replace** button to replace the next occurrence of the original text without confirmation.

6. Press the **Replace All** button to change all the occurrences of the original text without confirmation.

   Replace All operates only on the selected block of text. If no text is selected, it operates on the entire document.

Find Next

To search for the same text you last searched for, choose this command or press the F3 key. This searches in a 'forward' direction.

Find Previous

To search for the same text you last searched for in a backward direction, choose this command or press SHIFT+F3.

Find Marked Text

To quickly find the next occurrence of the currently highlighted text, choose this command or press CTRL+F3. This is equivalent to executing the **Find** command, typing the currently selected text in the **Find What** field, and specifying a forward search.

Block Indent

The Text Editor supports block indents, that is you can shift multiple lines of code to the left or right so all those pesky IFs line up with their ENDs.

To move a block of text, simply select the block then press the tab key. The **Block Indent** dialog appears. Specify the amount and direction of the indent, then press **OK.**

Macros

The Text Editor supports simple macros.

To create a macro

1. Press CTRL+=.

   That is, while holding the CTRL key, press the = key. The **Press Key for Macro** dialog appears.

2. Press the key that will invoke the macro, for example CTRL+z.

3. Press **OK**.

   The macro recorder is now active and recording your keystrokes.

4. Perform the operation to save.

   Type text, TAB, ENTER, function keys, etc.

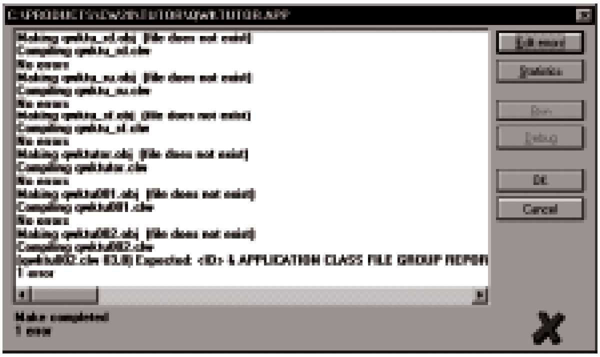
5. When you are finished, press CTRL+= again.

   The macro is saved and may be invoked by pressing the key you specified.

Note: The macro is not saved between sessions and is only available until you close Clarion.

Editing Errors

One of the chief entry points into the Text Editor or Embeditor is through the make results dialog—when an error aborts the make.



C:\PRODUCTS\CW20\TUTOR\QWKTUTOR.APP

```
Making qwktu_sf.obj  (file does not exist)
Compiling qwktu_sf.clw
No errors
Making qwktu_ru.obj  (file does not exist)
Compiling qwktu_ru.clw
No errors
Making qwktu_sf.obj  (file does not exist)
Compiling qwktu_sf.clw
No errors
Making qwktutor.obj  (file does not exist)
Compiling qwktutor.clw
No errors
Making qwktu001.obj  (file does not exist)
Compiling qwktu001.clw
No errors
Making qwktu002.obj  (file does not exist)
Compiling qwktu002.clw
qwktu002.clw (3,8) Expected: <ID> & APPLICATION CLASS FILE GROUP REPOR
1 error
```

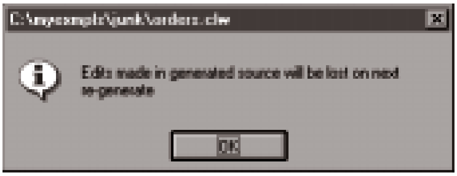
Make completed
1 error

The **Edit errors** button automatically calls the Text Editor, and places the insertion point at the position where the compiler detected the error. You may then edit the source code to correct the mistake. The **Edit ➤ Goto Next Error** command is available to jump to the next compile error once you correct the first error. Alternatively, CTRL+ and CTRL- jump to the next error and previous error, respectively.

You can control the mode in which the Text Editor/Embeditor opens. See *Application Generator—Configuring the Application Generator—Edit errors in context* for more information.

Note: **If you directly edit a generated source module, your changes will be overwritten the next time you generate source. Clarion warns you if you attempt to directly edit a generated source module from the make results dialog.**

To make permanent changes in a generated source module you must change a template prompt, use the Embedded Source dialog, or use the Embeditor.



C:\myexample\junk\orders.clw

Edits made in generated source will be lost on next re-generate

Configuring the Text Editor

To personalize your editing environment, use the **Editor Options** dialog, the **Application Options** dialog, and optionally, the ..\BIN\C5EDT.INI file.

Application Options Dialog

To open the **Application Options** dialog, choose **Setup ➤ Application Options**. Select the **Editor** tab to set Text Editor options. See *Application Generator—Configuring the Application Generator* for more information.

Editor Options Dialog

To open the **Editor Options** dialog, choose **Setup ➤ Editor Options**. Select the corresponding tab to set specific Text Editor options.

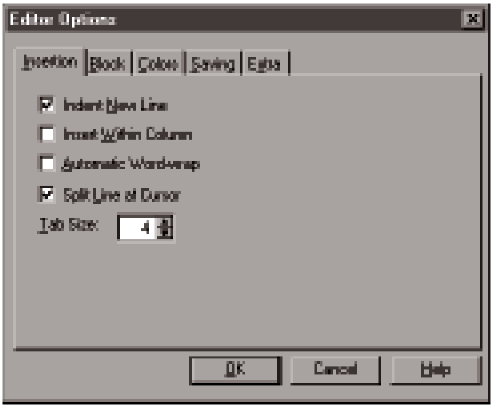
Insertion

This tab provides the following text insertion options.

Indent New Line
To automatically give a new line the same indention as the previous line, check this box. This will make your code more readable.

Insert Within Column
When the insertion point is in the middle of a line, ENTER adds a new line after the current line.

Automatic Word-wrap
To cause automatic line breaks at column 70, check this box.



☑ Indent New Line
☐ Insert Within Column
☐ Automatic Word-wrap
☑ Split Line at Cursor
Tab Size: 4

Split Line at Cursor

When this box is checked, ENTER splits the current line at the insertion point (cursor). The second part of the line moves to a new line. When this box is not checked, ENTER inserts a blank line below the current line, *without* splitting the current line.

Tab Size

To set the default spacing between tabs, enter a number in the **Tab Size** box.

Block

This tab provides the following block selection options.

Automatic Block Delete

To delete the selected text when pasting, check this box. To insert before a selected block, clear the box.

Remove Block On Copy

To delete the selected text when copying, check this box.

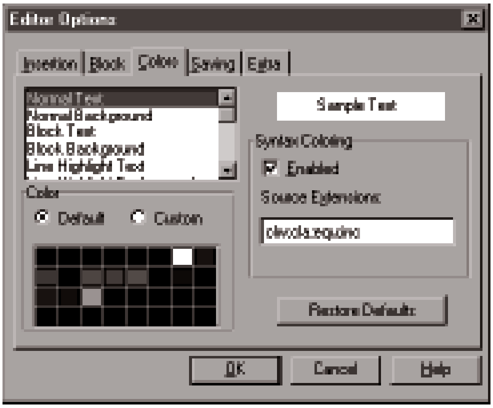
Colors

These options let you set color choices for twenty-one different Clarion language elements. Make Clarion keywords appear in red, or make equates appear in green.

Select a language or text element in the **Color Groups** list box, then CLICK on a color selection box. The sample text shows you how the selected language element will appear in the Text Editor.

Color Groups

Highlight the language or text element to receive a color assignment.



● Default ○ Custom

☑ Enabled

Source Extensions: clw;cla;equ;inc

Color

To assign a color to the selected language element, CLICK on a color selection box.

Default

To assign the default color to the selected language element, check this box.

Custom

To reset the custom color for the selected language element, check this box.

Sample Text

Shows how the selected language element will appear in the Text Editor.

Enabled

To apply the color syntax highlighting to the file types listed in the **Source Extensions** box, check this box.

Source Extensions

To specify the file types that color syntax highlighting is applied to, type a list of file extensions separated by semicolons.

Restore Defaults

To assign the default colors to all language and text elements, check this box.

Saving

These options let you determine when the Text Editor creates backup files and saves primary files.

Make Backup Files

To make a backup file (.BAK) each time you explicitly save a source file, check this box. The .BAK file contains the source as it was previously saved.

Prompt for Reload if file changed

To receive a "source.CLW has changed on disk. Do you want to reload?" message whenever the Text Editor detects such a change, check this box.

Automatic Save time (minutes)

Specify the time interval between automatic saves. When the interval expires, the Text Editor saves the file under its original name. Despite the automatic saves, cancelling the edit session restores the file to its unedited state.

Extra

This tab provides the following miscellaneous options.

Home to First Column

This box controls the action of the HOME key in the Text Editor. Check the box to make the HOME key position the cursor to column 1. Clear the box to make the HOME key position the cursor to the first non-blank column.

Show Field Box in Embed Editor

Check this box to show the Populate Field toolbox when editing embedded source code. This lets you choose variable names from a pick list, so you don't have to remember them.

Text Editor INI File

The Clarion Text Editor has its own .INI file that you can edit to customize the Text Editor's behavior. The behaviors you can set with the .INI file are Key Mapping and Color Mapping. After you have changed the .INI file, you must restart Clarion before your changes will take effect.

Key Mapping

Through the Key Mapping section of the C5EDT.INI file, you can specify the Text Editor functions associated with specific keys on your keyboard.

For example, the default NewLine function is mapped to the ENTER key:

```
NewLine=EnterKey
```

You can change this association simply by typing a new keycode equate (see ..\LIBSRC\KEYCODES.CLW) to the right of the equal (=) sign:

```
NewLine=TabKey
```

Please be careful not to map the same key twice.

Some handy but unmapped functions you may want to map are:

SetMark1 through SetMark9

These functions set "bookmarks" within a source module. You can jump back to the bookmarks with the corresponding JmpMark function. Bookmarks are not persistent between sessions or across multiple files.

```
SetMark1=CtrlAlt1
SetMark2=CtrlAlt2
```

JmpMark1 through JmpMark9

These functions jump to previously set "bookmarks" within a source module. You can set the bookmarks with the corresponding SetMark function.

```
JmpMark1=Alt1
JmpMark2=Alt2
```

UpperCaseWord

This function changes the selected text to uppercase.

```
UpperCaseWord=CtrlU
```

CommentBlock

This function changes the selected block to comments by inserting an exclamation point in the first column of each line.

```
CommentBlock=CtrlShiftC
```

UnCommentBlock

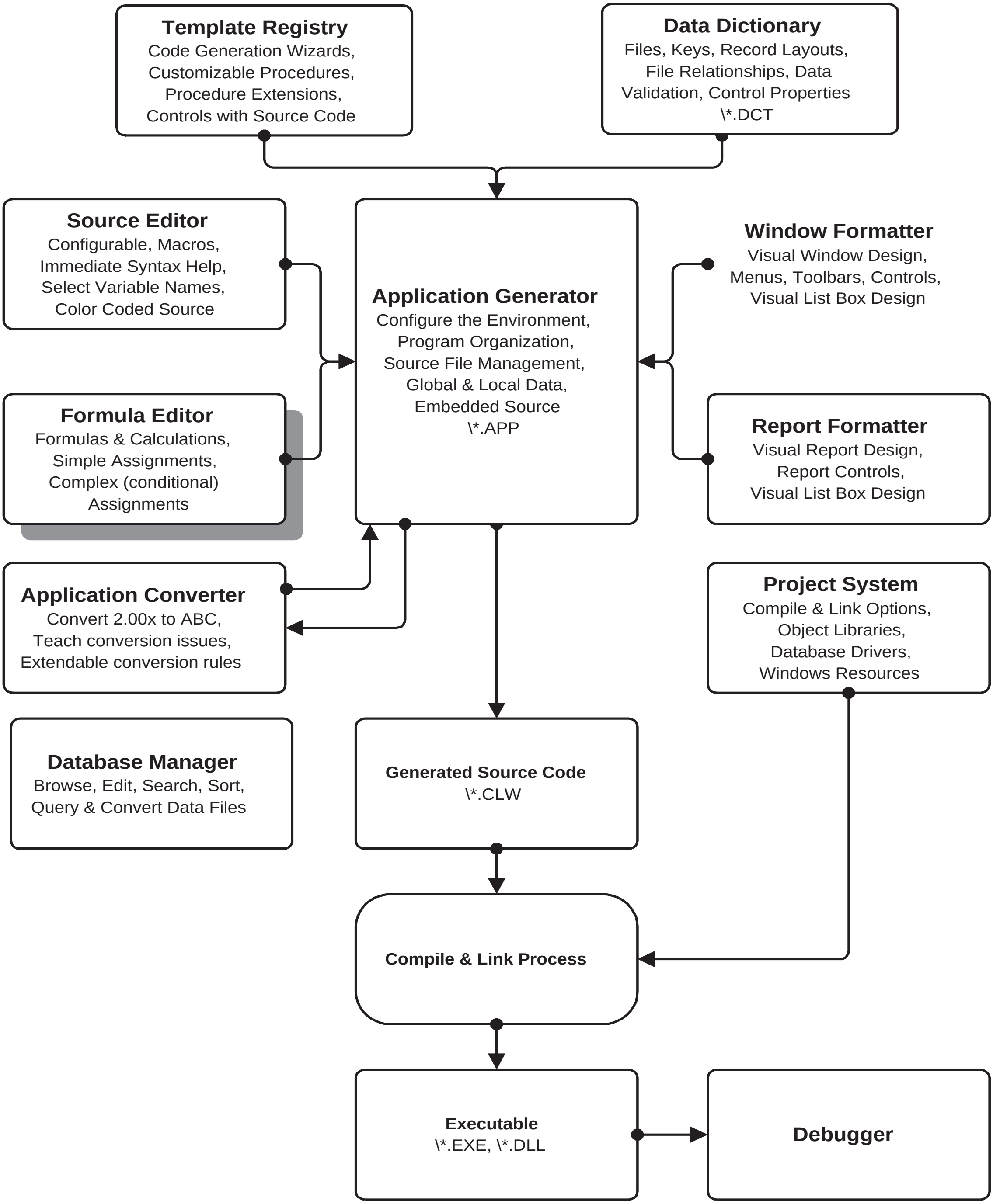
This function changes the selected block to compiled code by removing the exclamation point from the first column of each line.

```
UnCommentBlock=CtrlShiftU
```

Color Mapping

We recommend using the **Color Options** tab of the **Editor Options** dialog to customize the Text Editor colors. However, if you want more color customization, please back up your \BIN\C5EDT.INI file, then experiment with the various color assignments available in the .INI file.

10 - FORMULA EDITOR



Overview

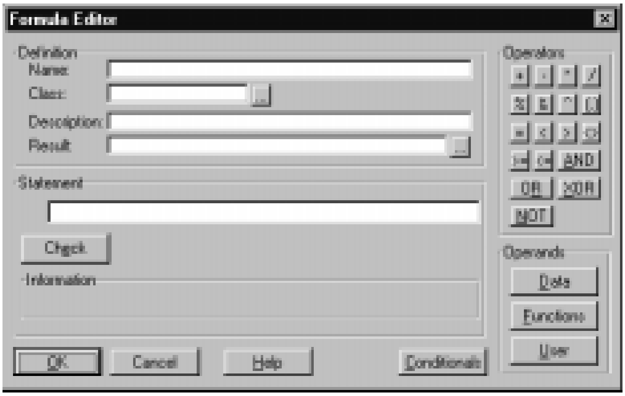
The Formula Editor helps you to quickly generate a statements or structures that assign a value to a variable. You can use the Formula Editor to create unconditional or conditional assignments.

- An unconditional assignment assigns the evaluation of an expression to the variable you specify: variable = expression. For example, a field called GrossPrice might receive the result of adding two fields called BasePrice and Tax.

- A conditional assignment places multiple possible assignments within a structure that executes only one of them. The Formula Editor builds IF structures and CASE structures for this purpose. The assignment statement executed depends on the evaluation of the IF or CASE condition. For example, a conditional field called "Tax" could equal 0 when "Taxable" (the IF condition) evaluates as false, or "Tax" could equal Price times TaxRate if "Taxable" is true.

The **Formula Editor** dialog provides access to data dictionary fields, as well as global and local memory variables, and helps you create syntactically correct expressions. This is its prime advantage: automatic syntax checking.

To create an expression, you press buttons to add expression components to the **Statement** line. You can also type in your expression then check the syntax upon completion.



Expressions

An expression is made up of two types of components: *operands* and *operators*. Operators perform an operation (such as addition, subtraction, etc.) on one or more operands of the expression. Operands are the components on which operations are performed. Operands either contain or return a value. Constants, data dictionary fields, memory variables, and functions are examples of operands. An operand can be made up of more than one component, such as a function and its parameters.

The **Formula Editor** lets you choose operators and operands, then insert them into the **Statement** line.

The table below lists all the components used in Clarion expressions.

Math Operators

+	Plus sign: Adds two operands together.
-	Minus sign: Subtracts one operand from another.
*	Asterisk: Multiplies one operand by another.
/	Slash: Divides one operand by another.
%	Percent sign: Returns the remainder from a division operation (modulus division).
^	Caret: Raises one operand to the power of the other.
( )	Parentheses: Groups components together within an expression.

Logical (Boolean) Operators

=	Equal: Evaluates whether one expression is equal to the other.
<	Less Than: Evaluates whether one expression is less than the other.
>	Greater Than: Evaluates whether one expression is greater than the other.
<>	Not Equal: Evaluates whether one expression is not equal to the other.
>=	Greater or Equal: Evaluates whether one expression is greater than or equal to the other.
<=	Less or Equal: Evaluates whether one expression is less than or equal to the other.

AND

Connects two logical expressions together. For an expression containing an AND to be true, both expressions of the AND must be true.

OR

Connects two logical expressions together. An expression containing an OR is true if either expression of the OR is true.

XOR

Connects two logical expressions together. An XOR expression is true if either expression is true, but not both.

NOT

Reverses the evaluation of an expression.

~

Reverses the evaluation of an expression.

String Operators

&

Ampersand: Appends one text string to another.

Operands

Data

Includes data dictionary fields, global, and local memory variables.

Functions

All of the built-in functions of the Clarion language. These functions all perform some operation on parameters (other operands) and return a value.

User

Any FUNCTION in your application. These functions perform some operation on parameters (other operands) and return a value.

Constant Text

You can type constant text surrounded in single quotes (‘A’) on the **Statement** line.

Constant Number

You can type constant numbers on the **Statement** line. Constant numbers can be represented in any valid format, such as Decimal (1 or 1.2345), Scientific Notation (22e4), Binary (0101b), or Hexadecimal (1AFFh).

Example Expressions

```
Lastname & ‘, ‘ & FirstName     !concatenated string constants & variables
ABS(Amount) * 100               !function, numeric variable & constant
TaxCode = True AND Amount > .25 !Boolean with variables & constants
```

Formula Editor Tools

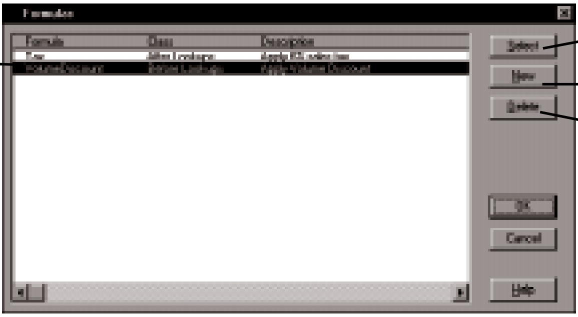
The Formula Editor consists of three dialog boxes:

Formulas Manages all the formulas you have created for the procedure.

Formula Editor Creates simple assignment statements.

Conditionals Creates conditional structures (IF..THEN or CASE..OF).

Formulas Dialog



Formula	Class	Description
Tax	After Lookups	Apply KS sales tax
VolumeDiscount	Before Lookups	Apply Volume Discount

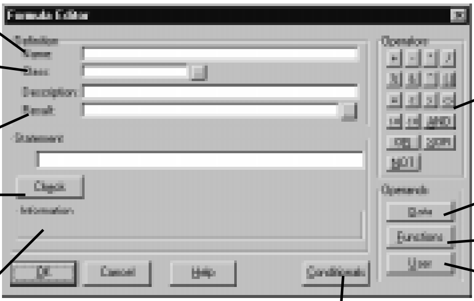
List of Formulas for a procedure with its execution class and a description

Selects the highlighted formula for editing

Enables creation of a new formula

Deletes the highlighted formula

Formula Editor



A descriptive label

Determines *when* the expression is evaluated

Variable to which the value is assigned

Validates your expression's syntax

Displays additional information about an expression's component

Creates Conditional structures

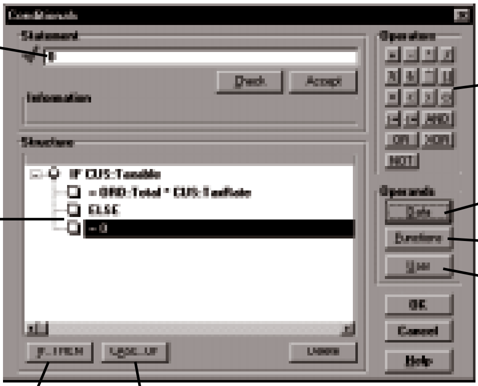
Operator buttons

Accesses the Data Dictionary fields and memory variables

Accesses Clarion's built-in functions

Accesses FUNCTIONS in your application

Conditionals Dialog



Expression to insert into the structure

Control structure expanding tree

Operator buttons

Accesses the Data Dictionary fields and memory variables

Accesses Clarion's built-in functions

Accesses FUNCTIONS in your application

Creates an IF structure

Creates or expands a CASE structure

All Formula Editor Assignments

The Formula Editor helps you to quickly generate a statements or structures that assign a value to a variable. You can use the Formula Editor to create unconditional or conditional assignments. In either case, you must always name the variable that is the target of the assignment, and you must choose the point in the generated procedure the expressions are evaluated and the assignment is executed.

To create *any* formula (conditional or unconditional) with the Formula Editor, you begin with the following steps.

Preliminary Steps

1. From the **Procedure Properties** dialog, press the **Formulas** button.

   This opens the **Formulas** dialog.

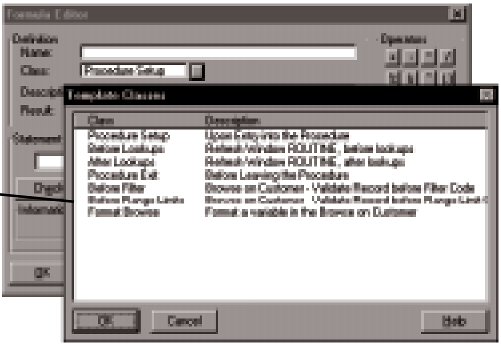
2. Press the **New** button to begin a new formula.

   This opens the **Formula Editor** dialog.

3. In the **Name** field, type a name for the formula.

   This does not affect the formula operation. It simply serves as an identifier or documentation for the formula.

4. Press the ellipsis (...) button next to the **Class** field to choose a Formula Class.



Class: Procedure Setup

Class	Description
Procedure Setup	Upon Entry into the Procedure
Before Lookups	Refresh Window ROUTINE, before lookups
After Lookups	Refresh Window ROUTINE, after lookups
Procedure Exit	Before Leaving the Procedure
Before Filter	Browse on Customer - Validate Record before Filter Code
Before Range Limits	Browse on Customer - Validate Record before Range Limit
Format Browse	Format a variable in the Browse on Customer

Choosing a formula class.

A formula class is simply a template embed point. The formula class determines *where* in the generated source code, the calculation is performed. Each Clarion procedure template has its own set of formula classes. For example, in the Form Template there is a class called “After Lookups” which tells the Application Generator to compute the formula after all lookups to secondary files are completed for the procedure.

The formula class you select displays in the **Formulas** dialog.

> **Note: Do not confuse formula classes with template classes. A *template* class is simply a group of templates. A *formula* class is a point within a procedure template where the formula is evaluated (a template embed point).**

5. In the **Description** field, type a description of the formula.

   This does not affect the formula operation. It simply serves as an identifier or documentation for the formula. The Description displays in the **Formulas** dialog.

6. In the **Result** field, type the variable to which the result of the expression is assigned, or press the ellipsis (...) button to choose a variable from the **Select Field** dialog.

   You can choose a local, module, or global variable, or a data dictionary field. This variable name displays in the **Formulas** dialog.

Unconditional Assignments

The Formula Editor helps you to quickly generate a statements or structures that assign a value to a variable. To create an unconditional assignment of the form variable = expression, do the following:

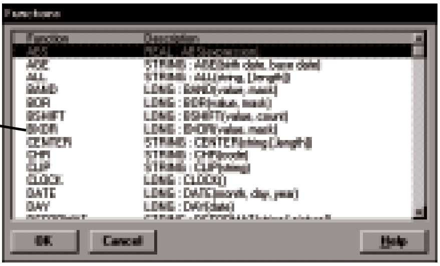
1. Follow the steps described in *Preliminary Steps*.

   This establishes the target of the expression as well as when the expression is evaluated and the assignment executed.

2. In the **Statement** field, create the expression.

   You may type the expression, use the **Formula Editor's** buttons, or both. The first component of an expression must be an operand, a left parenthesis, or a unary minus (the negative sign).

   For example, press the **Data** button and choose a variable, press the Multiply by (*) button, then press the **Functions** button to choose a Clarion built-in function. See *Expressions* for more information.



Functions

Function	Description
ABS	REAL : ABS(expression)
AGE	STRING : AGE(birth date, base date)
ALL	STRING : ALL(string, [length])
BAND	LONG : BAND(value, mask)
BOR	LONG : BOR(value, mask)
BSHIFT	LONG : BSHIFT(value, count)
BXOR	LONG : BXOR(value, mask)
CENTER	STRING : CENTER(string,[length])
CHR	STRING : CHR(code)
CLIP	STRING : CLIP(string)
CLOCK	LONG : CLOCK()
DATE	LONG : DATE(month, day, year)
DAY	LONG : DAY(date)

Some of the built-in Clarion functions you can choose from.

3. Press the **Check** button to check the syntax of the expression.

   If the syntax is correct, a large green check mark appears to the left of the statement. If the syntax is incorrect, a large red X appears indicating you need to correct the syntax.

4. When the syntax is correct, press the **OK** button.

Conditional Assignments

Creating conditional expressions with the **Formula Editor** actually creates control structures in the source code. There are two structures you can create with the **Formula Editor**—an IF or a CASE structure. You can also nest either of these structures, creating complex conditional statements.

An IF structure assigns a value to the Result variable based on the true/false evaluation of a *single* logical expression. There are only *two* possible assignments, because only one condition is tested for. If the condition tested is true, one assignment is made, if not true (false), then the other assignment is made. See *IF* in the *Language Reference* for more information.

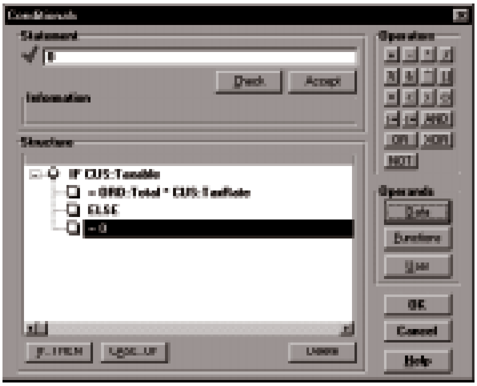
Nesting IF structures allows additional alternative assignments. However, the CASE structure offers a less complicated method for assigning values based on the evaluation of multiple logical expressions. See *Creating a CASE Structure*.

Creating an IF Structure

Use a simple IF structure to assign one of two values to the Result field depending on a condition. For example, you may want to determine the tax for an order. The tax depends on a condition—is the customer taxable or nontaxable? The resulting control structure would be:

```
IF CUS:Taxable                        ! conditional expression
  TAX = ORD:Total * CUS:TaxRate       ! True assignment expression
ELSE
  TAX = 0                             ! False assignment expression
END
```

The control structure in the **Conditionals** dialog would look like the illustration below.



To create an IF structure:

1. Follow the steps described in *Preliminary Steps*.

   This establishes the target of the expression as well as when the expression is evaluated and the assignment executed.

2. Press the **Conditionals** button.

   This opens the **Conditionals** dialog.

3. Press the **IF..THEN** button.

   This inserts an IF structure in the **Structure** window.

Define the IF condition to evaluate

1. CLICK on IF in the **Structure** window.

2. On the **Statement** line, enter the IF condition to evaluate.

   You can type the expression, or you can use the **Operators** and **Operands** buttons to select expression components, or you can do both.

3. Press the **Check** button to check your syntax.

   If the syntax is correct, a large green check mark appears to the left of the statement. If the syntax is incorrect, a red X appears indicating you need to correct the syntax.

4. When the syntax is correct, press the **Accept** button to insert your expression into the IF structure.

Define the true assignment expression

1. CLICK on line immediately below the IF in the **Structure** window.

   This is where the “True” assignment expression goes.

2. On the **Statement** line, enter the “True” assignment expression.

   You can type the expression, or you can use the **Operators** and **Operands** buttons to select expression components, or you can do both. If the IF condition is true, this expression is evaluated and the resulting value is assigned to the target variable.

3. Press the **Check** button to check your syntax.

   If the syntax is correct, a large green check mark appears to the left of the statement. If the syntax is incorrect, a red X appears indicating you need to correct the syntax.

4. When the syntax is correct, press the **Accept** button to insert your expression into the IF structure.

Define the false assignment

1. CLICK on line immediately below the ELSE in the **Structure** window.

This is where the "False" assignment expression goes.

2. On the **Statement** line, enter the "False" assignment expression.

   You can type the expression, or you can use the **Operators** and **Operands** buttons to select expression components, or you can do both. If the IF condition is false, this expression is evaluated and the resulting value is assigned to the target variable.

3. Press the **Check** button to check your syntax.

   If the syntax is correct, a large green check mark appears to the left of the statement. If the syntax is incorrect, a red X appears indicating you need to correct the syntax.

4. When the syntax is correct, press the **Accept** button to insert your expression into the IF structure.

Note: Neither a "true" nor a "false" assignment expression is required. If you don't enter an assignment expression, then the application generator generates an "empty" IF structure and no assignment occurs.

5. When your control structure is complete, press the **OK** buttons in the **Conditionals, Formula Editor, and Formulas** dialogs.

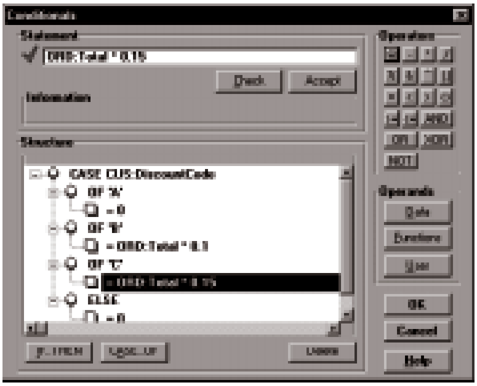
Creating a CASE Structure

A CASE structure selectively assigns a value to the Result variable based on the evaluation of multiple OF expressions against the CASE expression. Practically speaking, there are unlimited alternative assignments because any number of expressions may be evaluated. See *CASE* in the *Language Reference* for more information.

A simple CASE structure can be used to assign one of several values to the Result field depending on which OF expression is equal to the CASE expression. For example, you may wish to offer varying discounts for large purchases depending on the customer's discount code. The resulting CASE structure might be:

```
CASE CUS:DiscountCode   !CASE expression, compared to OF expressions
  OF 'A'                        ! 1st OF comparison expression
    Discount = 0                ! 1st OF assignment expression
  OF 'B'                        ! 2nd OF comparison expression
    Discount = ORD:Total * .1   ! 2nd OF assignment expression
  OF 'C'                        ! 3rd OF comparison expression
    Discount = ORD:Total * .15  ! 3rd OF assignment expression
  ELSE
    Discount = 0                ! catchall assignment
END
```

This control structure appears in the formula editor as shown below.



ORD:Total * 0.15

```
CASE CUS:DiscountCode
    OF 'A'
        = 0
    OF 'B'
        = ORD:Total * 0.1
    OF 'C'
        = ORD:Total * 0.15
    ELSE
        = 0
```

To create a CASE structure:

1. Follow the steps described in *Preliminary Steps*.

   This establishes the target of the expression as well as when the expression is evaluated and the assignment executed.

2. Press the **Conditionals** button.

   This opens the **Conditionals** dialog.

3. Press the **CASE..OF** button.

   This inserts a CASE structure in the **Structure** window.

Define the CASE condition to evaluate

1. CLICK on CASE in the **Structure** window.

2. On the **Statement** line, enter the CASE expression to evaluate.

   The CASE expression is compared to the multiple OF expressions.You can type the expression, or you can use the **Operators** and **Operands** buttons to select expression components, or you can do both.

3. Press the **Check** button to check your syntax.

   If the syntax is correct, a large green check mark appears to the left of the statement. If the syntax is incorrect, a red X appears indicating you need to correct the syntax.

4. When the syntax is correct, press the **Accept** button to insert your expression into the CASE structure.

Build the OF expressions

1. CLICK on the OF line below the CASE line in the **Structure** window.

   This is where the first OF *comparison* expression goes.

You can type the expression, or you can use the **Operators** and **Operands** buttons to select expression components, or you can do both. At runtime, if the CASE expression equals this OF expression, then the subsequent expression is evaluated and the resulting value is assigned to the target variable.

3. Press the **Check** button to check your syntax.

   If the syntax is correct, a large green check mark appears to the left of the statement. If the syntax is incorrect, a red X appears indicating you need to correct the syntax.

4. When the syntax is correct, press the **Accept** button to insert your expression into the CASE structure.

5. Highlight the line below the OF line in the **Structure** window.

   This is where the first OF *assignment* expression goes.

6. On the **Statement** line, insert the OF *assignment* expression.

   You can type the expression, or you can use the **Operators** and **Operands** buttons to select expression components, or you can do both. At runtime, if the CASE expression equals the above OF expression, then this assignment expression is evaluated and the resulting value is assigned to the target variable.

7. Press the **Check** button to check your syntax.

   If the syntax is correct, a large green check mark appears to the left of the statement. If the syntax is incorrect, a red X appears indicating you need to correct the syntax.

8. When the syntax is correct, press the **Accept** button to insert your expression into the structure.

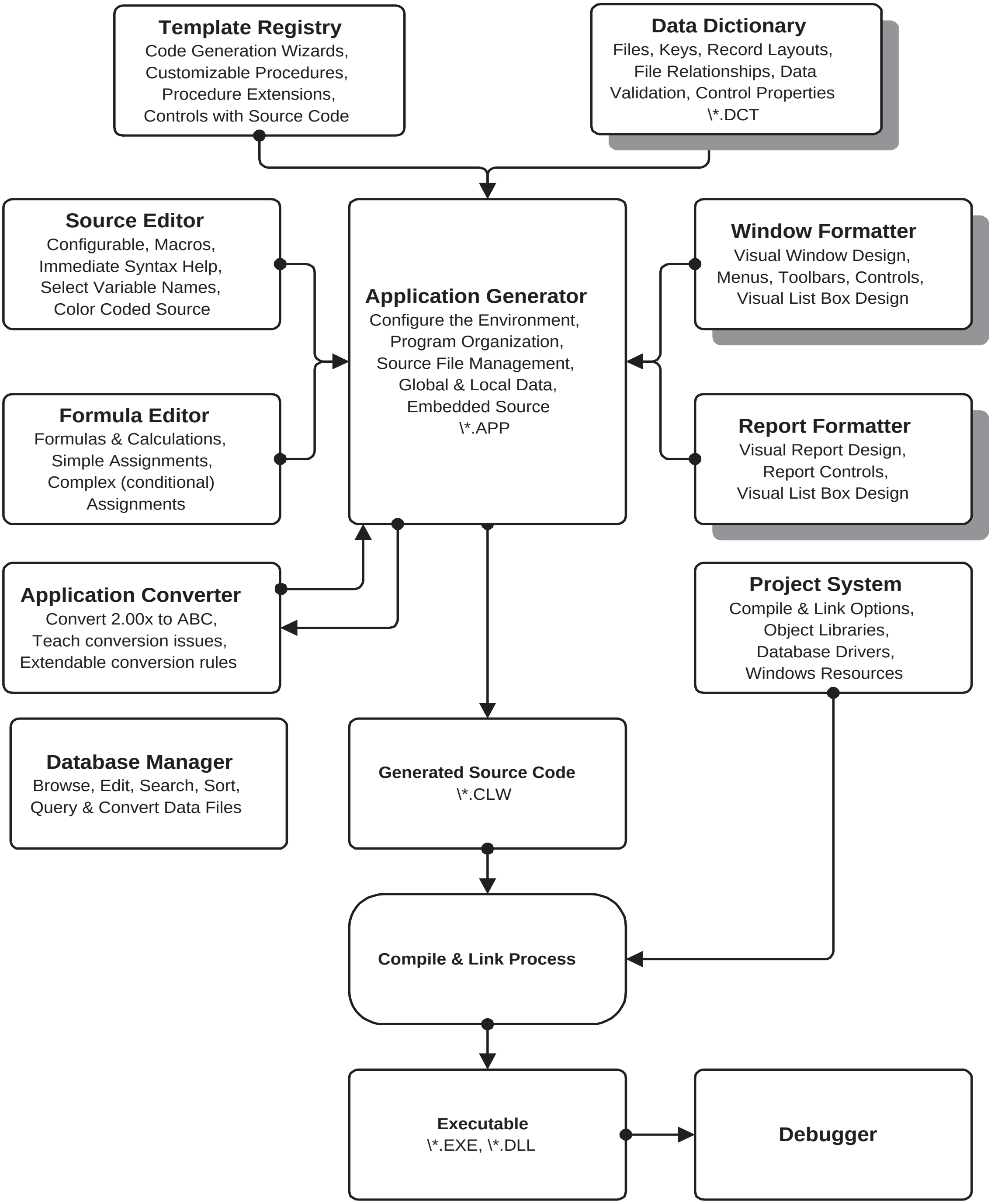
To add additional OF statements

1. Highlight an OF line in the **Structure** window.
2. Press the **Case..OF** button
3. Insert your expressions in the same manner as above.
4. When your control structure is complete, press the **OK** buttons in the **Conditionals, Formula Editor, and Formulas** dialogs.

Nesting Structures

Either of the available control structures can be nested inside another. This lets you create complex structures simply by combining the instructions in the previous sections for creating IF and CASE structures.

11 - PICTURE EDITOR


Template Registry
Code Generation Wizards, Customizable Procedures, Procedure Extensions, Controls with Source Code
Data Dictionary
Files, Keys, Record Layouts, File Relationships, Data Validation, Control Properties
*.DCT
Source Editor
Configurable, Macros, Immediate Syntax Help, Select Variable Names, Color Coded Source
Application Generator
Configure the Environment, Program Organization, Source File Management, Global & Local Data, Embedded Source
*.APP
Window Formatter
Visual Window Design, Menus, Toolbars, Controls, Visual List Box Design
Formula Editor
Formulas & Calculations, Simple Assignments, Complex (conditional) Assignments
Report Formatter
Visual Report Design, Report Controls, Visual List Box Design
Application Converter
Convert 2.00x to ABC, Teach conversion issues, Extendable conversion rules
Project System
Compile & Link Options, Object Libraries, Database Drivers, Windows Resources
Database Manager
Browse, Edit, Search, Sort, Query & Convert Data Files
Generated Source Code
*.CLW
Compile & Link Process
Executable
*.EXE, *.DLL
Debugger

Edit Picture Dialog

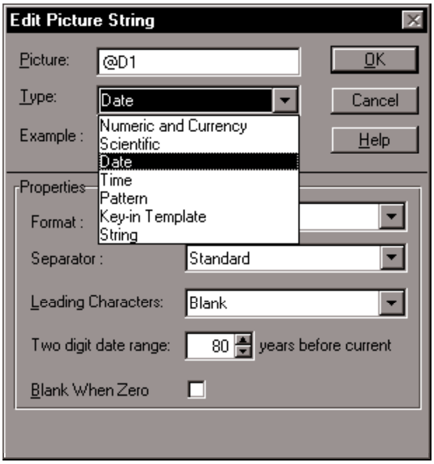
Many controls, such as ENTRYs, COMBOs, STRINGs, etc., display variable values as well as constant text. These controls provide a **Picture** field instead of the **Text** field. The Picture Editor (**Edit Picture String** dialog) helps you supply an appropriate picture token (display format) in the **Picture** field.

You can invoke the **Edit Picture String** dialog from the Dictionary Editor (see *Dictionary Editor—Defining Field Properties*) or from the ***control* Properties** dialog (see *Controls and Their Properties—Setting the Display Picture*)

To set the display picture for variable values, type a picture token in the **Picture** field or press the ellipsis button to use the **Edit Picture String** dialog. See the *Language Reference* for more information on picture tokens.

There is a great variety and diversity of picture token syntax which depends on the type of data you format: strings, numbers, currency, scientific, dates, times, etc.

The **Edit Picture** dialog lets you quickly and easily build an appropriate picture token without memorizing picture token syntax. Invoke this easy to use dialog by pressing the ellipsis (...) button beside the **Picture** prompt in the ***control* Properties** dialog or the **Field Properties** dialog.


Edit Picture String

Picture: @D1

Type: Date

Example :

Properties

Format :

Separator : Standard

Leading Characters: Blank

Two digit date range: 80 years before current

Blank When Zero □

Example

An example of the display format currently specified in the dialog. What you see is what you get.

Favorite Pictures Pool

Pool

Select a predefined picture from the drop list.

Save As

Press this button to save the displayed picture to the Pool, and name the saved picture. The saved pictures are available in the **Pool** drop-list. You can save your most frequently used pictures, then quickly reuse them from the Pool.

Tip: The Picture Editor stores the pictures in the ..\BIN\C5Pict.ini file.

Delete

Press this button to delete the displayed Pool picture currently displayed in the **Pool** field.

Picture

The picture token currently specified. This picture token produces the example shown.

Picture Type

Choose the type of data to format from this drop-down list. Choose from String, Numeric and Currency, Scientific Notation, Date, Time, Pattern, and Key-in Template.

String

String picture tokens specify a length with no other formatting. See *String Pictures* in the *Language Reference*.

Length

Specify the length of the string. This length also determines the width of the control if the width is not otherwise specified by the control’s AT attribute.

Numeric and Currency

Numeric and currency picture tokens specify a length, plus conventional formatting to convey positive and negative values, various currencies, etc. See *Numeric and Currency Pictures* in the *Language Reference*.

Size

The total number of significant digits, plus any formatting characters. For example, $22.25- is 4 significant digits + 3 formatting characters for a size of 7.

Decimal Digits

The number of digits to the right of the decimal.

Currency

Choose from None, Leading, and Trailing. None shows no currency symbol. Leading puts the currency symbol to the left of the number and Trailing puts the symbol right of the number.

Symbol

The currency symbol to display: either a dollar sign ($) or a string constant.

Negative Sign

Specify how negative values are formatted. Choose from:

Bracket Negatives surrounded by parentheses.

Leading Negatives get a leading minus sign.

Trailing Negatives get a trailing minus sign.

None No sign display.

Decimal Separator

Specify the character inserted between the integer and fractional portion of the value. Choose from:

Period Period is the separator.

Comma Comma is the separator.

None Displays no separator.

Grouping

Specify the character inserted at every third digit to aid readability. Choose from:

Comma Comma is the separator.

Period Period is the separator.

Space Space is the separator.

Hyphen Hyphen is the separator.

Leading Character

Specify the character to represent leading zeroes.

Clip Remove leading zeroes so that any leading format characters abut the left most digit.

Zero Leading zeroes display as zeroes (0).

Space Leading zeroes display as spaces ( ).

Asterisk Leading zeroes display as asterisks (*).

Blank When Zero

Check this box to display nothing when the value is zero.

Scientific

Scientific Notation picture tokens let you display very large or very small numbers with a decimal format raised by a power of ten. The display takes the form -9.99e+999. See *Scientific Notation Pictures* in the *Language Reference*.

Number of Characters

The total number of characters, including the 7 format characters. For example, -1.96e+007 requires 10 characters.

Leading Digits

The number of digits to the left of the decimal point (typically 1).

Decimal Separator

Specify the character inserted at every third digit to aid readability.

Point	Period is the separator.
Comma	Comma is the separator.
Space	Space is the separator.

Separator

Specify the character inserted between the integer and fractional portion of the value.

Point	Period is the separator.
Comma	Comma is the separator.

Blank When Zero

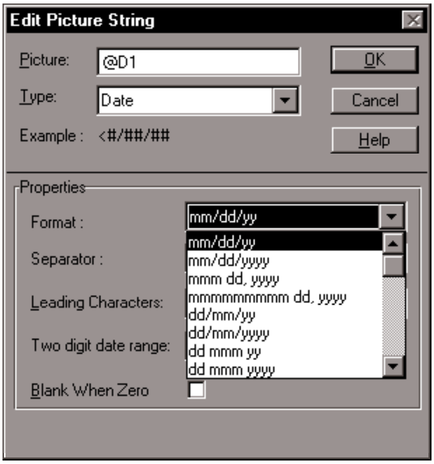
Check this box to display nothing when the value is zero.

Date

Date Notation picture tokens let you display dates in a number of different formats. Choose the format you want from the **Format** drop-down list. See *Date Pictures* in the *Language Reference*.

Better still, date picture tokens in entry fields automatically invoke Clarion's run-time date parsing functions, so you can enter '21' and Clarion expands it to the 21st day of the current month and year. Or you can enter 'DEC' and Clarion expands it to the 1st day of December of the current year. The date is then formatted according to the picture token.

> **Tip: The MASK attribute (Entry Patterns check box) on a window preempts the date parsing functions.**



Edit Picture String

Picture: @D1

Type: Date

Example : <#/##/##

Properties

Format : mm/dd/yy

Format

Choose the format you want from the drop-down list. What you see is what you get except for the Windows Short and Windows Long formats. Additionally, the separator character and leading zeroes may be specified independent of the chosen format.

Windows Short Uses the short date format specified in the Windows control panel or the Windows 95 Regional Settings control panel.

Windows Long Uses the long date format specified in the Windows control panel or the Windows 95 Regional Settings control panel.

Separator

Choose from Standard (/), Period (.), Dash (-), Space ( ), and Comma (,).

Leading Characters
Specify the character to represent leading zeroes.

Zero Leading zeroes display as zeroes (0).

Blank Remove leading zeroes.

Asterisk Leading zeroes display as asterisks (*).

Two digit date range
Change the default century interpretation for dates input with a two digit year. By default, Clarion assumes any date input with a two digit year (i.e. the century value is omitted) falls between today-80 years and today+19 years.

For example, if today is June 1, 1996 and the date input is 9/2/59, Clarion assumes the 59 means 1959 since 1959 falls between today-80 years (June 1, 1916) and today+19 years (June 1, 2015). To force a different interpretation, set the **Two digit date range** to 30. Now Clarion assumes the 59 means 2059 since 2059 falls between today-30 years (June 1, 1966) and today+69 years (June 1, 2065).

Blank When Zero
Check this box to display nothing when the value is zero.

Time

Time Notation picture tokens let you display times in a number of formats. Choose the format you want from the **Format** drop-down list. See *Time Pictures* in the *Language Reference*.

Format Choose the format you want from the drop-down list. What you see is what you get except for the Windows Short and Windows Long formats. Additionally, the separator character and leading zeroes may be specified independent of the chosen format.

Windows Short Uses the time format specified in the Windows control panel.

Windows Long Uses the time format specified in the Windows control panel.

Separator
Choose from Standard (:), Period (.), Dash (-), Space ( ), and Comma (,).

Leading Character
Specify the character to represent leading zeroes.

Zero Leading zeroes display as zeroes (0).

Blank Remove leading zeroes.

Blank When Zero
Check this box to display nothing when the value is zero.

Pattern

Pattern picture tokens let you build custom display formats for various *numbers*: phone numbers, social security numbers, room numbers, dates, times, measurements, etc. See *Pattern Pictures* in the *Language Reference*.

Picture
Type the picture token between the 'P's according to the Legend below. Your picture token can include any displayable characters, including all the standard keyboard characters.

At runtime, the constants in the picture token display just as they appear in the token. The left angle (<) and the pound sign (#) resolve into the individual digits from the display variable.

Legend
< integer, blank if zero
# integer
constant (any displayable character except < and #)

Blank When Zero
Check this box to display nothing when the value is zero.

Tip: To use a lowercase p in your picture, use an uppercase P at the start and end of your picture token. To use an uppercase P in your picture, use a lowercase p at the start and end of your picture token.

Key-in Template

Pattern picture tokens let you build custom edit formats for STRINGs, CSTRINGs and PSTRINGs containing *mixed alphanumeric characters*. Although Key-in tokens affect output as well as input, their primary purpose is to provide custom field editing and validation on input. See *Key-in Template Pictures* in the *Language Reference*.

Picture
Type the picture token between the 'K's according to the Legend below. Your picture token can include any characters (displayable or not), including all the standard keyboard characters.

Legend

< accept an integer, blank if zero
# accept an integer
? accept any character (even non-display)
^ accept an upper case character
_ accept a lower case character
| input may stop here
constant (any displayable character except <#?^_| or \)
\ display next character (lets you display <#?^_| or \)

Only alphabetic characters

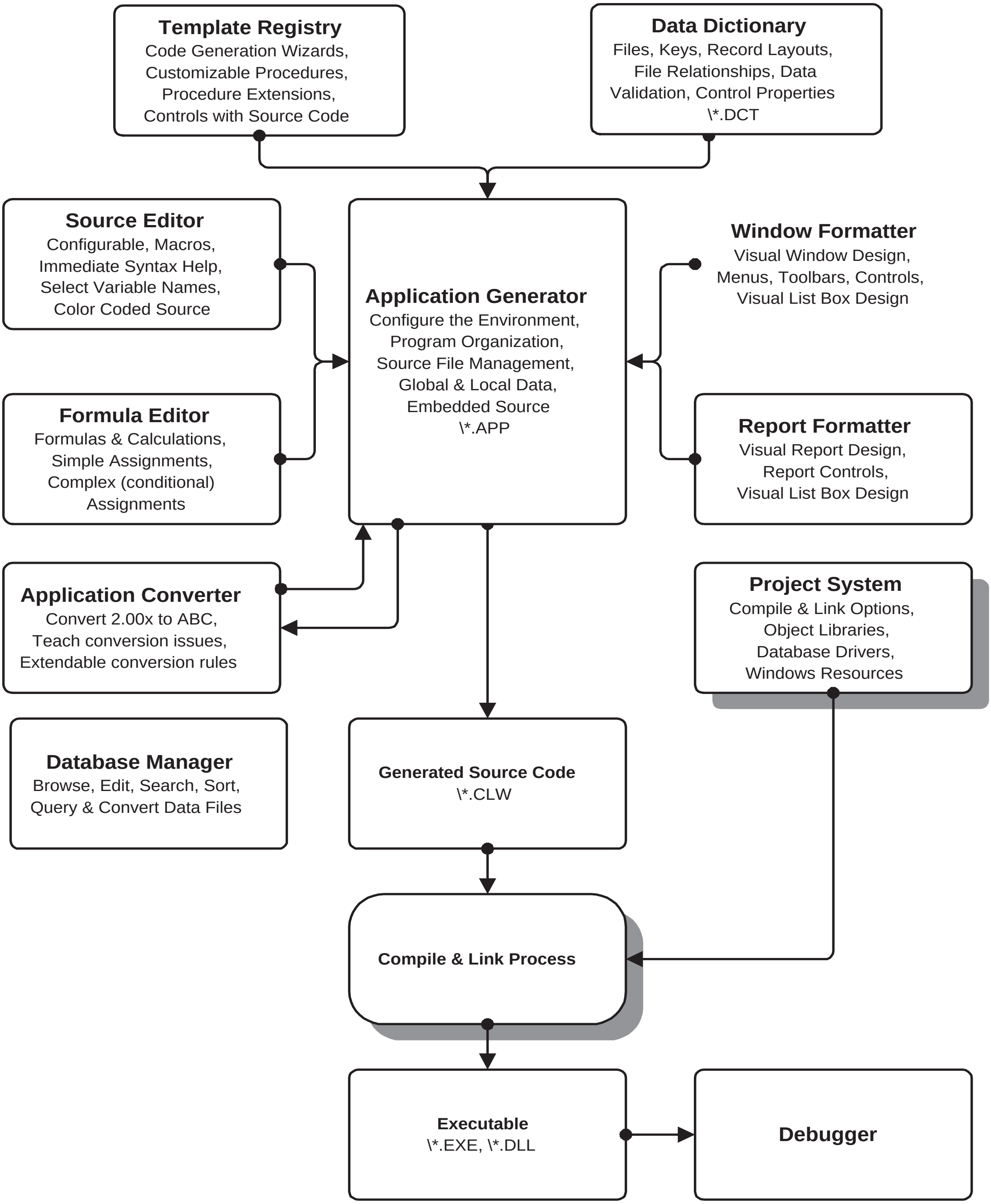
Check this box to accept only alphabetic characters.

Blank When Zero

Check this box to display nothing when the value is zero.

Tip: To use a lowercase k in your picture, use an uppercase K at the start and end of your picture token. To use an uppercase K in your picture, use a lowercase k at the start and end of your picture token.

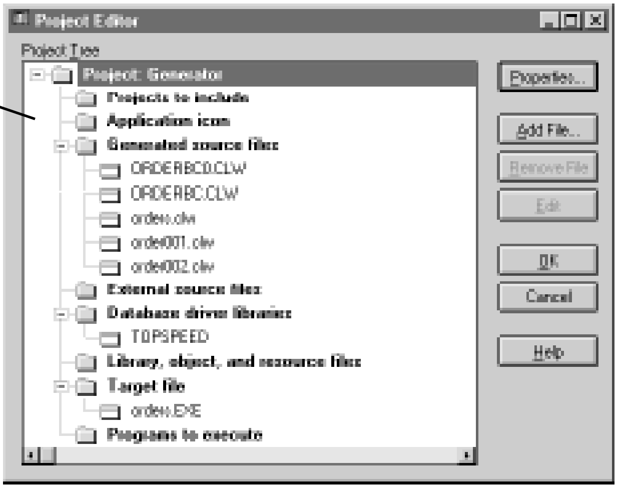
12 - PROJECT SYSTEM



Overview

The project file (.PRJ) or the application file (.APP) tracks all the components that are used to create the final executable (target file) for your application. It also stores the compiler options ranging from whether to include debug code or not, to setting a preferred optimization method. The compiler and the linker depend on this project information to tell them how, and what, to compile and link. The project information is functionally equivalent to a MAKE file for other language compilers.

The Clarion Project System visually manages the project information. It maintains tree diagrams of the source files, external libraries, resources, and other project components.



The Project Editor dialog contains the Project Tree. The tree controls expand and contracts when you click them. When expanded, they list the component files. When contracted, the box contains a plus (+) to show that you can expand the control.

This chapter discusses the Project System and related topics. It shows you how to:

- Add source code files to the Project Tree.
- Add external libraries to the Project Tree, and how to access their functions and procedures in your source code.
- Specify the target file and set other compiler options. The target file is the ultimate executable created for your application or project

Hand Coded Projects

This section provides an *overview* of the steps necessary to create a project file (*.PRJ). The Project file tracks all the components that are used to create the executable file. It also sets the compiler options ranging from whether to include debug code or not, to setting a preferred optimization method.

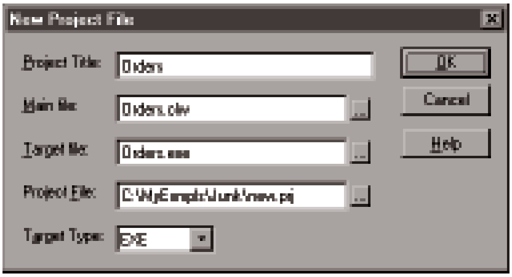
If you use the Application Generator to create your source code, no separate .PRJ file is created, and the only thing you will probably use the Project System for is to set debugging options. The Application Generator takes care of maintaining most everything else for you. Therefore, this chapter is primarily concerned with using the Project System for hand-coded programs.

The **Project Tree** dialog organizes all the project components and provides access to other dialogs that manage your project file.

To create a project file

1. Choose **File** ➤ **New** ➤ **Project**.

   This opens the **New Project** dialog.



Project Title: Orders
Main file: Orders.clw
Target file: Orders.exe
Project File: C:\MyExamples\Junk\new.prj
Target Type: EXE

2. In the **Project Title** field, type a descriptive name for your project.

   This is documentary only.

3. In the **Main File** field, type the name of your main source code file, or press the ellipsis (...) button to choose a source file with the Windows File dialog.

   This step automatically fills in the **Target file, Project file** and **Target Type** fields as well. If you wish to set non-default values you may do so.

4. Press the **OK** button.

   This opens the **Project Tree** dialog.

Add Project Components

1. CLICK on a Project Tree component category (such as *Database driver libraries*), then press the **Add File** button.

This opens a dialog where you can select a component of the specified type. For Database driver libraries, the **Add File** button calls the **Select Driver** dialog so you can choose from a list of valid Clarion database drivers. In all other instances, the **Add File** button opens the Windows File dialog to help you locate the component file.

2. DOUBLE-CLICK the component fileto add it to your project.
3. Repeat the above steps for all component files.

Set Global Make Options

1. CLICK on the **Project** (first) line then press the **Properties** button.

   This opens the **Global Options** dialog. This dialog sets various compile and link options for your entire project, including optimization method, type of executable created, whether to include debug code, etc.

   Select the tabs in the **Global Options** dialog to get an idea of the available options, or press the **Help** button to see a description of each option. See *Compile and Link Options* for more information on these compile and link options.

Set Local Make Options

1. CLICK on the **Project:** (first) line, then press the **Properties** button.

   This opens the **Global Options** dialog. This dialog sets various compile and link options for your entire project, including optimization method, type of executable created, whether to include debug code, etc.

   Like the **Add File** button, the **Properties** button behaves differently depending on which Project Tree folder is highlighted. When a *source file* is highlighted, the **Properties** button calls the **Compile Options** dialog. This dialog sets compile options for *the specific source file* highlighted. *Specific* compile options take precedence over *global* compile options.

   For now, press the tabs in the **Global Options** dialog to get an idea of the available options, or press the **Help** button to see a description of each option. When you are finished, press the **OK** button. See *Setting Project File Options* below, for more details on setting these compile and link options.

Compile and Link Options

This section provides an overview of the steps necessary to maintain a project file. Maintaining the project file includes adding and removing source files, object files, libraries, etc. from the compile and link process. In addition, you may set both global and local compile and link options in the project file.

Global Compile and Link Options

Select the first line of the Project Tree listing and press the **Properties** button to open the **Global Options** dialog. This includes the following options:

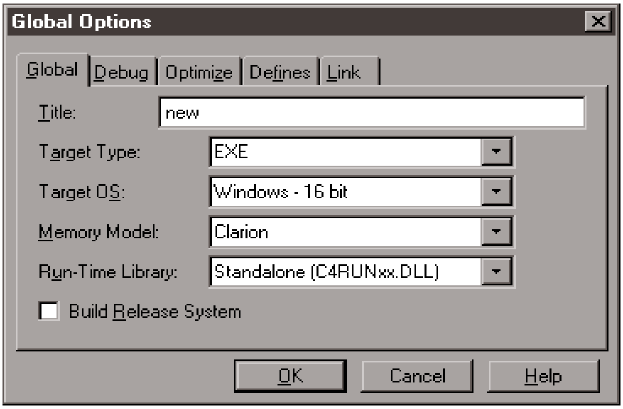
Global Tab

Title

To add a short text description, type it in the **Title** field. The Project System will list the description next to the Project name in the Project Tree list.

Target Type

To specify the executable file type, choose **.EXE, .LIB**, or **.DLL** from the **Target Type** drop list.



Global Options

Title: new

Target Type: EXE

Target OS: Windows - 16 bit

Memory Model: Clarion

Run-Time Library: Standalone (C4RUNxx.DLL)

□ Build Release System

Target OS

To specify the executable's targeted operating system, choose Windows 16-bit or Windows 32-bit from the **Target OS** drop list.

Note: You can compile and link 32-bit executables with Windows 3.1 if you have Win32S installed, but you must have Windows 95 or Windows NT to run them.

Memory Model

The Clarion model is not optional.

Run-Time Library

To specify how the run-time library is called by the target file, choose **Standalone, Local**, or **External** from the **Run-Time Library** drop list. Available only in Professional and Enterprise Editions. See *Development and Deployment Strategies* for more information.

Standalone Creates the target file so it calls the Clarion run-time library as C5RUN[x].DLL. The [x] indicates 32-bit library.

Local Creates the target file with the Clarion run-time library linked internally (a "one-piece" executable). Available only in Professional and Enterprise Editions.

External Links the application so it calls the Clarion run-time library from a .DLL which you have created with the run-time library linked internally and exported. Available only in Professional and Enterprise Editions. See the *Programmer's Guide* for more information on the export file (.EXP).

Build Release System

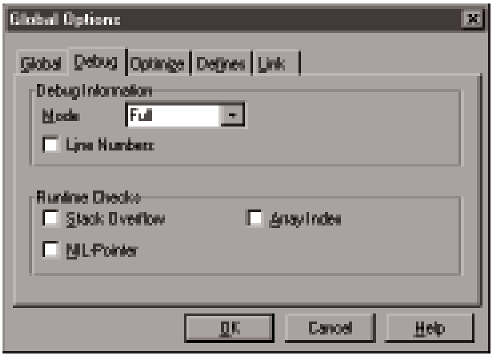
To create an executable for release, check the **Build Release System** box. To create an executable for use with the Debugger, clear the **Build Release System** box.

Debug Tab

Debug Mode

To specify the level of debug capability, choose **Off, Min**, or **Full** from the **Mode** drop list.

For 16-bit programs, the compiler generates debug information into separate .DBD files. By default the .DBD files are in the \CLARION5\OBJ folder. For 32-bit programs, the compiler generates the debug information into the .EXE or .DLL.



Global Options

Debug Information
Mode: Full
□ Line Numbers

Runtime Checks
□ Stack Overflow
□ Array Index
□ NIL Pointer

Line Numbers

To specify line numbers be built into the object file, check the **Line Numbers** box. This is not necessary for the Clarion debugger, but may be helpful when using other debuggers.

Stack Overflow

To enable stack overflow warnings at run-time, check the **Stack Overflow** box.

NIL-Pointer

To allow compiler warnings when dereferencing null pointers, check the **NIL-Pointer** box.

Array Index

To enable array index larger than the array size warnings at run-time, check the **Array Index** box.

Optimize Tab

Clarion 5 fully optimizes all executables. These options have no effect.

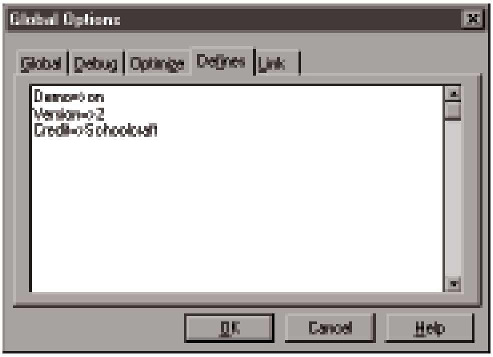
Defines Tab

Defines

To define a switch, or switches for use with the COMPILE and OMIT compiler directives, type a list of valid Clarion labels. Each label defines a separate switch. The default value of each switch is ‘on.’

To specify a non-default value for the switch, type label=>value.

Defines refers to the Project System language statement #PRAGMA DEFINE(). The #PRAGMA DEFINE() statement creates a switch that can be toggled on and off. See the *Programmer’s Guide* for more information on #PRAGMA DEFINE(). The switch can then be interrogated by the COMPILE and OMIT compiler directives. See the *Language Reference* for more information on COMPILE and OMIT.



```
Demo=>on
Version=>2
Credit=>Schoolcraft
```

For example, type ‘Demo’ in the **Defines** field. The Project

System will create a switch called Demo and turn it "on." Now you can use the switch in conditional COMPILE and OMIT statements within your source code. For example:

```
COMPILE('END COMPILE',DEMO=ON)
  IF TODAY() > FirstRunDate + 30
    #ReturnCode = MESSAGE('Beta period expired')
    RETURN
  END
END COMPILE
```

Zero Divide

To enable division by zero using floating point variables (REAL, SREAL, BFLOAT4, etc.) in Clarion subexpressions, the zero_divide switch should be set in the application's project file.

Type 'zero_divide' in the **Defines** field. The Project System creates the switch and turns it "on." Compile your application, then, at run-time, a division by zero will return a zero rather than a floating point exception.

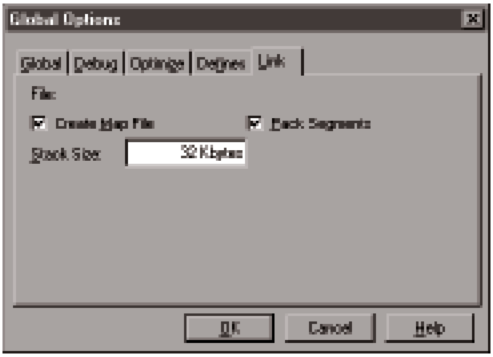
Link Tab

Create Map File

To create a map file, which contains information about segment sizes and public functions, check the **Create Map File** box. The map file may be used with third party debuggers.

Pack Segments

To pack the data and program segments in the .EXE file, check the **Pack Segments** box.



File:

☑ Create Map File ☑ Pack Segments

Stack Size: 32 Kbytes

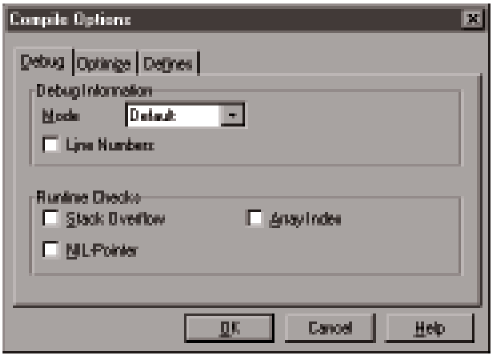
Stack Size

To specify the stack size, type a number and unit of measure in the **Stack Size** field.

Individual Source Module Compile Options

You may set compile options for individual source modules as well as for the project as a whole. Individual compile settings take precedence over *global* compile settings. By setting the compile options for individual source modules, you may specify full debug information for one module and none for another.

Highlight a source code file in the Project Tree dialog, then press the **Properties** button. This opens the **Compile Options** dialog, showing most of the same options as the **Global Options** dialog. This dialog sets compile options for the individual source module highlighted.



Debug Information

Mode: Default

□ Line Numbers

Runtime Checks

□ Stack Overflow □ Array Index

□ NIL Pointer

See *Debug Tab*, *Optimize Tab*, and *Defines Tab* above for information on using this dialog.

Component Files

Projects to Include

The Project System can compile and link other projects referenced in the current project file. The other project can even specify yet another, in a cascading sequence of compile references.

Cascading projects lets you split the development process into separate projects, then link them all together when you're ready.

❏ To add a project to the Project Tree, highlight *Projects to include* in the Project Tree list. Then press the **Add File** button and select the project file you wish to add with the standard **Open File** dialog.

Application Icon

You can specify icon files to link first into your executable. Windows displays the first linked icon in various contexts, for example, on the Windows 95 taskbar, or in the Windows 3.x File Manager. To specify the first linked icon file, highlight **Application Icon**, then press the **Add File** button and select the icon file with the standard **Open File** dialog.

To specify the icon for individual windows, use the ICON attribute for the window. See *Window Formatter—Window Properties Dialog* for more information on specifying icons.

For windows with no icon specified, the default (Clarion) icon is used. You can use System{PROP:Icon} to specify the default icon for your application.

Source Code Files

Source code files are those files that contain your Clarion Language statements (or other language statements, if you have TopSpeed compilers to support them). Adding a source code file to the **Project Editor** dialog is simply a matter of DOUBLE-CLICKING a file name.

External Source Files

❏ To add "hand-coded" source code files, highlight *External source files.* Then press the **Add File** button and select the file you wish to add with the Windows file dialog.

If you choose the wrong *type* of file (for example, a generated source file, or a .LIB file), the Project System adds the file to the appropriate Project Tree folder.

Generated Source Files

For .APP files,*Generated source files* cannot be added or deleted with the Project System. They can only be added or deleted with the Application Generator. Any attempt to add source modules to the will add the source file to the *External source files*.

Database Driver Libraries

Your application calls various database driver routines to access your database files. These routines are in libraries supplied with Clarion and installed by default in the \LIB subdirectory by the Clarion setup program. During the *link* process, references to these external routines can only be resolved if the library containing the routines is added to your project file.

The Application Generator automatically adds the appropriate driver libraries based on the Data Dictionary file driver selections and the Project System's 16-bit or 32-bit settings. For hand coded projects, you should manually add the appropriate driver libraries.

❏ To link a database driver library, highlight *Database driver libraries* in the Project Tree list. Then press the **Add File** button and select the driver to add from the **Select Driver** dialog.

Library, Object, and Resource Files

Libraries and Objects

An object (.OBJ) file contains objects—routines, functions, or procedures that can be linked into your program during the *link* process. A library (.LIB) file is simply a file that contains multiple objects. When properly linked, your program can call on these objects to perform certain tasks.

> **Tip: You can create .LIB files for use with your Clarion applications. See *Setting the Target File* below and see *Development and Deployment Strategies*.**

Objects used in this manner need not be written with Clarion. Your Clarion programs can call objects compiled from C, C++, Pascal, etc.

❏ To link a library, object or resource file, highlight *Library, object, and resource files* in the Project Tree list. Then press the **Add File** button and select the file you wish to add with the Windows file dialog.

The .LIB, .OBJ, .RSC, etc. file appears in the Project Tree, and any objects from the file that are properly referenced in your source code are linked into your target (executable) file.

Resource Files

A resource file is any other file (.RSC, .ICO, .BMP) that should be linked into your program. One of the most likely things you'll do with the Project System is to specify *resources* to link into your executable. By linking them into the executable, you avoid having to ship them as separate, external files.

> **Tip: For 16-bit projects with Run-Time Library set to Local, you should add to your project all the.RSC, .ICO, .BMP, etc. files used within the project's LIBs. By default, the .RSC files are in the \CLARION5\OBJ folder. Failure to add required .RSC files generates a WSLDIAL error.**

If you directly reference a graphic file within a data structure, the compiler automatically links the graphic, so there is no need to add the graphic file to your Project Tree. For example, if you place an IMAGE control in a window, and specify a file by name in the **Image Properties** dialog, the linker automatically includes that file in your executable. But if you assign a different graphic to a control using a run-time property assignment statement, the linker will only include the new file in your executable if you add the file to your Project Tree.

> **Tip: The Clarion runtime libraries assume the .EXE or .DLL where a window was most recently opened is where any referenced icons are located.**

To add graphic files to the executable

1. Highlight *Library, object and resource files* then CLICK on the **Add File** button.

   Select the bitmap, icon, or metafile graphic from the standard Open File dialog.

2. Press the **OK** button to return to the **Project Editor** dialog.

3. Highlight the source code file that references the graphic, then CLICK on the **Edit** button.

   The Text Editor opens the source code file.

4. Place a tilde (~) in front of the graphic file name in the source code assignment statement (not in the data section).

   For example: change ?Image{PROP:Text} = 'I.ICO' to ?Image{PROP:Text} = '~I.ICO.' The tilde indicates the program should find the item as a linked in resource, not as an external file.

   Optionally, choose **Search ➤ Find** to locate the file name.

5. Choose **File ➤ Exit**, then CLICK on **Yes** when asked if you want to save.

   Now, when you recompile and link, the executable will no longer require the external graphic file.

Programs to Execute

Programs to execute lets you customize the compile and link process by executing the program(s) of your choice at the end of the process. These can be .BAT files or more sophisticated .EXE files that perform any additional tasks you specify as part of the compile and link process. The programs execute in the order they appear in the Project Tree, commencing immediately *after* the target file is made. That is, after the entire compile and link process is completed.

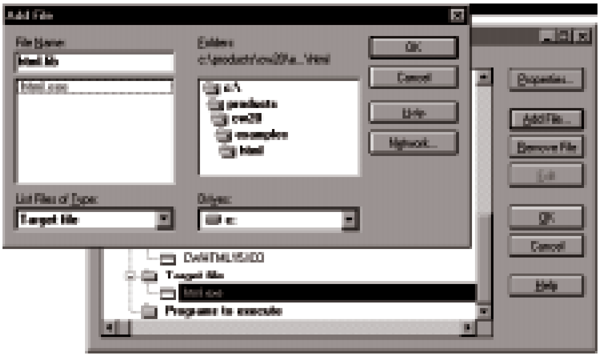
❏ To add a program to execute to the Project Tree, highlight *Programs to execute* in the Project Tree list. Then press the **Add File** button and select the program file you wish to add with the Windows file dialog.

The Target File

Using the Project System, you can create .EXE, .LIB, and .DLL files. This section describes these three target file types and how to create them. See *Adding Object Files and Libraries* and *Distributing Files* for more information. See also *Development and Deployment Strategies*.

❑ By default, the project system creates a standard executable (.EXE + C5RUN[x].DLL) file. When you name the project file in the **New Project File** dialog, it automatically sets the **Target File** extension to .EXE and the **Run-Time Library** to Standalone.

❑ To create a library (.LIB file), simply change the target file extension in the Project Tree. Highlight **Target file** in the Project Tree then press the **Add File** button. Then type in the name of the .LIB file you wish to create, including the file extension. Press the **OK** button.



❑ To create a dynamic link library (.DLL), change the target extension in the Project Tree. Highlight **Target file** in the Project Tree then press the **Add File** button. Then type in the name of the .DLL file you wish to create, including the file extension. Press the **OK** button.

> **Tip: Setting the Project's Target Type is equivalent to setting the Application's Destination Type and vice versa.**

.LIB Files

Library files (.LIB) contain procedures and functions which are linked into your executable at *compile* time. To create library files which may be accessed by Clarion, or by any of the other TopSpeed compilers, just set a .LIB file as the target file.

To use procedures and functions from a precompiled .LIB file, you must prototype the procedures and functions called by your program. Prototyping is accomplished by adding a MODULE structure to your application's MAP. To call an external .LIB procedure from "hand coded" source:

1. Add a MODULE structure to your application's MAP.

   The MODULE should reference the external library file. In the Application Generator, you can place this in the *Inside the Global Map* embed point.

2. Add the procedure prototypes:

```
MAP
  MODULE('EXTERNAL.LIB')
    ExtProc(*CSTRING),RAW
    ExtFunc(USHORT, *BYTE[]),USHORT
  END
END
```

   Each prototype specifies the name of the procedure, the data types of any parameters (in parentheses), and the return data type (if any).

   In the example above, the ExtProc procedure ExtProc expects the address (without the length, hence the RAW attribute) of a CSTRING to be passed to it as a parameter.

   The ExtFunc procedure expects the value of a USHORT variable, the address of an array of BYTEs, and will return a USHORT.

3. To specify a different calling convention, add it to the prototype.

   You may use .LIB or .OBJ files created by other compilers.

   Modifying the above examples, the first line below identifies a procedure expecting the C calling convention. The second line identifies a procedure expecting the PASCAL calling convention, which is the Windows standard calling convention:

```
ExtProc(*CSTRING),C,RAW
ExtFunc(USHORT, *BYTE[]),USHORT,PASCAL
```

4. To optionally specify a third party linker's identifier, add it to the prototype.

   Some compilers, most notably 'C' language compilers, add a leading underscore to the name of procedures and functions at compile time. The examples below add the NAME attribute:

```
ExtProc(*CSTRING),C,RAW, NAME('_ExtProc')
ExtFunc(USHORT, *BYTE[]),USHORT,PASCAL,NAME('_ExtFunc')
```

   See the *Language Reference* for more information on MODULE, MAP, PROCEDURE, and prototypes.

.DLL Files

Dynamic Link Libraries (.DLL) contain procedures which are linked to your application at *run-time*. To create dynamic link libraries, just specify .DLL as the target file extension.

To call a .DLL procedure follow the steps outlined for calling a .LIB procedure, above.

Tip: Setting the Project's Target Type is equivalent to setting the Application's Destination Type and vice versa.

Distributing Files

Choosing a Configuration

This section is included to help you decide what kind of target file to specify for your project. See *Setting the Target File*. Also see *Development and Deployment Strategies.*

Clarion produces executable files which you may distribute on a royalty-free basis. The applications you distribute require Windows 3.10, 3.11, 95, or NT.

Clarion executables come in two flavors: .EXE files, and .DLL files. An .EXE file is simply an executable program. A .DLL (Dynamic Link Library) file is executable code that is linked into an .EXE file *at run-time*. This is in contrast to .OBJ and .LIB files which are linked into an .EXE at *compile* time. The most obvious benefit of the .DLL is that it provides a method of modifying .EXE operation without remaking (compiling and linking) the .EXE.

Clarion executables may be distributed in the following four configurations where [x] indicates the 32-bit runtime library for applications running on 32-bit operating systems (Windows 95 or Windows NT); no [x] indicates the 16-bit runtime library for 16-bit operating systems (Windows 3.10, 3.11):

- *.EXE (available only in Professional and Enterprise Editions)

  A one-piece .EXE will usually be larger than an .EXE distributed with .DLLs. However, the one-piece .EXE will probably be smaller than the combined sizes of an .EXE and its associated .DLLs.

  The one-piece .EXE is made as small as possible by Clarion's smart linking process that only links in procedures actually called by the application program (whereas the .DLL contains a fixed set of procedures, whether or not they are actually called by your program).

  A one-piece .EXE cannot have conflicts or problems that arise from linking with the wrong .DLLs at run time.

  Make (compile and link) time for a one-piece .EXE is greater than for an .EXE combined with .DLLs.

Tip: **To make a one-piece .EXE follow the steps described in the *Development and Deployment Strategies* appendix.**

- *.EXE + C5RUN[x].DLL

  Splitting the executables between .EXEs and .DLLs allows for more efficient use of disk space. Many Clarion applications (.EXEs) can share a single C5RUN[x].DLL. Or, a single application suite with several

.EXEs can share a single C5RUN[x].DLL. However, as a developer, you must ensure that your application accesses the correct version of C5RUN[x].DLL.

An example of .DLL usage is the typical accounting system where the .EXE controls the system main menu, and calls system subparts such as Accounts Receivable and Accounts Payable from separate .DLLs. This method of distribution allows for program parts to be sold and maintained separately.

Tip: Splitting executables between .EXEs and .DLLs allows for more efficient use of disk space, but less efficient use of RAM. This is because Windows 95 loads an additional C5RUN[x].DLL into memory for each active Clarion executable, and because the C5RUN[x].DLL contains some procedures your .EXE never calls.

Tip: To make an .EXE + C5RUN[x].DLL: in the Global Options dialog, set Target Type to .EXE and set Run-Time Library to *Standalone*.

- *.EXE + C5RUN[x].DLL + *.DLL$_1$ + ... + *.DLL$_n$

  This configuration offers the same advantages and disadvantages as the .EXE + C5RUN[x].DLL configuration. It is listed here to illustrate that you are not limited to a single .DLL, nor are you limited to Clarion .DLLs. Your Clarion applications may make use of .DLLs compiled from other languages as well as the C5RUN[x].DLL and TopSpeed database driver .DLLs. See the *Application Handbook—Database Drivers* for more information on database drivers.

- *.EXE + *.DLL$_1$ + ... + *.DLL$_n$

  This configuration offers most of the same advantages and disadvantages as the .EXE + C5RUN[x].DLL configuration. It is listed here to illustrate that the C5RUN[x].DLL may be linked into another .DLL. This technique “hides” the C5RUN[x].DLL and ensures that your application will never get the wrong version of C5RUN[x].DLL, because, technically, it isn’t looking for C5RUN[x].DLL.

Tip: To “hide” C5RUN[x].DLL in another DLL, follow the steps described in the *Development and Deployment Strategies* appendix.

Installing and Accessing Your Application's DLLs

If you distribute C5RUN[x].DLL, it must reside in the same directory as the application program, in the Windows\System subdirectory, or in a directory referenced in the system PATH. We recommend that you install C5RUN[x].DLL to the application directory.

Remember, multiple Clarion applications may use the same C5RUN[x].DLL file, thus saving space on the users' hard drive. On the other hand, sharing a single C5RUN[x].DLL raises the possibility of conflicts among applications developed under different versions of Clarion. To avoid possible conflicts, install a separate C5RUN[x].DLL to each application directory, or distribute the application as a single .EXE file, or link the C5RUN[x].DLL into another .DLL that is unique to your application.

The Ship List

For generated applications, Clarion templates automatically create a ship file (.SHP) that contains the names of the files that are needed to run your application. The file is called *application*.SHP and is in the same subdirectory as your .APP file.

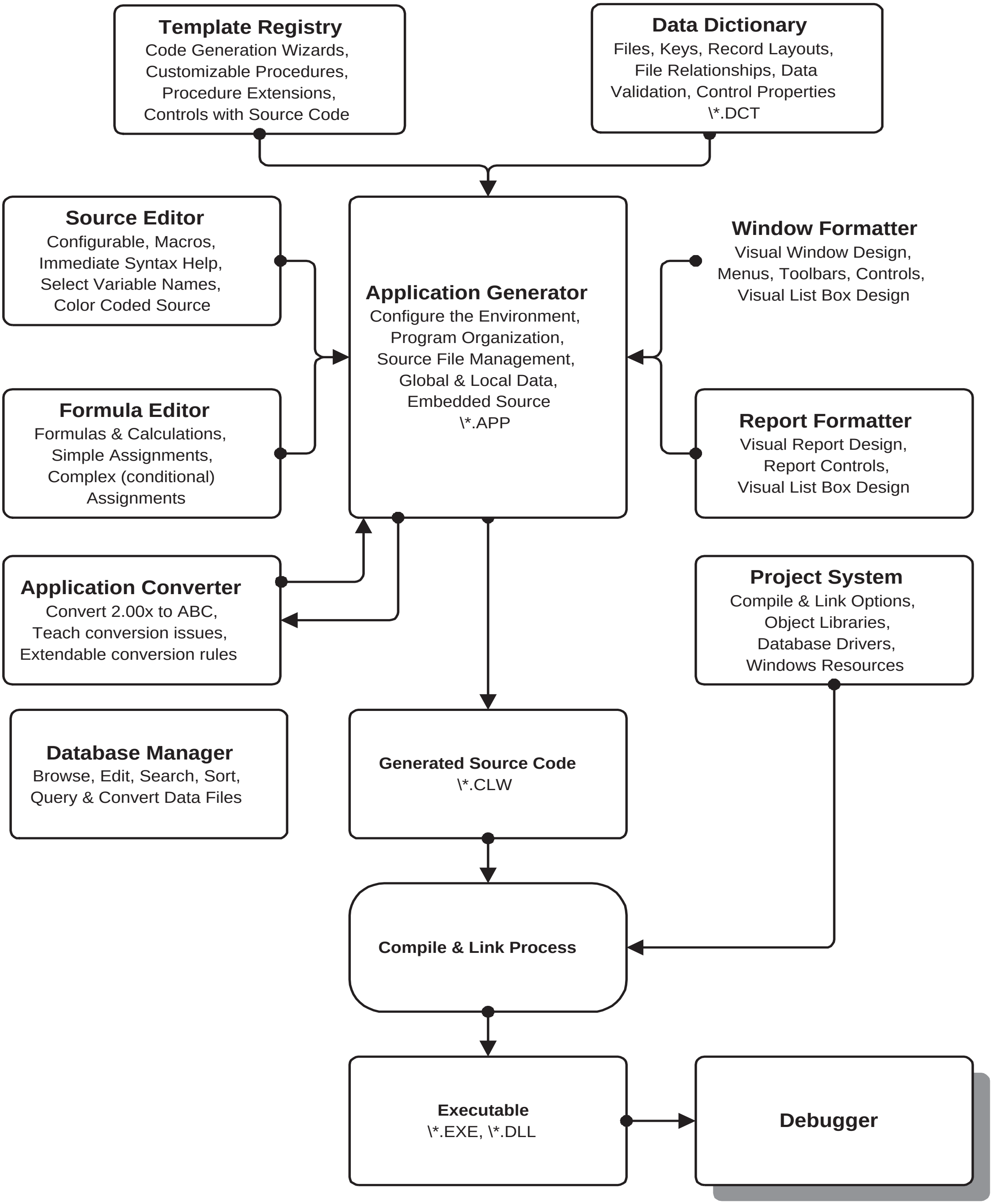
This ship file only includes those files that are visible to the templates. Any DLLs loaded in EMBEDs or INCLUDE files may not be visible to the templates, and may not be in the list.

In the case of external library modules, the .LIB file is also included in the list. Some of the .LIBs (WINDOWS.LIB for example) do not have associated DLLs; however, most do have associated .DLLs that you will need to distribute with your application.

In the case of an external library module generated by Clarion, you must ensure that all files on the shipping list for that LIB/DLL are also included.

Tip: You can modify your application's ship list by embedding text at the *Inside the Shipping List* Global Embed point.

13 - DEBUGGERS


Template Registry
Code Generation Wizards, Customizable Procedures, Procedure Extensions, Controls with Source Code
Data Dictionary
Files, Keys, Record Layouts, File Relationships, Data Validation, Control Properties *.DCT
Source Editor
Configurable, Macros, Immediate Syntax Help, Select Variable Names, Color Coded Source
Application Generator
Configure the Environment, Program Organization, Source File Management, Global & Local Data, Embedded Source *.APP
Window Formatter
Visual Window Design, Menus, Toolbars, Controls, Visual List Box Design
Formula Editor
Formulas & Calculations, Simple Assignments, Complex (conditional) Assignments
Report Formatter
Visual Report Design, Report Controls, Visual List Box Design
Application Converter
Convert 2.00x to ABC, Teach conversion issues, Extendable conversion rules
Project System
Compile & Link Options, Object Libraries, Database Drivers, Windows Resources
Database Manager
Browse, Edit, Search, Sort, Query & Convert Data Files
Generated Source Code
*.CLW
Compile & Link Process
Executable
*.EXE, *.DLL
Debugger

Overview

Clarion ships two debuggers: a 16-bit debugger for 16-bit applications and a 32-bit debugger for 32-bit applications. Both are powerful tools for finding and diagnosing errors in your applications. You can examine source code and data as your program executes, and exercise complete control over your program's execution.

This chapter tells you how to:

- Prepare your projects for debugging.
- Start the debugger.
- Customize the debugger's operation to your work environment.
- Monitor your program's execution and check its state at specific points by setting breakpoints and watch expressions.

The Debugging Process

The debuggers are very flexible, quite complex, and there are many windows, options, and features available. This overview of the debugging process suggests a general sequence of steps that introduces you to the most important features of the debuggers with the least amount of confusion. Keep this sequence in mind as you explore the debuggers.

1. Shut down other applications, then start the debugger.

   This offers two benefits. First, more system resources are available to your application and the debugger. Second, you won't lose data from other active applications if a system crash occurs during the debugging process.

2. Load only the source files you need to debug.

   Each source file you select becomes a child window in the debugger. The fewer source files you select, the less clutter you have on your debugger screen, and the less overhead the debugger must manage.

3. Set Debug Options.

   Take a few minutes to read *Setting Debugger Options*. Options such as Clarion Soft Mode, Autotile, Clean Desktop, Debugger on Top, Global Find Text, and others can make the (16-bit) debugger easier to read and work with.

> **Tip: We recommend Clarion Soft Mode for most 16-bit projects. However, under Windows 95, you must use Hard Mode. In Hard Mode, *all* other system activity is suspended while the debugger is active. This means the desktop is not redrawn, which can be confusing if you are not expecting it. It also means correct debugger configuration is critical to successful debugging. See *Setting Debugger Options*.**

4. Set a breakpoint.
5. Run your application (the debuggee) with **Go** or **Step** commands.
6. Select and arrange the debugger windows.

   Many of the debugger windows will be empty until your application stops at a breakpoint. Once your application stops, and the windows are populated, they will be more meaningful and easier to understand and work with. Iconize or close the windows you don't need to see.
7. Set breakpoints, set watch expressions, and change variable values.
8. Run your application with **Go** or **Step** commands.
9. Repeat steps 7 and 8 as needed.
10. Exit your application (debuggee).

    It is very important that you exit the debuggee program *before* you exit the 16-bit debugger. Exiting the debugger while the debuggee is still active can cause system crashes.

Preparing Your Projects for Debugging

The Project System lets you set the debug options for your application in the **Global Options** dialog. To make your executable (.EXE or .DLL) suitable for debugging:

Generate Global Debug Information

1. Create your project file, and make it the current project (the *Project System* chapter explains how).
2. Choose **Project ➤ Edit** to view the **Project Editor** dialog.
3. Select the top level of the tree, which contains the name of the project, then press the **Properties** button.
4. When the **Global Options** dialog appears, select the **Debug** tab, then choose **Full** from the **Debug Mode** drop list.

   For 16-bit programs, the compiler generates debug information into separate .DBD files. By default the .DBD files are in the \OBJ folder.

For 32-bit programs, the compiler generates the debug information into the .EXE or .DLL.

5. Optionally check the **Line Numbers** box.

   Line numbers are automatically available to the Clarion debugger, however, if you are using another debugger, checking this box will make line numbers available to it.

6. Press the **OK** button to close the **Global Options** dialog, then the **Project Editor** dialog.

7. Press the *Make* button on the toolbar to compile and link the application.

   The application now includes the information the debugger needs.

Generate Local Debug Information

You can also turn on debugging information for a *single* module in the project. This reduces the overhead for the debugger. To do so, follow the steps above for **Global Options**, except choose **None** from the **Debug Mode** drop list. Then follow the steps below:

1. Choose **Project ➤ Edit** to view the **Project Editor** dialog.

2. Select only the source module you need to debug, then press the **Properties** button.

3. When the **Compile Options** dialog appears, choose **Full** from the **Debug Mode** drop list.

4. Press the **OK** button to close the **Project Editor** dialog and the **Compile Options** dialog.

5. Press the **Make** button on the toolbar to compile and link the application.

   This includes debug information for that module only.

Debug Configuration

You can tell the 16-bit debugger to remember and apply settings (breakpoints, window coordinates, watch variables, etc) from the previous debug session. To remember and apply the previous sesssion settings:

1. Choose **Project ➤ Auto Resume on Debug**.

Locating Page Faults (GPF)

The debuggers can identify a specific line of code where your program is crashing. For 16-bit programs, simply start the debugger, then start the debuggee and let it crash. The debugger displays the line of code that caused the crash. For 32-bit programs, make the debugger the system debugger (see Setting Debugger Options), then run your program. When it crashes, select debug to invoke the 32-bit debugger to take you to the offending line.

Starting the Debugger from a Popup Menu

When you start the debugger from Windows, you should take steps to set the proper working directory—usually the directory that contains the debuggee. For example, create a Windows 95 shortcut that sets the “Start in” directory.

Or you can use Regedit to make Windows 95 registry entries to start the debugger by RIGHT-CLICKING on the .exe to debug. Here are some example registry entries that start the debugger with the appropriate working directory:

```
[HKEY_CLASSES_ROOT\exefile\shell\16bit Debugger]
[HKEY_CLASSES_ROOT\exefile\shell\16bit Debugger\command]
@=“c5db.exe c:\\clarion5\\bin\\clarion5.red c:\\clarion5\\bin\clarion5.ini %1”

[HKEY_CLASSES_ROOT\exefile\shell\32bit Debugger]
[HKEY_CLASSES_ROOT\exefile\shell\32bit Debugger\command]
@=“c5dbx.exe c:\\clarion5\\bin\\clarion5.red c:\\clarion5\\bin\clarion5.ini
%1”

[HKEY_CLASSES_ROOT\exefile\shell\Restart Debugger]
[HKEY_CLASSES_ROOT\exefile\shell\Restart Debugger\command]
@=“c5db.exe /r”
```

16-bit Debugger

Under Windows 95, the 16-bit debugger only operates in hard mode. In hard mode, *all* other system activity is suspended while the debugger is active; therefore, proper debugger configuration is critical for successful debugging. The tutorial shows how to configure the 16-bit debugger for best results under hard mode.

The 16-bit debugger is not compatible with Windows NT. To debug under Windows NT, develop and debug in 32-bit mode, then return to 16-bit mode.

Tutorial

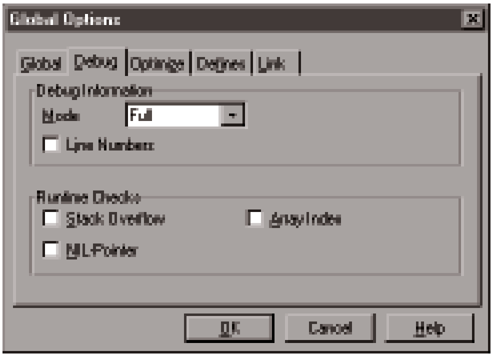
> **Note: This tutorial uses the Clarion Template chain. You must register these templates (CW.tpl) before doing the tutorial.**

Prepare Your Program for Debugging

In this tutorial you will debug the Orders application, installed by default as \EXAMPLES\TUTOR\ORDERS.APP. We have deliberately created a bug in this application. To begin, start Clarion and open the Orders application.

To debug your program with the 16-bit debugger, you must use the 16-bit compiler and tell the compiler to generate debug information. These two settings (16-bit with debug information) are the default for Clarion applications. However, you should confirm these settings before you begin.

1. Open the Orders application with Clarion.
2. Press the **Project** button to open the **Project Editor**.
3. Press the **Properties** button to open the **Global Options** dialog.
4. On the **Global** tab, select *Windows - 16 bit* from the**Target OS** drop-list.
5. On the **Debug** tab, select *Full* from the **Mode** drop-down list.



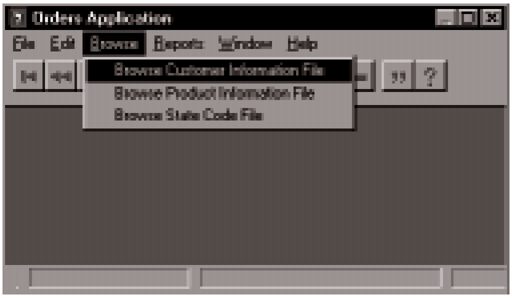
Mode: Full
□ Line Numbers
□ Stack Overflow
□ Array Index
□ NIL Pointer

6. Press **OK** twice to return to the application tree.

Identify the Bug

We have deliberately created a bug in the Orders application. To see it:

1. Press the button to make and run the Orders program.
2. From the Orders program menu, choose **Browse ➤ Browse Customer Information File**.


Orders Application
File Edit Browse Reports Window Help
Browse Customer Information File
Browse Product Information File
Browse State Code File

This opens the customer browse list.

3. With the customer list open, press the ▶ button to scroll down one row.

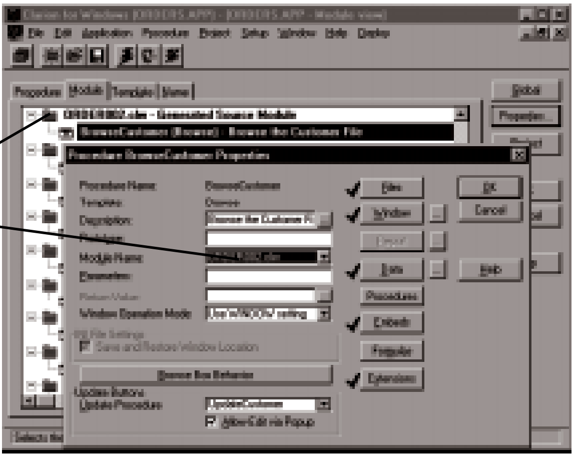
The list doesn't scroll! This bug only occurs with next record and next page (downward scrolling). Prior record, prior page, first page, and last page, all work fine.

You've identified the bug: it is in the BrowseCustomers procedure and it only occurs when scrolling downward.

4. Shut down the Orders program.

The Clarion environment gains focus.

5. Select the **Module** tab (or open the **Procedure Properties** dialog for the BrowseCustomer procedure).



Two ways to identify the module name.

Procedure BrowseCustomer Properties

Procedure Name: BrowseCustomer

Template: Browse

Description: Browse the Customer F

Window Operation Mode: Use WINDOW setting

Save and Restore Window Location

Update Procedure: UpdateCustomer

Allow Edit via Popup

6. Note the name of the source module that contains the BrowseCustomer procedure—ORDER002.

   You can have the debugger load all the source modules for an application, but is more efficient to debug only the affected modules.

Start the Debugger

To start the debugger:

1. From the Clarion environment, press the button to start the debugger.

   You may want to shut down other applications to conserve resources because you will be running the Clarion environment, the debugger, plus the Orders program in addition to any others.

> **Tip:** **You can run the debugger independent of the development environment to save even more resources: run C5DB.EXE, then choose File ➤ File to Debug.**

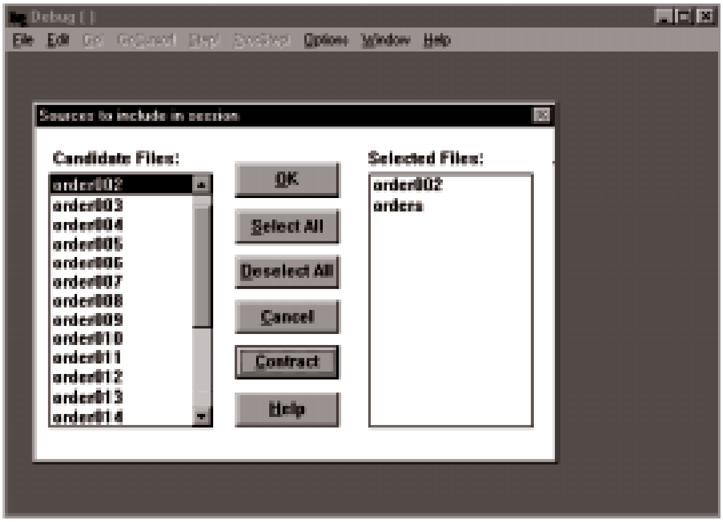
   The Clarion environment verifies all files are current, then starts the debugger. The debugger opens the **Sources to include in session** dialog.

2. Press the **Expand >>** button to show the source modules selected for debugging.

3. Select ORDER002 and ORDERS.

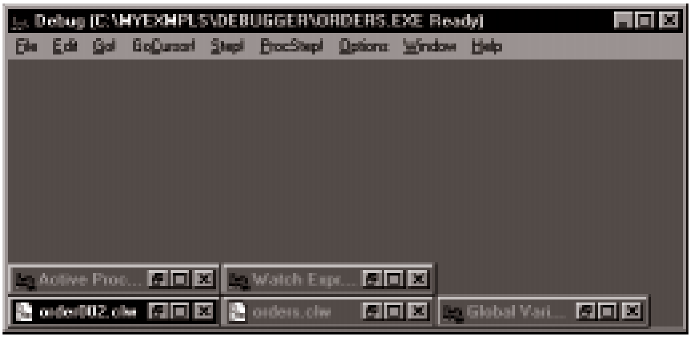
   ORDERS.CLW contains the Program that declares global data such as record buffers. You should usually include the Program module so you can use the debugger to review and manipulate your program's global data. ORDER002 contains the BrowseCustomer procedure.

   The debugger "remembers" the source modules between sessions.



4. Press **OK**.

   The debugger opens several iconized windows.



Debug [C:\MYEXMPLS\DEBUGGER\ORDERS.EXE Ready]
File Edit Go! GoCursor! Step! ProcStep! Options Window Help
Active Proc...
Watch Expr...
order002.clw
orders.clw
Global Vari...

Configure the Debugger

To successfully use the 16-bit debugger in hard mode you should do the following.

1. Maximize the debugger.

   You will usually want to see several debugger windows at once, so you need as much screen space as possible; besides, all other activity is suspended while the debugger is active, so you can't access other applications that are sharing screen space with the debugger.

2. Choose **Options ➤ Setup**.

   This opens the debugger's **Setup** dialog.

3. Check the **Bring debugger to the top on hard mode break** box.

   When the debugger hits a breakpoint, it automatically gains focus and displays the active source code line.

4. Check the **Smart single stepping** box.

   When enabled, STEPping on a line with a procedure call automatically loads the debug information for the called procedure, if available.

5. Press **OK**.

Arrange the Debugger Windows

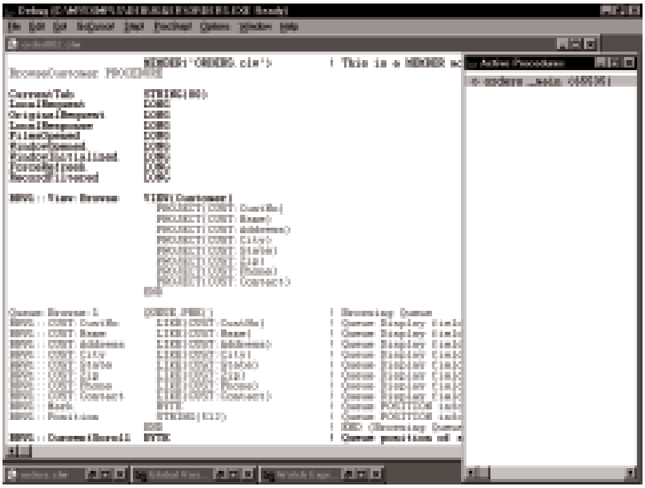
Another key to successful debugging is to optimally arrange the debugger windows so you can quickly access pertinent information. You should restore only those windows you need to see often, and you should arrange the windows so they are easy for you to work with.

For this tutorial, you need to see the ORDER002.CLW source code window, the Global Variables window, and the Active Procedures window.

1. Restore the ORDER002.CLW window.
2. Restore the Global Variables window.
3. Restore the Active Procedures window.
4. Reposition and resize the windows so they are easy for you to work with.

   The debugger "remembers" the window positions between sessions.

   We recommend a configuration that shows lots of source code, shows as many global and local variables as possible, and provides a small overlap for easy "switching" between windows.



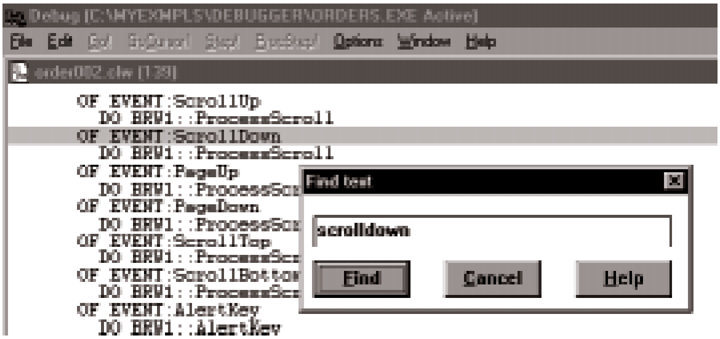
```
                        MEMBER('ORDERS.clw')          ! This is a MEMBER mo
BrowseCustomer PROCEDURE

CurrentTab              STRING(80)
LocalRequest            LONG
OriginalRequest         LONG
LocalResponse           LONG
FilesOpened             LONG
WindowOpened            LONG
WindowInitialized       LONG
ForceRefresh            LONG
RecordFiltered          LONG

BRW1::View:Browse       VIEW(Customer)
                          PROJECT(CUST:CustNo)
                          PROJECT(CUST:Name)
                          PROJECT(CUST:Address)
                          PROJECT(CUST:City)
                          PROJECT(CUST:State)
                          PROJECT(CUST:Zip)
                          PROJECT(CUST:Phone)
                          PROJECT(CUST:Contact)
                        END

Queue:Browse:1          QUEUE,PRE()                   ! Browsing Queue
BRW1::CUST:CustNo         LIKE(CUST:CustNo)           ! Queue Display fiel
BRW1::CUST:Name           LIKE(CUST:Name)             ! Queue Display fiel
BRW1::CUST:Address        LIKE(CUST:Address)          ! Queue Display fiel
BRW1::CUST:City           LIKE(CUST:City)             ! Queue Display fiel
BRW1::CUST:State          LIKE(CUST:State)            ! Queue Display fiel
BRW1::CUST:Zip            LIKE(CUST:Zip)              ! Queue Display fiel
BRW1::CUST:Phone          LIKE(CUST:Phone)            ! Queue Display fiel
BRW1::CUST:Contact        LIKE(CUST:Contact)          ! Queue Display fiel
BRW1::Mark                BYTE                        ! Queue POSITION info
BRW1::Position            STRING(512)                 ! Queue POSITION info
                        END                           ! END (Browsing Queue
BRW1::CurrentScroll     BYTE                          ! Queue position of s
```

Set BreakPoints

You want to suspend (or break) the program at the point the bug occurs, then look for the cause of the problem, usually in the form of incorrect variable values or incorrect execution sequence. Recall the bug is associated with scrolling downward. To set the breakpoint:

1. Choose **Window ➤ Order002** to access the BrowseCustomers source window.
2. Choose **Edit ➤ Find Text** (or press CTRL+F) to open the **Find Text** dialog.
3. In the **Find Text** dialog, type *scrolldown*, then press the **Find** button.

Tip: The search is not case sensitive.


Debug [C:\MYEXMPLS\DEBUGGER\ORDERS.EXE Active]
File Edit Go! Step! Options Window Help
order002.clw [139]
Find text
Find
Cancel
Help

```
OF EVENT:ScrollUp
  DO BRW1::ProcessScroll
OF EVENT:ScrollDown
  DO BRW1::ProcessScroll
OF EVENT:PageUp
  DO BRW1::ProcessScr
OF EVENT:PageDown
  DO BRW1::ProcessScr
OF EVENT:ScrollTop
  DO BRW1::ProcessScr
OF EVENT:ScrollBottom
  DO BRW1::ProcessScr
OF EVENT:AlertKey
  DO BRW1::AlertKey
```

scrolldown

This finds the EVENT:ScrollDown where the BRW1::ProcessScroll routine is called. This routine is a likely location for the breakpoint you need.

4. Choose **Edit ➤ Find Text** (or press CTRL+F).
5. In the **Find Text** dialog, type *BRW1::ProcessScroll routine*, then press the **Find** button.

   This finds the BRW1::ProcessScroll routine.
6. CLICK on the first executable statement after the routine label:

   ```
   IF BRW1::RecordCount
   ```

Note: Although you can set a breakpoint on routine labels, you may get unexpected results. We recommend setting breakpoints only on executable statements.

7. Press the INS key to set an unconditional breakpoint on this statement.

   The debugger displays the break statement in bright pink.

Debug the Program

The debugger is now set to suspend program execution at the first statement in the BRW1::ProcessScroll routine. To debug the program:

1. Choose **Go!** or type G to start the Orders program.

   The Orders program begins execution. Its main window opens on top of the debugger.
2. Choose **Browse ➤ Browse Customer Information file**.

   This opens the customer browse list.
3. With the customer list open, press the ▶ button to scroll down one row.

   The debugger detects the breakpoint, suspends the Orders program, and displays the next source statement to execute with a green background.
4. Choose **Step!** or type S to execute the next statement.

The debugger executes a single statement, then displays the next source statement to execute with a green background.

5. Repeat step 4 until the first statement in the BRW1::ScrollOne routine executes:

```
 IF BRW1::CurrentEvent = Event:ScrollUp AND BRW1::CurrentChoice > 1
   BRW1::CurrentChoice -= 1
   EXIT
ELSIF BRW1::CurrentEvent = Event:ScrollDown AND BRW1::CurrentChoice < BRW1::RecordCount
   BRW1::CurrentChoice += 1
   EXIT
END
```

Notice that none of the conditional statements execute. This is because neither the primary IF condition nor the ELSIF condition are true.

Examine Local Variable Values

You can surmise that the above IF condition must be true in order for the customer list to scroll *up* one row, and the ELSIF condition must be true in order for the customer list to scroll *down* one row. You can examine (and modify) the values of the variables within the ELSIF condition to test this theory—BRW1::CurrentEvent must equal Event:ScrollDown, and BRW1::CurrentChoice must be less than BRW1::RecordCount.

To examine and edit the variable values:

1. Choose **Go!** or type G to continue executing the Orders program.

   The Orders program continues execution where it left off; however, it has not regained focus, so you cannot see it.

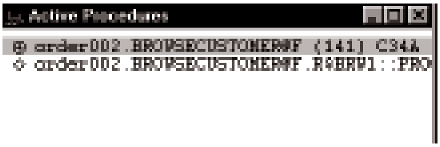
2. Minimize the debugger to reveal the Orders program.

   The Orders program window gains focus.

3. Again, press the ▶ button to scroll down one row.

   Again, the debugger detects the breakpoint, suspends the Orders program, and displays the next source statement to execute with a green background.

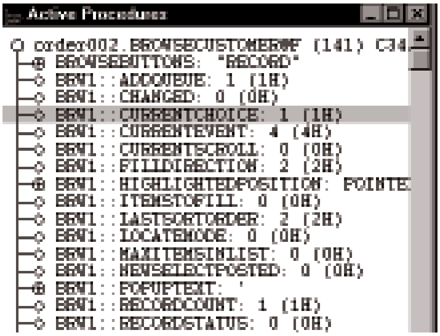
4. CLICK in the Active Procedures window, or choose **Window ➤ Active Procedures**.



Active Procedures

```
⊕ order002.BROWSECUSTOMERWF (141) C34A
◇ order002.BROWSECUSTOMERWF.R4BRW1::PRO
```

5. In the Active Procedures window, CLICK on the (+) to expand the list of local variables.



```
○ order002.BROWSECUSTOMERWF (141) C34
├⊕ BROWSEBUTTONS: "RECORD"
├◇ BRW1::ADDQUEUE: 1 (1H)
├◇ BRW1::CHANGED: 0 (0H)
├◇ BRW1::CURRENTCHOICE: 1 (1H)
├◇ BRW1::CURRENTEVENT: 4 (4H)
├◇ BRW1::CURRENTSCROLL: 0 (0H)
├◇ BRW1::FILLDIRECTION: 2 (2H)
├⊕ BRW1::HIGHLIGHTEDPOSITION: POINTE
├◇ BRW1::ITEMSTOFILL: 0 (0H)
├◇ BRW1::LASTSORTORDER: 2 (2H)
├◇ BRW1::LOCATEMODE: 0 (0H)
├◇ BRW1::MAXITEMSINLIST: 0 (0H)
├◇ BRW1::NEWSELECTPOSTED: 0 (0H)
├⊕ BRW1::POPUPTEXT: '
├◇ BRW1::RECORDCOUNT: 1 (1H)
├◇ BRW1::RECORDSTATUS: 0 (0H)
```

6. Choose **Edit ➤ Find Text** (or press CTRL+F) to open the **Find Text** dialog.

7. In the **Find Text** dialog, type *BRW1::CurrentChoice*, then press the **Find** button.

   Find only searches the displayed text, so you should expand the list before searching.

   The Active Procedures window shows the label of the variable on the left and its value on the right. The value of BRW1::CurrentChoice is 1. Several rows down, you can see the value of BRW1::RecordCount is also 1. These values would cause the ELSIF condition to fail because the condition requires BRW1::CurrentChoice to be less than BRW1::RecordCount, when in fact, they are equal.

   Again, you can surmise that BRW1::RecordCount is the offender because you certainly saw more than one record in the customer browse list. Test this theory by changing the value of BRW1::RecordCount to a larger number, for example, 20.

Edit Local Variable Values

1. In the Active Procedures window, CLICK on BRW1::RecordCount to select it.

2. Choose **Edit ➤ Edit** or press F2 to open the **Edit Variable** dialog.

Tip: You can only edit fully expanded variables; that is, if there is a + sign beside the variable, you cannot edit it's value. click on the + sign to expand, then edit.

3. In the **Edit Variable** dialog, type *20*, then press **OK**.

   The debugger changes the value of BRW1::RecordCount to 20. Now you can continue execution of the Orders program to see if it scrolls properly.

4. Choose **Step!** or type S to execute the next statement.

   The debugger displays the next source statement to execute with a green background.

5. Repeat step *4* until the first statement in the BRW1::ScrollOne routine executes.

Notice that the ELSIF conditional statements do execute. This is because the ELSIF condition is now true: BRW1::CurrentChoice is less than BRW1::RecordCount.

6. Choose **Go!** or type G to continue executing the Orders program.

   The Orders program continues execution where it left off; however, it has not regained focus, so you cannot see it.

7. Minimize the debugger to reveal the Orders program.

   The customer list scrolls correctly! This confirms the theory that BRW1::RecordCount contains an incorrect value at this point in the program.

Examine Global Variable Values

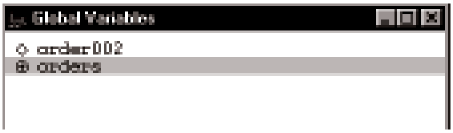
Often you will want to examine global variables and record buffer values.

To examine and edit global variable values:

1. Again, press the ► button to scroll down one row.

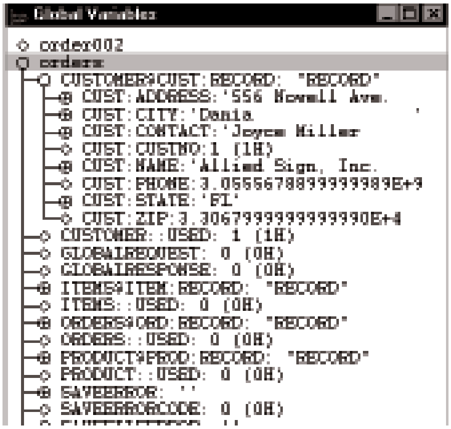
   Again, the debugger detects the breakpoint, suspends the Orders program, and displays the next source statement to execute with a green background.

2. CLICK in the Global Variables window, or choose **Window ➤ Global Variables**.



```
◇ order002
⊕ orders
```

3. In the Global Variables window, CLICK on the (+) to expand the list of global variables.



```
◇ order002
○ orders
 ├─○ CUSTOMER$CUST:RECORD:  "RECORD"
 │  ├─⊕ CUST:ADDRESS:'556 Nowell Ave.
 │  ├─⊕ CUST:CITY:'Dania                 '
 │  ├─⊕ CUST:CONTACT:'Joyce Miller
 │  ├─◇ CUST:CUSTNO:1 (1H)
 │  ├─⊕ CUST:NAME:'Allied Sign, Inc.
 │  ├─◇ CUST:PHONE:3.0555678899999989E+9
 │  ├─⊕ CUST:STATE:'FL'
 │  └─◇ CUST:ZIP:3.3067999999999990E+4
 ├─◇ CUSTOMER::USED: 1 (1H)
 ├─◇ GLOBALREQUEST: 0 (0H)
 ├─◇ GLOBALRESPONSE: 0 (0H)
 ├─⊕ ITEMS$ITEM:RECORD:  "RECORD"
 ├─◇ ITEMS::USED: 0 (0H)
 ├─⊕ ORDERS$ORD:RECORD:  "RECORD"
 ├─◇ ORDERS::USED: 0 (0H)
 ├─⊕ PRODUCT$PROD:RECORD:  "RECORD"
 ├─◇ PRODUCT::USED: 0 (0H)
 ├─⊕ SAVEERROR: ''
 ├─◇ SAVEERRORCODE: 0 (0H)
```

The Global Variables window shows global variables and record buffers. The labels are on the left and the values are on the right.

4. Choose **Edit ➤ Find Text** (or press CTRL+F) to open the **Find Text** dialog.

5. In the **Find Text** dialog, type *CONTACT*, then press the **Find** button.

   Find only searches the displayed text, so you should expand the list before searching.

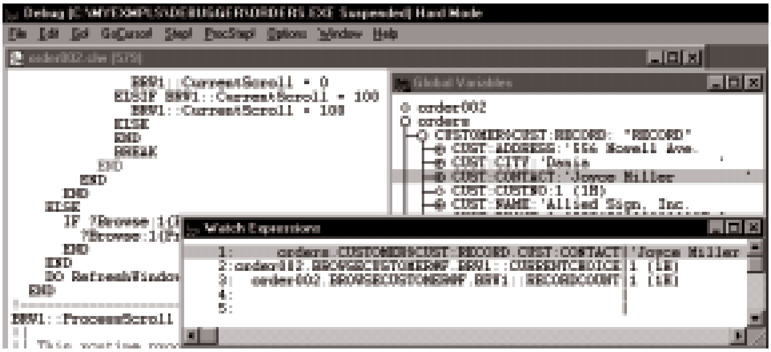
> **Tip:** **You may edit global variables the same way you edited local variables.**

Set a Watch Expression

The Watch Expressions window lets you pick specific variables (local, global, or both) to monitor during the debugging process. That is, rather than searching through the Active Procedures window and Global Variables window with each debugging cycle, you can add the variables to the Watch Expressions window and examine all the pertinent values in one place.

1. In the Global Variables window, DOUBLE-CLICK the CONTACT variable, then choose **Copy Variable to Watch** to add it to the Watch Expressions window.

   The debugger adds the variable to the Watch Expressions window. The illustration shows both local and global variables in the Watch Expressions window.


Debug [C:\MYEXMPLS\DEBUGGER\ORDERS.EXE Suspended] Hard Mode
File Edit Go! GoCursor Step! ProcStep! Options Window Help
order002.clw [570]
Global Variables
Watch Expressions

```
          BRW1::CurrentScroll = 0
        ELSIF BRW1::CurrentScroll = 100
          BRW1::CurrentScroll = 100
        ELSE
        END
        BREAK
      END
    END
  END
ELSE
  IF ?Browse:1{
    ?Browse:1{P
  END
END
DO RefreshWindow
END
!-------------------
BRW1::ProcessScroll
!|
!| This routine proc
```

```
○ order002
○ orders
  └○ CUSTOMER:CUST:RECORD: "RECORD"
     ├◎ CUST:ADDRESS:'556 Howell Ave.
     ├◎ CUST:CITY:'Denis
     ├◎ CUST:CONTACT:'Joyce Miller
     ├○ CUST:CUSTNO:1 (1H)
     ├◎ CUST:NAME:'Allied Sign, Inc.
```

```
1:      orders.CUSTOMER:CUST:RECORD.CUST:CONTACT |'Joyce Miller
2:order002.BROWSECUSTOMERWF.BRW1::CURRENTCHOICE |1 (1H)
3:    order002.BROWSECUSTOMERWF.BRW1::RECORDCOUNT |1 (1H)
4:
5:
```

2. Choose **Go!** or type G to continue executing the Orders program.

   When the debugger detects a breakpoint, it suspends the Orders program, and displays the next source statement to execute with a green background.

3. CLICK in the Watch Expressions window, or choose **Window ➤ Watch Expressions**.

   The Watch Expressions window shows the selected variable values. The labels are on the left and the values are on the right.

> **Tip: You may set breakpoints that suspend the program based on the evaluation of a Watch expression (see *Setting BreakPoints—Conditional Breakpoints*).**

Close the Debugger

When you shut down the debugger, you should first shut down the debuggee to preserve Windows resources.

1. In the Orders program, choose **File ➤ Exit**.
2. In the debugger, choose **File ➤ Exit**.

Starting the Debugger

The 16-bit debugger runs as a separate application, but you can start it either from the development environment, or directly from Windows. Starting from Windows, with the development environment unloaded, means more system resources are available for your application and the debugger.

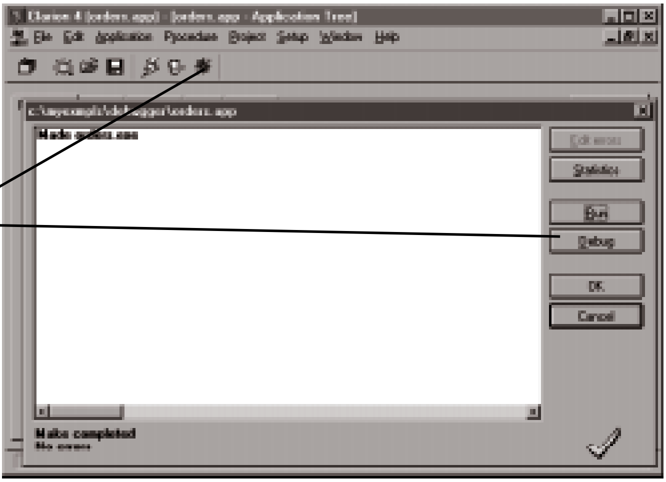
Start the debugger from the development environment

1. Choose **Project ➤ Debug** *or* press the button on the toolbar.

   The development environment checks the project information to determine if your application is 16-bit or 32-bit, and starts the corresponding debugger.

   or...

2. Compile and link your application by pressing the button, then, with the compile results dialog still open, press the **Debug** button.


Clarion 4 [orders.app] - [orders.app - Application Tree]
File Edit Application Procedure Project Setup Window Help
c:\myexmpls\debugger\orders.app
Edit errors
Statistics
Run
Debug
OK
Cancel

Make orders.exe

Make completed
No errors

Two ways to start the debugger.

Starting the debugger from the environment sets the working directory to the debuggee program's directory. This ensures the debugger will use any applicable configuration (.INI) files and redirection (.RED) files.

Start the debugger from Windows

When you start the debugger from Windows, you should take steps to set the proper working directory—usually the directory that contains the debuggee. For example, create a Windows 95 shortcut that sets the "Start in" directory.

1. With your preferred method (Start menu, Explorer, Program Manager, etc.) start C5DB.EXE.
2. Choose **File ➤ File to Debug**, then choose the .EXE file to debug in the Windows file dialog.

   You *can* load the debugger, then debug a program which was already running *before* you loaded the debugger. This is useful for situations where the program under development unexpectedly "misbehaves," but hasn't yet produced a fatal error.

   Start the debugger as usual, then choose **File ➤ File to Debug**. Choose the .EXE file for the running program from the Windows file dialog. The debugger asks you to confirm that you wish to debug a running program.

Tip: When debugging, run only the debugger and the debuggee programs. By doing so, you won't lose data in other applications if a crash occurs during the debugging process.

Loading the Source Files

When you run the debugger, you must select the source code files to debug. For this purpose, the **Sources to include in session** dialog automatically appears when you start the debugger.

To load the source files when the debugger starts

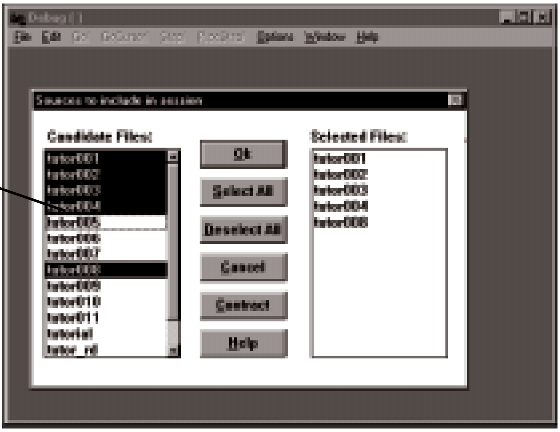
1. Select the source code files in the **Sources to include in session** dialog by CLICKING on them.

   The debugger remembers the files you select between debug sessions. We recommend selecting the main souce file (*appname*.CLW) so that global variables are available in the Global Variables window during the debugging process.
2. Optionally, press the **Select All** button to include all project source files.

   Generally we don't recommend loading all source files because it results in extra overhead for the debugger and visual clutter.
3. Press the **Expand/Contract** button for a list of only the selected source files.

4. Press the **OK** button.

   The debugger windows appear.



The Debugger prompts you to select the source code files to debug.

Setting Debugger Options

Because of its broad range of flexibility, the debugger is quite complex, so setting some basic options prior to using the debugger can pay off in reduced learning curves. In particular, we recommend enabling **Clarion Soft Mode** for most projects developed under Windows 3.1.

The debugger **Options** menu provides several toggles which can help fine tune the way you debug your project.

Soft Mode

Toggles hard and soft mode debugging. Soft Mode is unavailable under 32-bit Windows (95 and NT).

In *soft mode*, when the program being debugged is suspended in the debugger, part of the debugger attempts to simulate the behavior of the program being analyzed (debugee).

In *hard mode*, when the program being debugged is suspended in the debugger, the only window to operate is the debugger. *All other activity is suspended*. One consequence of this is that *the desktop is not redrawn*. Another is that *other* active applications are inaccessible until the debugger returns control to the debuggee.

Tip: When working in Hard mode, type D to bring the debugger to the top.

Clarion Soft Mode

The debugger will use part of the run-time library to simulate the behavior of the program being debugged. *This is the* *recommended mode for most projects.* Clarion Soft Mode is unavailable under 32-bit Windows (95 and NT).

Extended Stack Trace

The debugger shows information about procedures when no debug information is available. A disassembly window opens, containing the relevant segment.

Disassembly On

The Disassembly Windowdisplays assembler code. It “shadows” the active source window. When you select a line of source, the cursor in the Disassembly Window moves to the corresponding line of assembler code.

This menu is a toggle option. If the Disassembly Window is closed when you turn on the option, you can open it by DOUBLE-CLICKING on a source line, then pressing **Cancel** in the **Breakpoint** dialog.

Assembly Single Step

Toggles step mode for assembler breakpoints. When execution reaches an assembler breakpoint, step mode is set on. When execution reaches a source breakpoint, it turns off.

Control Panel

Displays a toolbox window with buttons corresponding to the four **Go!** commands. The next time the program being debugged is suspended, the control panel receives focus.

Tip: When debugging in hard mode, when you activate the main debugger window, you cannot access the control panel, so don’t use the control panel for hard mode debugging.

Setup

Opens the **Setup** dialog. See *Debugger Setup Options*.

Debugger Setup Options

Access the debugger **Setup** dialog using the **Options** ➤ **Setup** command. The dialog provides the following options.

Ignore Dll’s Instructs the debugger to ignore debug information in .DLL files. This reduces the start-up time for the debugger.

Disable Kernel messages

If you are running the Debug version of Windows (available in the Microsoft Windows 3.1 SDK), the debugger will automatically trap error messages posted by the kernel (one of the three main dynamic link libraries used by Windows). You can locate such errors with the **Find Last Error** command.

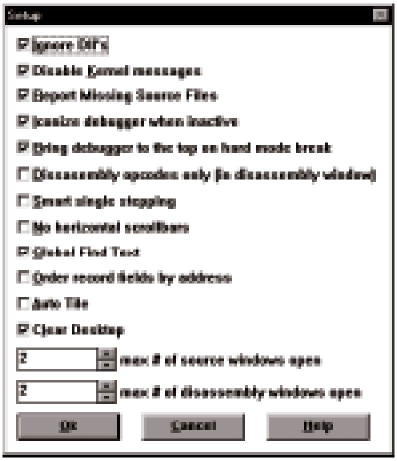
If you are *not* using the AUX device to report messages, add the line OutputTo=NUL in the [DEBUG] section of your SYSTEM.INI file.

Report Missing Source Files
The debugger automatically prompts for source code files it cannot locate.

Iconize debugger when inactive
Automatically iconizes the debugger when the program being debugged is active.

Bring debugger to the top on hard mode break
The debugger appears on top of any other open windows when active; relevant for hard mode debugging only.

Tip: When working in Hard mode, type D to bring the debugger to the top.



Setup

☑ Ignore DIFs
☑ Disable Kernel messages
☑ Report Missing Source Files
☑ Iconize debugger when inactive
☑ Bring debugger to the top on hard mode break
☐ Disassembly opcodes only (in disassembly window)
☐ Smart single stepping
☐ No horizontal scrollbars
☑ Global Find Text
☐ Order record fields by address
☐ Auto Tile
☑ Clear Desktop
2 max # of source windows open
2 max # of disassembly windows open

Disassembly opcodes only (in disassembly window)
The disassembly window contains only opcodes, eliminating the space taken up by binary codes.

Smart single stepping When enabled, single stepping on a line with a procedure call will load the debug information for the target procedure, if available. This option extends to .DLL's with debug information.

No horizontal scrollbars
Hides the horizontal debugger scroll bars.

Global Find Text
When disabled, each source window "remembers" its own search text string. When enabled, the default search text will be the same as the last search, regardless of the window.

Order record fields by address
When enabled, it orders the RECORD variables by memory address.

Auto Tile

Tiles the open debugger windows.

Clear Desktop

When enabled, it minimizes all other running applications (other than the debuggee) when the debugger activates. This has no effect under hard mode.

Max # of source windows open

Specifies the maximum number of source windows the debugger will open at one time.

Max # of disassembly windows open

Specifies the maximum number of disassembly windows the debugger will open at one time.

Additional Debugger Options

In addition to the normal debugging window and setup options, you can activate special modes and options from these menu commands:

Redirection

To use a redirection file other than CLARION5.RED with the debugger, choose **File ➤ Load Redirection**. The Redirection file helps the debugger locate files such as *.DBD, and *.CLW. See *Clarion's Development Environment—Search Paths—the Redirection File*.

Active DLL's

To add a related dynamic link library (*.DLL) to the debug session, choose **File ➤ Debug Active DLL**. Choose a file from the **Active Module** dialog. This option is available for hard mode debugging only.

Sleep

To set the debugger into sleep mode, in which it waits for a general protection fault (GPF), CTRL+ALT+SYSRQ, or an INTERRUPT(INT3), choose **File ➤ Sleeper Mode**. This option is available for hard mode debugging only.

You can start the debugger in sleep mode from a DOS prompt by adding `/S` to the command line.

Tip: If the program being debugged goes into an infinite loop, CTRL+ALT+SYSRQ will break it.

Restart

To start a debug session with the watch expressions and breakpoints from a previously terminated session, choose **File ➤ Restart**. This option is available, *provided* the source code has not changed. See also *Reapply Settings from Last Session* below.

Position

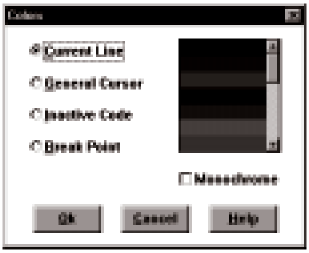
To size the debuggee's window to the maximum desktop area not taken up by the debugger, choose **Window ➤ Position Debuggee**. This has no effect when the debugger is maximized.

Message Groups

To set up your own custom message groups to watch, choose **Options ➤ Custom Groups**. Type a name for the group, then choose the Windows messages from the list box in the **Selective Breakpoint Groups** dialog.

Colors

To customize the debugger selection colors, choose **Options ➤ Custom Colors**. Select the colors for the **Current Line**, **General Cursor**, **Inactive Code** and **Breakpoints** in the **Color** dialog.



Reapply Settings from Last Session

You can tell the debugger to remember and apply settings (breakpoints, window positions and sizes, watch variables, etc) from the previous debug session. To do so choose **Project ➤ Auto Resume on Debug**. The development environment button "restarts" the debugger by adding the "/r" parameter.

Debugger Windows

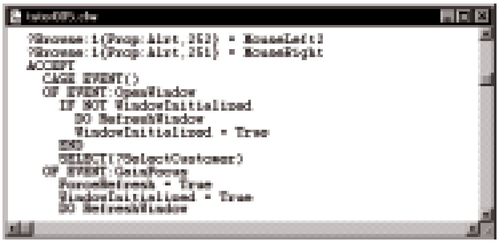
The debugger contains a collection of child windows which track information about the debuggee program. These windows are:

- The **source code** window
- The **Watch Expressions** window
- The **Global Variables** window
- The **Active Procedures** window
- The **Disassembly** window
- The **Machine Registers** window
- The **Library States** window
- The **Windows Messages** window

After you start the debugger, take a moment to arrange the various windows in a format that is comfortable for you. Position the most important windows where you can quickly scan for the information you need. Close or iconize unneeded windows.

Source Code Windows

The **source code** windows display the selected source code modules. The title bar shows the source module name. By default, the next line to execute is green. Lines manually elected by you are light cyan.

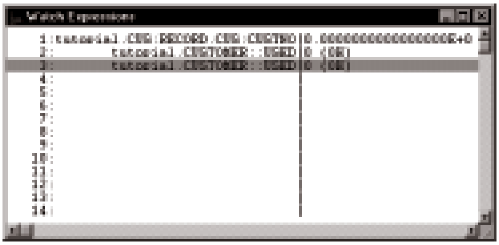


```
?Browse:1{Prop:Alrt,252} = MouseLeft2
?Browse:1{Prop:Alrt,251} = MouseRight
ACCEPT
  CASE EVENT()
  OF EVENT:OpenWindow
    IF NOT WindowInitialized
      DO RefreshWindow
      WindowInitialized = True
    END
    SELECT(?SelectCustomer)
  OF EVENT:GainFocus
    ForceRefresh = True
    WindowInitialized = True
    DO RefreshWindow
```

Tip: **If the debugger opens without listing any source code documents in the Sources to include in this session dialog, the most probable cause is that none of the source code files listed in the Project Tree contained debug information. See *Preparing Your Projects for Debugging*.**

Watch Expressions Window

The **Watch Expressions** window lets you pick specific variables (local, global, or both) and expressions to monitor during the debugging process. That is, rather than searching through the Active Procedures window and Global Variables window with each debugging cycle, you can add the variables to the **Watch Expressions** window and examine all the pertinent values in one place.



```
Watch Expressions
 1:tutorial.CUS:RECORD.CUS:CUSTNO|0.00000000000000000E+0
 2:        tutorial.CUSTOMER::USED|0 (0H)
 3:        tutorial.CUSTOMER::USED|0 (0H)
 4:                               |
 5:                               |
 6:                               |
 7:                               |
 8:                               |
 9:                               |
10:                               |
11:                               |
12:                               |
13:                               |
14:                               |
```

The title bar is **Watch Expressions**. See *Editing Watch Expressions* to learn the syntax for watch expressions. To add a variable to the **Watch Expressions** window:

1. DOUBLE-CLICK on an empty line in the **Watch Expressions** window.

2. When the **Watch Expression** dialog opens, type a variable name (as it appears in the global variables list) then press **OK**.
3. Alternatively, press the **Browse** button, select a variable from the list, then press **OK** twice.

   The expression or variable appears in the **Watch Expressions** window, and its current contents appear next to it.

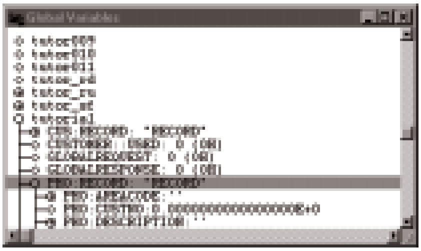
You can also add a variable to the **Watch Expressions** window by DOUBLE-CLICKING on a variable in either the **Global Variables** window or the **Active Procedures** window.

You can edit a variable in the **Watch Expressions** window by DOUBLE-CLICKING on it, and typing an expression in the **Watch Expression** dialog. See *Editing Watch Expressions*.

Tip: To quickly add a structured variable (such as a record, string or array) to the watch list, DOUBLE-CLICK on it in the Global Variables or Active Procedures windows, then press the Copy Variable to Watch button.

Global Variables Window

The **Global Variables** window shows the current value of each component of each global variable. For example, a string variable of eight characters, appears on eight separate lines showing the contents of each position of the string.



```
tutor009
tutor010
tutor011
tutor_sd
tutor_ru
tutor_sf
tutorial
  CUS:RECORD: 'RECORD'
  CUSTOMER:USED: 0 (0H)
  GLOBALREQUEST: 0 (0H)
  GLOBALRESPONSE: 0 (0H)
  PRO:RECORD: 'RECORD'
    PRO:AREACODE: ''
    PRO:CUSTNO:0.00000000000000000E+0
    PRO:DESCRIPTION: ''
```

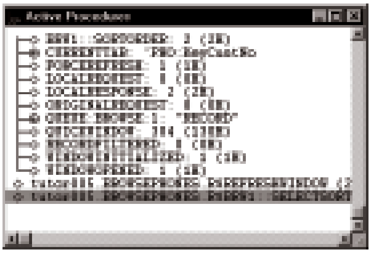
The **Global Variables** window contains tree controls, so you can display only the variables you want to examine. Controls containing a (+) are expandable by CLICKING on them. Controls containing a (-) are contractible by CLICKING on them.

The top level is the source code module which contains the variable. The next level is the variable names.

Tip: To edit global variable values, select an expanded item, then choose Edit ➤ Edit.

Active Procedures Window

The **Active Procedures** window lists the local variables in scope, plus the currently executing procedures and routines, which lets you monitor nested procedure calls. The window appears in tree format. The upper levels represent the names of procedures and the lower levels represent their variables. Procedure and routine names are listed in the order they are called.



```
BRW1::SORTORDER: 1 (1H)
CURRENTVAR: "PHD:RegCustNo
FORCEREFRESH: 1 (1H)
LOCALREQUEST: 0 (0H)
LOCALRESPONSE: 2 (2H)
ORIGINALREQUEST: 0 (0H)
QRYE BROWSE 1: "RECORD"
QRYCWINDOW: 304 (130H)
RECORDFILTERED: 0 (0H)
WINDOWINITIALIZED: 1 (1H)
WINDOWOPENED: 1 (1H)
tutor005 BROWSEPHONES PARREFRESHWINDOW (2
tutor005 BROWSEPHONES PARRW1 :SELECTSORT
```

DOUBLE-CLICKING on an active procedure displays its source or disassembly. DOUBLE-CLICKING on a variable copies it to the **Watch Expressions** window.

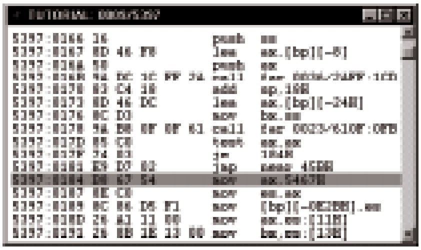
The **Active Procedures** window displays information for the current thread only.

Tip: To edit local variable values, select an expanded item, then choose Edit ➤ Edit.

Disassembly Window

The **Disassembly** window is optional for a project with debug information: choose **Options ➤ Disassembly On** to display it.

If you run the debugger on a program with no debug information, the **Disassembly** window automatically displays the assembly language instructions. The current instruction is selected.



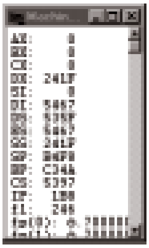
```
5397:0366 16             push  ss
5397:0367 8D 46 F8       lea   ax,[bp][-8]
5397:036A 50             push  ax
5397:036B 9A DC 1C FF 34 call  far 0834/34FF:1CDC
5397:0370 83 C4 10       add   sp,10H
5397:0373 8D 46 DC       lea   ax,[bp][-24H]
5397:0376 8C D3          mov   bx,ss
5397:0378 9A 08 0F 0F 61 call  far 0023/610F:0F08
5397:037D 85 C0          test  ax,ax
5397:037F 74 03          je    384H
5397:0381 E9 D7 02       jmp   near 465BH
5397:0384 B8 67 54       mov   ax,5467H
5397:0387 8E C0          mov   es,ax
5397:0389 8C 86 D8 F1    mov   [bp][-0E28H],es
5397:038D 26 A1 11 00    mov   ax,es:[11H]
5397:0391 26 8B 1E 13 00 mov   bx,es:[13H]
```

DOUBLE-CLICKING (or pressing ENTER) on a line in the **Disassembly** window which contains a jump or call instruction moves the cursor to the target location. ESC returns the cursor to the original location.

INS inserts an unconditional breakpoint at the cursor. DEL removes one. Pressing the SPACE BAR displays the **Breakpoint** dialog.

Machine Registers Window

The **Machine Registers** window shows the current register values; choose **Window ➤ Registers** to display it.



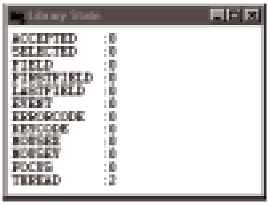
```
AX:     0
BX:     0
CX:     0
DX:  241F
SI:     0
DI:  5467
ES:  529F
DS:  5467
SS:  241F
SP:  B4F8
BP:  C346
CS:  5397
IP:   1B8
FL:   246
```

The **Machine Registers** window shows the register in the left column, and its value to the right.

Library States Window

The **Library States** window displays return values for common Clarion library functions; choose **Window ➤ Library State** to display it. These functions represent all the field and other events.

Functions include ACCEPTED, SELECTED, FIELD, FOCUS, FIRSTFIELD, LASTFIELD, ERRORCODE, AND ERRORFILE.



```
ACCEPTED    :0
SELECTED    :0
FIELD       :0
FIRSTFIELD  :0
LASTFIELD   :0
EVENT       :0
ERRORCODE   :0
KEYCODE     :0
MOUSEX      :0
MOUSEY      :0
FOCUS       :0
THREAD      :2
```

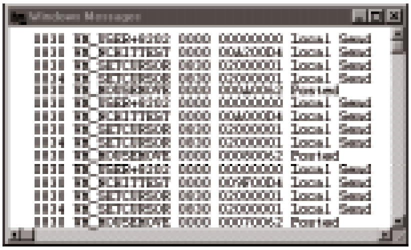
The names listed in EQUATES.CLW and KEYCODES.CLW appear next to the return values.

Windows Messages Window

The **Windows Messages** window displays up to 200 of the most recent message events generated by or directed to your application; choose **Window ➤ Messages** to display it.

The debugger adds a separator line ("——") to indicate a breakpoint occurred.

Every action the user takes—from mouse movement to menu commands—is first processed by Windows. If Windows determines the action is for your application, it passes the information to your application with a message. For example, if the user types the letter “A,” it sends a WM_KEYDOWN message to your application, with the key code for “A” as the first message parameter.



```
[illegible] WM_USER+0202 0000 00000000 Local Send
[illegible] WM_NCHITTEST 0000 00A200D4 Local Send
[illegible] WM_SETCURSOR 0830 02000001 Local Send
[illegible] WM_SETCURSOR 0830 02000001 Local Send
[illegible] WM_MOUSEMOVE 0000 [illegible] Posted
[illegible] WM_USER+0202 0000 00000000 Local Send
[illegible] WM_NCHITTEST 0000 00A000D4 Local Send
[illegible] WM_SETCURSOR 0830 02000001 Local Send
[illegible] WM_SETCURSOR 0830 02000001 Local Send
[illegible] WM_MOUSEMOVE 0000 00080062 Posted
[illegible] WM_USER+0202 0000 00000000 Local Send
[illegible] WM_NCHITTEST 0000 009F00D4 Local Send
[illegible] WM_SETCURSOR 0830 02000001 Local Send
[illegible] WM_SETCURSOR 0830 02000001 Local Send
[illegible] WM_MOUSEMOVE 0000 00070062 Posted
```

Tip: If you include DDE services in your application, we recommend testing your application with another DDE application and monitoring the DDE messages. For more information, see the *Microsoft Windows 3.1 Programmers Reference, Volume 3*, available from Microsoft Press.

Setting Breakpoints

Normally, when debugging an application, you identify a small part of the program which produces incorrect output, or crashes. The Debugging process for this situation will probably require running just that part of the program, and stopping it at one or more points to check its status.

Breakpoints let you automatically halt execution at the line of code at which (or near which) you think the problem occurs. Your program runs up to the breakpoint, then halts and turns control back to the debugger. You can then check the contents of variables and expressions to identify the cause of the problem.

You can also set conditions on the breakpoint, telling the program to continue executing if the condition is false, or turning control over to the debugger if true.

When you set a breakpoint, the source code line where the breakpoint occurs appears in magenta in the source window.

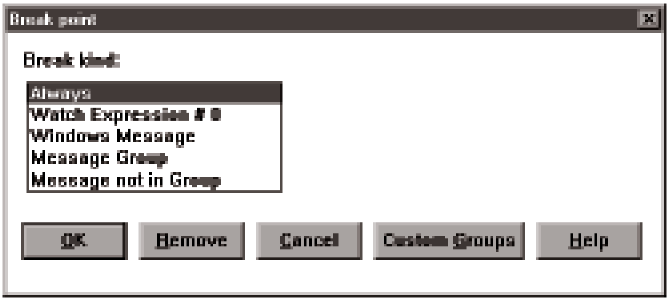
Unconditional Breakpoints

An unconditional or “sticky” breakpoint is placed on a source code line, and stops execution whenever the program encounters that statement. To set an unconditional breakpoint:

1. Open the source code or disassembly window.
2. Locate the line of code to break on and DOUBLE-CLICK on it.

   This opens the **Breakpoint** dialog.
3. Select **Always** then press the **OK** button.

When you execute the **Go!** command, the program will run until it reaches the breakpoint, then stop.



Break point

Break kind:

Always
Watch Expression # 0
Windows Message
Message Group
Message not in Group

Tip: When a source code or disassembly window is the active window, press INSERT to add an unconditional breakpoint, or DELETE to remove one.

Conditional Breakpoints

To narrow the search for bugs, you can tell the debugger to break only when a certain condition exists. The condition takes the form of an expression which can include program variables, operators and constants. You can also tell the debugger to break when it detects a particular Window message or messages.

Breakpoint conditioned on a watch expression

1. Establish a watch expression as described in *Editing Watch Expressions*.
2. Locate the line of code to break on then DOUBLE-CLICK on it.
3. When the **Breakpoint** dialog appears, select **Watch Expression #0**.
4. Type the number of the watch expression (from the watch expressions window) in the **Watch Nr** field.

   This associates the breakpoint with an expression to evaluate.
5. Press the **OK** button.

   When you execute the **Go!** command, the program runs until it reaches the breakpoint, evaluates the watch expression, then stops if the expression is true, i.e., evaluates to a non-zero value.

For example, if variable X should have a maximum value of 999, but increments to 1000 anyway, causing havoc, you can tell the debugger to break at 999, then step through the program to see when and how it reaches 1000.

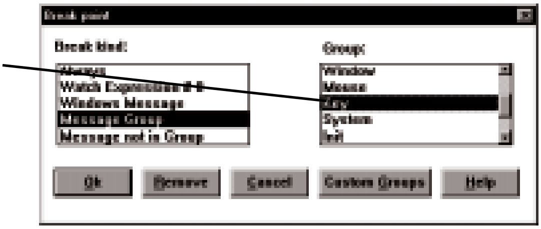
Breakpoint conditioned on a Windows message

1. Locate a line of code to break on and DOUBLE-CLICK on it.
2. When the **Breakpoint** dialog appears, select **Windows Message**.
3. Select a Windows message from the **Windows Message** combo box.
4. Press the **OK** button.

   For example, you could place a breakpoint in a loop which checks for a WM_RBUTTONDOWN message, which is the message Windows sends when the user clicks the right mouse button in your window. When you run the program, you RIGHT-CLICK inside it, and Windows sends a WM_RBUTTONDOWN message to your application. The breakpoint condition would be true.

Breakpoint conditioned on one of several Windows messages

1. Locate the line of code to break on and DOUBLE-CLICK on it.
2. When the **Breakpoint** dialog appears, select **Message Group**; or **Message Not in Group**.



Setting a breakpoint upon receipt of any KEY related message from Windows.

3. Select a message group from the list. This can indicate a category of messages, such as mouse or key messages. You can also set up a custom message group (by pressing the **Custom Groups** button) to remember several specific messages, so that the breakpoint will occur only on one of these messages.
4. Press the **OK** button.

   For example, you could place a breakpoint in a loop which checks for a Key message. When you run the program, when you press a key, Windows sends a message to your application, and the breakpoint condition would be true.

Breakpoint conditioned on an unexpected Windows message

1. Locate the line of code at which you want to establish the breakpoint and DOUBLE-CLICK it.

2. When the **Breakpoint** dialog appears, select **Message not in Group**.
3. Select a message group from the combo box. The breakpoint occurs only when the application receives a message *not* in this group.
4. Press the **OK** button.

Running the Program

The **Go!**, **GoCursor!**, **Step!**, and **ProcStep!** commands execute your application while the debugger monitors it in the background. They allow you to test your application in a controlled environment which helps you identify bugs faster.

> **Tip: These commands are all top level menu commands. No pull down menus appear below them; just place the cursor on the menu command and click, or press ALT plus the underlined letter to execute.**

Go! To run the program from its current state to the next breakpoint, choose **Go!**

When a source or disassembly window is active, the G key executes the command.

GoCursor!

To run the program from its current state to the selected source or assembler line in the source code or disassembly window, choose **GoCursor!**

When a source or disassembly window is active, the C key executes the command.

Step!

To advance the program from the currently selected source or assembler line, one line of code at a time, choose **Step!**

When a source or disassembly window is active, the S key executes the command.

ProcStep!

To advance the program from the currently selected source or assembler line to the next, but to execute through procedure calls without stopping, choose **ProcStep!**

When a source or disassembly window is active, the P key executes the command.

Working with Source Code

When the source code window is active, you can navigate through the source code document with the Edit menu.

> **Tip:** **DOUBLE-CLICKING on a source code line containing a call to a procedure takes you to the first line of that procedure. ESC returns you.**

The following commands are available:

Find Text
Locate the line which contains the text you type into the **Find Text** dialog.

Find Next
Locates the next line which contains text you previously searched for with the **Find Text** command.

Find Procedure
Locates the first source code line for the procedure you pick from the **Find Procedure** dialog. The application's procedures appear in a combo box inside the dialog.

Goto Line
Advances the cursor to the line number you specify.

Current Line
Advances the cursor to the source code line which contains the next statement to execute.

Find Last Error
Places the cursor on the last error.

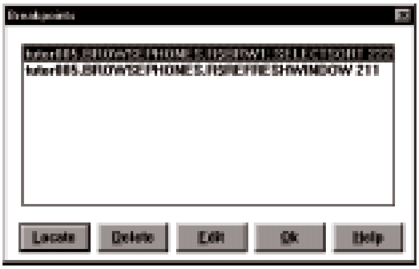
This command will even work after most General Protection Fault errors. The cursor will appear at the source code line where the error took place, or at the line calling the function causing the problem.

Breakpoints
Displays the **Breakpoints** dialog, which lists the breakpoints you've set for this debug session. The breakpoints appear in the format *Source Module: Procedure: Line Number*. Select a breakpoint from the list, then press one of the following buttons: **Locate**, **Delete**, **Edit**, **OK**, or **Help.**

Locate
Scrolls the source window to the line containing the breakpoint.

Delete
Removes the breakpoint.

Edit Opens the **Breakpoint** dialog, See *Setting Breakpoints*.



tutor005.BROWSEPHONES.RSSHOW1.SELECTSORT 222
tutor005.BROWSEPHONES.RSREFRESHWINDOW 211

Editing Watch Expressions

The debugger contains an expression editor dialog which lets you edit a watch expression. Sometimes you want to suspend the program depending on the value of a variable or an expression. For example, you may want to stop the application and look at a variable only if it's value is negative.

To edit a watch expression, select a line in the **Watch Expressions** window (choose **Window ➤ Watch Expressions**) then choose **Edit ➤ Edit**. This opens the **Watch Expression** dialog.

Type an expression in the **Expression to Evaluate** field, then press the **OK** button. When the debugger runs the program and reaches a breakpoint associated with the expression (see *Setting BreakPoints—Conditional Breakpoints*), it tests the expression and halts if the expression evaluates true.

The **Edit Expression** dialog contains the following tools to help you build your expression.

Browse button

Press this button to see a list of the variables local to the procedure you're currently debugging.

The **Make Abs** button automatically prefixes variable names with their memory addresses, module names, and procedure names.

Duplicate button

Press this button to duplicate the selected watch expression.

Memory Overrides

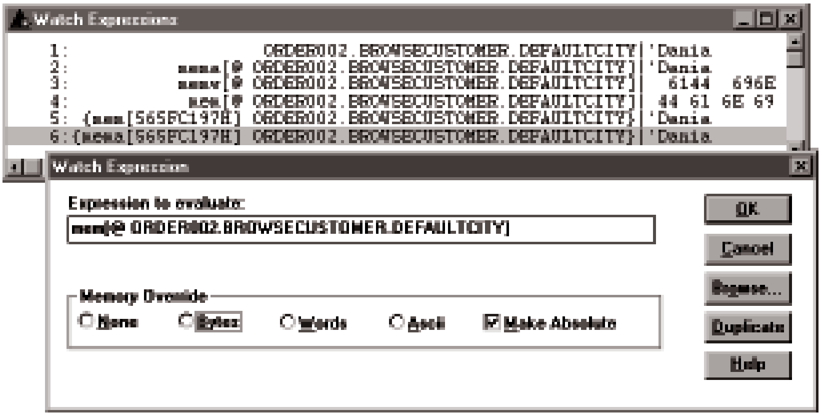
Use these options to fine tune how the debugger displays the watched item. Choose from:

Bytes Select this button to add mem[] to your expression to display it as an array of bytes. This is useful for string variables containing unprintable values.

Words Select this button to add memw[] to your expression to display it as an array of words. This is useful for string variables containing unprintable values.

Ascii Select this button to add mema[] to your expression to display it as ascii text. This is useful for numeric variables that containing ascii (printable) values.

Make Absolute Check this box to use the absolute address of a variable rather than the current variable address. This lets you continue to view local reference variables (or parameters) that would otherwise go out of scope when the procedure returns.



```
Watch Expressions
  1:                      ORDER002.BROWSECUSTOMER.DEFAULTCITY|'Dania
  2:            mema[@ ORDER002.BROWSECUSTOMER.DEFAULTCITY]|'Dania
  3:            memw[@ ORDER002.BROWSECUSTOMER.DEFAULTCITY]|  6144  696E
  4:             mem[@ ORDER002.BROWSECUSTOMER.DEFAULTCITY]| 44 61 6E 69
  5: {mem[565FC197H] ORDER002.BROWSECUSTOMER.DEFAULTCITY}|'Dania
  6:{mema[565FC197H] ORDER002.BROWSECUSTOMER.DEFAULTCITY}|'Dania
```

Watch Expression

Expression to evaluate: mem[@ ORDER002.BROWSECUSTOMER.DEFAULTCITY]

Memory Override: ○ None ○ Bytes ○ Words ○ Ascii ☑ Make Absolute

You can prefix the variable name with a procedure name, a module name, or both. This lets you name a variable not currently in scope, for example, a variable in another procedure that would not be visible for the current procedure.

❑ To specify a procedure and variable, prefix the variable with the procedure name plus a period (“.”).

  For example, “RoyalFlush.King” refers to a variable called King in the procedure called RoyalFlush.

❑ To specify a module and global variable, prefix the variable with the module name plus a period (“.”).

  For example, “NewDeal.Shuffled” refers to a global variable called Shuffled in the module called NewDeal.

❑ To specify a local variable in a procedure in another module, combine the prefixes.

  For example, “Poker.RoyalFlush:King” refers to the variable called *King* in the procedure called *RoyalFlush* in the module called *Poker*.

❏ You may specify register names (for example, *ax*) in a watch expression.

❏ You may use the unary operator (@) to denote the address of a memory object.

Tip: The debugger automatically applies the correct prefix if the variable is unique.

The following list presents the operators and expression syntax for the **Edit Expression** dialog. The operators are language independent, derived from Clarion, C/C++, and Modula 2/Pascal operators.

Key	*Function*
`+`	add
`-`	subtract
`*`	multiply
`/ or DIV`	divide
`% or MOD`	modulus (remainder)
`|`	bitwise OR
`&`	bitwise AND
`<`	less than
`<=`	less than or equal to
`>`	greater than
`>=`	greater than or equal to
`=`	equal
`!= or <>`	not equal
`! or NOT`	logical NOT
`& or AND`	logical AND
`| or OR`	logical OR
`*`	indirection (when prefix)
`^`	indirection (when post-fix)
`->`	point at member
`.`	select member (record field)
`::={e,d}`	display expression e as if it was the same type d

Editing Variables at Run Time

Using the debugger, you can change the value contained in a variable while the program is suspended. You can then resume the program to test execution with the variable containing the new value.

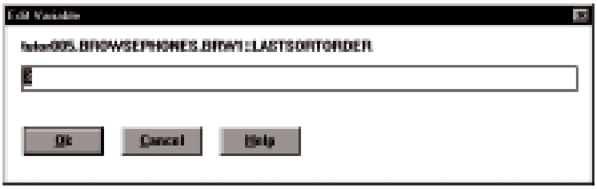
To change the contents of the variable

1. Select the variable in either the **Global Variables** or the **Active Procedures** windows.

Tip: You can only edit fully expanded variables; that is, if there is a + sign beside the variable, you cannot edit it's value. click on the + sign to expand, then edit.

2. Press F2, or choose **Edit ➤ Edit.**

This opens the **Edit Variable** dialog.



tutor005.BROWSEPHONES.BRW1::LASTSORTORDER

3. Type a new value for the variable and press the **OK** button.

   When you choose **Go!**, **GoCursor!**, **Step!**, or **ProcStep!**, the program resumes execution with the variable changed to the new value.

32-bit Debugger

The 32-bit debugger provides debugging for 32-bit programs under Windows 95 and Windows NT. It may also be used as the system debugger.

Tutorial

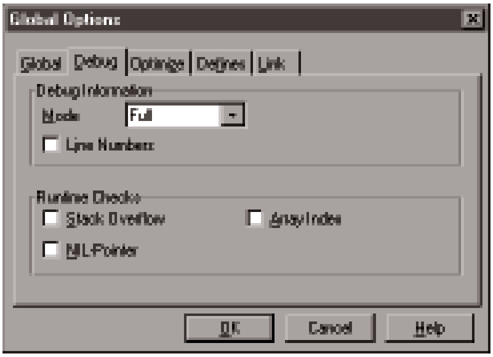
> **Note: This tutorial uses the Clarion Template chain. You must register these templates (CW.tpl) before doing the tutorial.**

Prepare Your Program for Debugging

In this tutorial you will debug the Orders application, installed by default as \EXAMPLES\TUTOR\ORDERS.APP. We have deliberately created a bug in this application. To begin, start Clarion and open the Orders application.

To debug your program with the 32-bit debugger, you must use the 32-bit compiler and tell the compiler to generate debug information. Use the **Project Editor** to establish these two settings (32-bit with debug information).

1. Open the Orders application with Clarion.
2. Press the **Project** button to open the **Project Editor**.
3. Press the **Properties** button to open the **Global Options** dialog.
4. On the **Global** tab, select *Windows - 32 bit* from the**Target OS** drop-list.
5. On the **Debug** tab, select *Full* from the **Mode** drop-down list.



Mode: Full

☐ Line Numbers

☐ Stack Overflow ☐ Array Index

☐ NIL Pointer

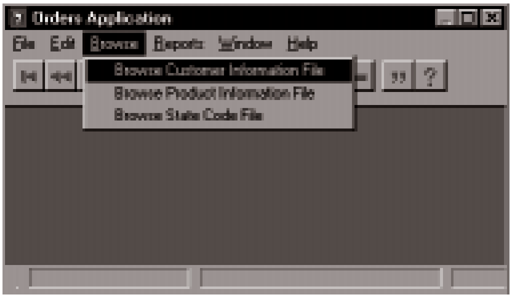
6. Press **OK** twice to return to the application tree.

Identify the Bug

We have deliberately created a bug in the Orders application. To see it:

1. Press the button to make and run the Orders program.

2. From the Orders program menu, choose **Browse ➤ Browse Customer Information File**.

   This opens the customer browse list.

3. With the customer list open, press the ▶ button to scroll down one row.

   The list doesn’t scroll! This bug only occurs with next record and next page (downward scrolling). Prior record, prior page, first page, and last page, all work fine.

   You’ve identified the bug: it is in the BrowseCustomers procedure and it only occurs when scrolling downward.

4. Shut down the Orders program.

   The Clarion environment gains focus.

Start the Debugger

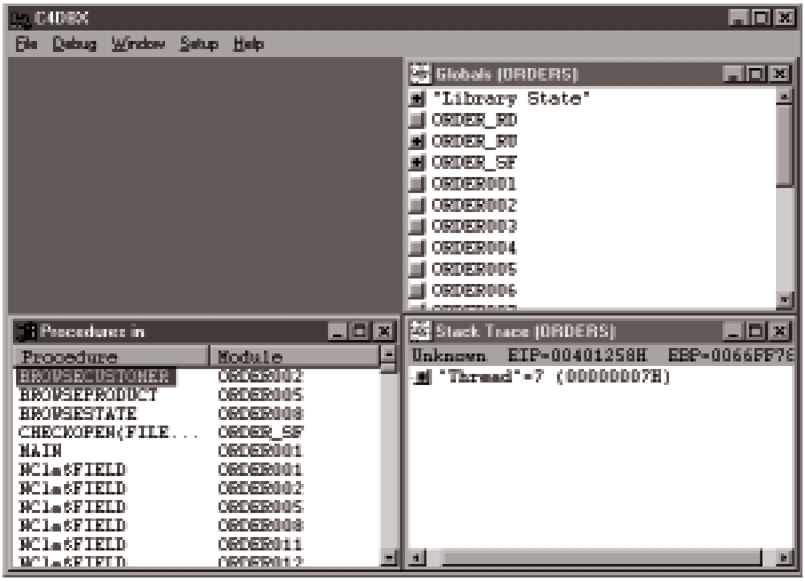
To start the debugger:

1. From the Clarion environment, press the 🐞 button to start the debugger.

   You may want to shut down other applications to conserve resources because you will be running the Clarion environment, the debugger, plus the Orders program in addition to any others.

Tip: You can run the 32-bit debugger independent of the development environment to save even more resources: run C5DBx.EXE.

The Clarion environment verifies all files are current, then starts the debugger. The debugger opens the Globals window, the Procedures in window, and the Stack Trace (local variables) window.



"Library State"
ORDER_RD
ORDER_RU
ORDER_SF
ORDER001
ORDER002
ORDER003
ORDER004
ORDER005
ORDER006

Procedure	Module
BROWSECUSTOMER	ORDER002
BROWSEPRODUCT	ORDER005
BROWSESTATE	ORDER008
CHECKOPEN(FILE...	ORDER_SF
MAIN	ORDER001
WCla$FIELD	ORDER001
WCla$FIELD	ORDER002
WCla$FIELD	ORDER005
WCla$FIELD	ORDER008
WCla$FIELD	ORDER011
WCla$FIELD	ORDER012

```
Unknown   EIP=00401258H   EBP=0066FF78
'Thread'=7 (00000007H)
```

The debugger also opens the Trace window and the Disassembly window in the iconized state.

Arrange the Debugger Windows

To successfully use the 32-bit debugger you should optimally arrange the debugger windows so you can quickly access pertinent information. You should restore only those windows you need to see often, and you should arrange the windows so they are easy for you to work with.

1. Maximize the debugger.

   You will usually want to see several debugger windows at once, so you need as much screen space as possible.

2. In the Procedures in window, DOUBLE-CLICK BROWSECUSTOMERS@F to open the source window for the BrowseCustomers procedure.

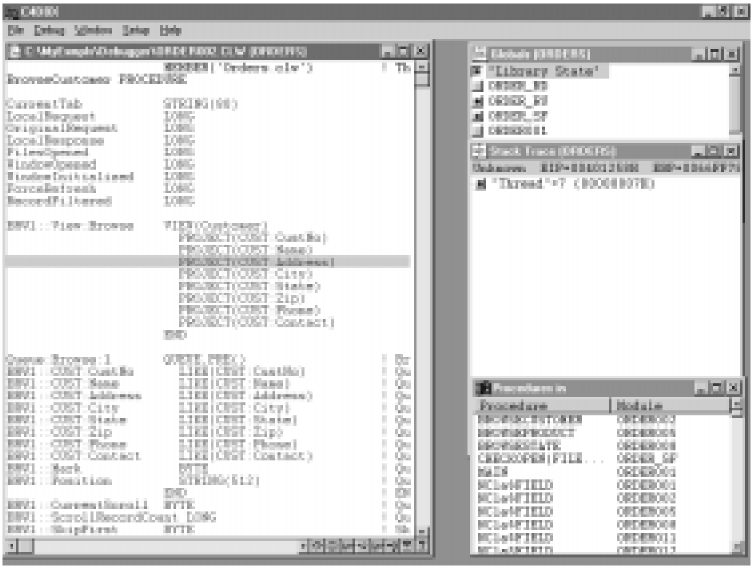
   The debugger opens a source code window for ORDER002.CLW, the source module containing the BrowseCustomers procedure.

   The Procedures in window shows all the procedures and routines for the program you are debugging. DOUBLE-CLICK on any procedure to see the source code for that procedure.

3. Reposition and resize the windows so they are easy for you to work with.

   The debugger "remembers" the window positions between sessions.

   We recommend a configuration that shows lots of source code, shows as many local variables as possible (in the Stack Trace window), and provides a small overlap for easy "switching" between windows.


C4DBX
File Debug Window Setup Help
C:\MyExmpls\Debugger\ORDER002.CLW (ORDERS)
Globals (ORDERS)
Stack Trace (ORDERS)
Procedures in
Procedure
Module

```
                    MEMBER('Orders.clw')           ! Th
BrowseCustomer PROCEDURE

CurrentTab          STRING(80)
LocalRequest        LONG
OriginalRequest     LONG
LocalResponse       LONG
FilesOpened         LONG
WindowOpened        LONG
WindowInitialized   LONG
ForceRefresh        LONG
RecordFiltered      LONG

BRW1::View:Browse   VIEW(Customer)
                      PROJECT(CUST:CustNo)
                      PROJECT(CUST:Name)
                      PROJECT(CUST:Address)
                      PROJECT(CUST:City)
                      PROJECT(CUST:State)
                      PROJECT(CUST:Zip)
                      PROJECT(CUST:Phone)
                      PROJECT(CUST:Contact)
                    END

Queue:Browse:1      QUEUE,PRE()                    ! Br
BRW1::CUST:CustNo     LIKE(CUST:CustNo)            ! Qu
BRW1::CUST:Name       LIKE(CUST:Name)              ! Qu
BRW1::CUST:Address    LIKE(CUST:Address)           ! Qu
BRW1::CUST:City       LIKE(CUST:City)              ! Qu
BRW1::CUST:State      LIKE(CUST:State)             ! Qu
BRW1::CUST:Zip        LIKE(CUST:Zip)               ! Qu
BRW1::CUST:Phone      LIKE(CUST:Phone)             ! Qu
BRW1::CUST:Contact    LIKE(CUST:Contact)           ! Qu
BRW1::Mark            BYTE                         ! Qu
BRW1::Position        STRING(512)                  ! Qu
                    END                            ! EN
BRW1::CurrentScroll BYTE                           ! Qu
BRW1::ScrollRecordCount LONG                       ! Qu
BRW1::SkipFirst     BYTE                           ! Sk
```

```
'Library State'
ORDER_NB
ORDER_RD
ORDER_SF
ORDER001
```

```
Unknown  EIP=[illegible]  EBP=[illegible]
'Thread'=7 ([illegible])
```

Procedure	Module
BROWSECUSTOMER	ORDER002
BROWSEPRODUCT	ORDER005
BROWSESTATE	ORDER008
CHECKOPEN|FILE...	ORDER_SF
MAIN	ORDER001
NCla#FIELD	ORDER001
NCla#FIELD	ORDER002
NCla#FIELD	ORDER005
NCla#FIELD	ORDER008
NCla#FIELD	ORDER011
NCla#FIELD	ORDER012

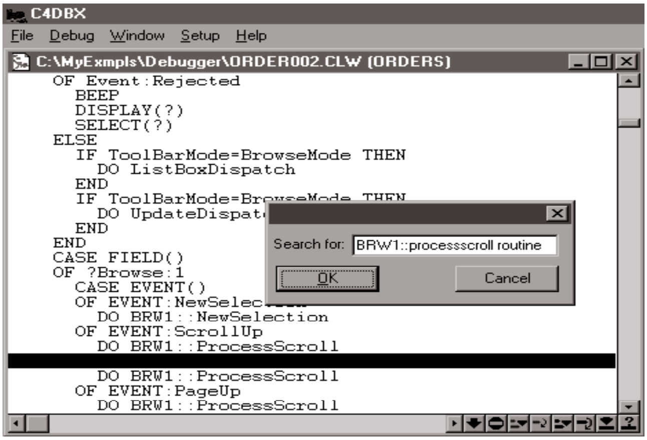
Set BreakPoints

You want to suspend (or break) the program at the point the bug occurs, then look for the cause of the problem, usually in the form of incorrect variable values or incorrect execution sequence. Recall the bug is associated with scrolling downward. To set the breakpoint:

1. CLICK on the ORDER002.CLW window to give it focus.
2. Press F to open the **Search for** dialog.

> **Tip:** **RIGHT-CLICK on the source code window for a popup menu of all the source window functions and their keyboard shortcuts.**

3. In the **Search for** dialog, type *scrolldown*, then press the **OK** button.


C4DBX
File Debug Window Setup Help
C:\MyExmpls\Debugger\ORDER002.CLW (ORDERS)
OK
Cancel

```
    OF Event:Rejected
      BEEP
      DISPLAY(?)
      SELECT(?)
    ELSE
      IF ToolBarMode=BrowseMode THEN
        DO ListBoxDispatch
      END
      IF ToolBarMode=BrowseMode THEN
        DO UpdateDispat
      END
    END
    CASE FIELD()
    OF ?Browse:1
      CASE EVENT()
      OF EVENT:NewSelec
        DO BRW1::NewSelection
      OF EVENT:ScrollUp
        DO BRW1::ProcessScroll

        DO BRW1::ProcessScroll
      OF EVENT:PageUp
        DO BRW1::ProcessScroll
```

Search for: BRW1::processscroll routine

This finds the EVENT:ScrollDown where the BRW1::ProcessScroll routine is called. This routine is a likely location for the breakpoint you need.

Tip: The search is not case sensitive.

4. Press F to open the **Search for** dialog.
5. In the **Search for** dialog, type *BRW1::ProcessScroll routine*, then press the **OK** button.

   This finds the BRW1::ProcessScroll routine.
6. CLICK on the first executable statement after the routine label:

```
IF BRW1::RecordCount
```

Note: Although you can set a breakpoint on routine labels, you may get unexpected results. We recommend setting breakpoints only on executable statements.

7. Press the ⊖ button or type B to to set a breakpoint on this statement.

   The debugger displays breakpoints in red; however, because this is also the currently selected line (green), the breakpoint appears yellow.

Debug the Program

The debugger is now set to suspend program execution at the first statement in the BRW1::ProcessScroll routine. To debug the program:

1. Press the ⇩ button or type G to start the Orders program.

   The Orders program begins execution. Its main window opens on top of the debugger.
2. Choose **Browse ➤ Browse Customer Information file**.

   This opens the customer browse list.
3. With the customer list open, press the ► button to scroll down one row.

   The debugger detects the breakpoint, suspends the Orders program, and displays the breakpoint statement (with a yellow background).
4. Press the button or type T to execute the next statement.

   The debugger executes a single statement, then displays the next source statement to execute with a green background.
5. Repeat step 4 until the first statement in the BRW1::ScrollOne routine executes:

```
 IF BRW1::CurrentEvent = Event:ScrollUp AND BRW1::CurrentChoice > 1
   BRW1::CurrentChoice -= 1
   EXIT
ELSIF BRW1::CurrentEvent = Event:ScrollDown AND BRW1::CurrentChoice < BRW1::RecordCount
   BRW1::CurrentChoice += 1
   EXIT
END
```

Notice that none of the conditional statements execute. This is because neither the primary IF condition nor the ELSIF condition are true.

Examine Variable Values

You can surmise that the above IF condition must be true in order for the customer list to scroll *up* one row, and the ELSIF condition must be true in order for the customer list to scroll *down* one row. You can examine (and modify) the values of the variables within the ELSIF condition to test this theory—BRW1::CurrentEvent must equal Event:ScrollDown, and BRW1::CurrentChoice must be less than BRW1::RecordCount.

To examine and edit the variable values:

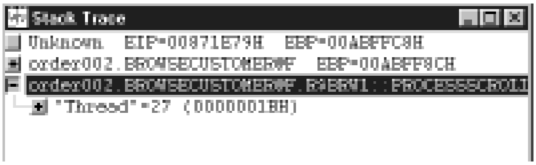
1. Press the button or type G to continue executing the Orders program.

   The Orders program gains focus and continues execution where it left off.

2. Again, press the ▶ button to scroll down one row.

   Again, the debugger detects the breakpoint, suspends the Orders program, and displays the breakpoint statement.

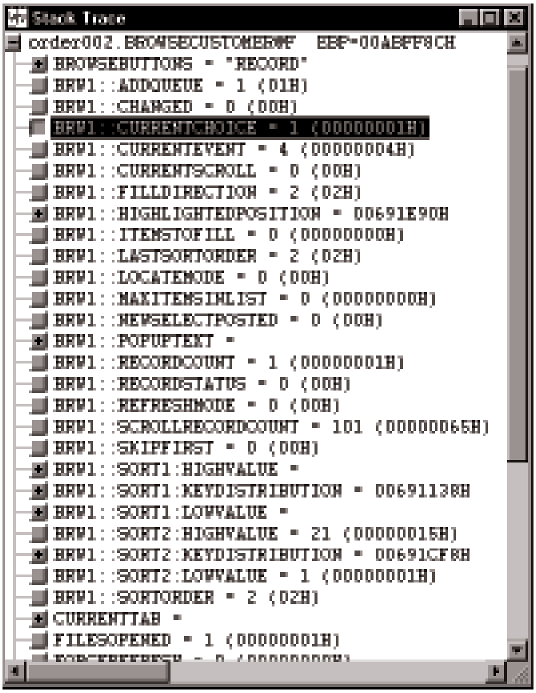
3. CLICK in the Stack Trace window, or choose **Window ➤ Stack Trace**.



```
Unknown  EIP=00871E79H  EBP=00ABFFC8H
order002.BROWSECUSTOMER@F  EBP=00ABFF8CH
order002.BROWSECUSTOMER@F.B#BRW1::PROCESSSCROLL
  "Thread"=27 (0000001BH)
```

4. In the Stack Trace window, CLICK on the (+) beside the ORDER002.BROWSECUSTOMER@F to expand the list of local variables.

   The debugger shows the label of the variable on the left and its value on the right. The value of BRW1::CurrentChoice is 1. Several rows down, you can see the value of BRW1::RecordCount is also 1. These values would cause the ELSIF condition to fail because the condition requires BRW1::CurrentChoice to be less than BRW1::RecordCount, when in fact, they are equal.



```
order002.BROWSECUSTOMERWF   EBP=00ABFF8CH
    BROWSEBUTTONS = "RECORD"
    BRW1::ADDQUEUE = 1 (01H)
    BRW1::CHANGED = 0 (00H)
    BRW1::CURRENTCHOICE = 1 (00000001H)
    BRW1::CURRENTEVENT = 4 (00000004H)
    BRW1::CURRENTSCROLL = 0 (00H)
    BRW1::FILLDIRECTION = 2 (02H)
    BRW1::HIGHLIGHTEDPOSITION = 00691E90H
    BRW1::ITEMSTOFILL = 0 (00000000H)
    BRW1::LASTSORTORDER = 2 (02H)
    BRW1::LOCATEMODE = 0 (00H)
    BRW1::MAXITEMSINLIST = 0 (00000000H)
    BRW1::NEWSELECTPOSTED = 0 (00H)
    BRW1::POPUPTEXT =
    BRW1::RECORDCOUNT = 1 (00000001H)
    BRW1::RECORDSTATUS = 0 (00H)
    BRW1::REFRESHMODE = 0 (00H)
    BRW1::SCROLLRECORDCOUNT = 101 (00000065H)
    BRW1::SKIPFIRST = 0 (00H)
    BRW1::SORT1:HIGHVALUE =
    BRW1::SORT1:KEYDISTRIBUTION = 00691138H
    BRW1::SORT1:LOWVALUE =
    BRW1::SORT2:HIGHVALUE = 21 (00000015H)
    BRW1::SORT2:KEYDISTRIBUTION = 00691CF8H
    BRW1::SORT2:LOWVALUE = 1 (00000001H)
    BRW1::SORTORDER = 2 (02H)
    CURRENTTAB =
    FILESOPENED = 1 (00000001H)
    FORCEREFRESH = 0 (00000000H)
```

You can surmise that BRW1::RecordCount is the offender because you certainly saw more than one record in the customer browse list. Test this theory by changing the value of BRW1::RecordCount to a larger number, for example, 20.

Edit Variable Values

1. In the Stack Trace window, RIGHT-CLICK on BRW1::RecordCount, then choose **Edit variable** to open the **Edit** dialog.

2. In the **Edit** dialog, type *20*, then press **OK**.

   The debugger changes the value of BRW1::RecordCount to 20. Now you can continue execution of the Orders program to see if it scrolls properly.

3. Press the button or type T to execute the next statement.

   The debugger displays the next source statement to execute with a green background.

4. Repeat step 3 until the first statement in the BRW1::ScrollOne routine executes.

   Notice that the ELSIF conditional statements do execute. This is because the ELSIF condition is now true: BRW1::CurrentChoice is less than BRW1::RecordCount.

5. Press the button or type G to continue executing the Orders program.

   The Orders program continues execution where it left off.

This time, the customer list scrolls correctly! This confirms the theory that BRW1::RecordCount contains an incorrect value at this point in the program.

Examine Global Variable Values

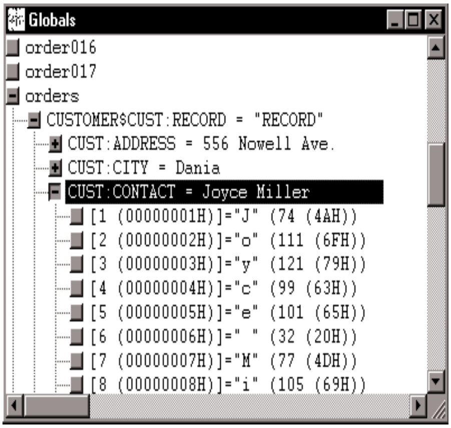
Often you will want to examine global variables, record buffer values, and the values of Clarion's built-in "Library State functions (ACCEPTED(), EVENT(), ERRORCODE(), etc.).

To examine and edit these global variable values:

1. Again, press the ► button to scroll down one row.

   Again, the debugger detects the breakpoint, suspends the Orders program.

2. CLICK in the Globals window, or choose **Window ➤ Globals**.
3. In the Globals window, CLICK on the (+) beside ORDERS to expand the list of global variables for the ORDERS module.
4. In the Globals window, CLICK on the (+) beside CUSTOMER$CUST:RECORD to expand the list of customer record buffer fields.
5. In the Globals window, CLICK on the (+) beside CUST:CONTACT to see the value of each byte in the CUST:CONTACT field.



```
order016
order017
orders
  CUSTOMER$CUST:RECORD = "RECORD"
    CUST:ADDRESS = 556 Nowell Ave.
    CUST:CITY = Dania
    CUST:CONTACT = Joyce Miller
      [1 (00000001H)]="J" (74 (4AH))
      [2 (00000002H)]="o" (111 (6FH))
      [3 (00000003H)]="y" (121 (79H))
      [4 (00000004H)]="c" (99 (63H))
      [5 (00000005H)]="e" (101 (65H))
      [6 (00000006H)]=" " (32 (20H))
      [7 (00000007H)]="M" (77 (4DH))
      [8 (00000008H)]="i" (105 (69H))
```

The Global Variables window shows global variables, record buffers, and Library States. The labels are on the left and the values are on the right.

Tip: You may edit global variables the same way you edited local variables.

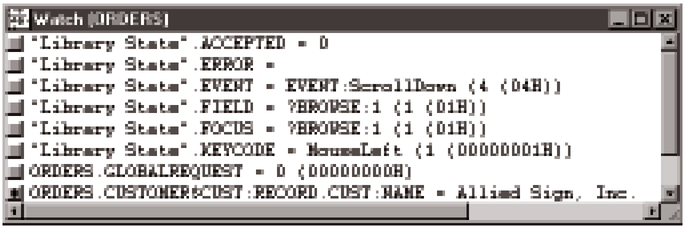
Watch Selected Variables

The Watch window lets you pick specific variables (local, global, or both) and “Library States” to monitor during the debugging process. That is, rather than searching through the Stack Trace window and the Globals window with each debugging cycle, you can add the variables to the Watch window and examine all the pertinent values in one place.

To place variables in the Watch window:

1. In the Stack Trace window or the Globals Window, RIGHT-CLICK on an item (variable or Library State), then choose **Watch variable.**

   This opens the Watch window and places the selected item in the window.



```
"Library State".ACCEPTED = 0
"Library State".ERROR =
"Library State".EVENT = EVENT:ScrollDown (4 (04H))
"Library State".FIELD = ?BROWSE:1 (1 (01H))
"Library State".FOCUS = ?BROWSE:1 (1 (01H))
"Library State".KEYCODE = MouseLeft (1 (00000001H))
ORDERS.GLOBALREQUEST = 0 (00000000H)
ORDERS.CUSTOMER&CUST:RECORD.CUST:NAME = Allied Sign, Inc.
```

2. Repeat step *1* for all the pertinent variables and Library States.

Close the Debugger

1. In the debugger, choose **File ➤ Exit**.

   The debugger automatically shuts down the debuggee (the Orders program) too.

Starting the Debugger

The 32-bit debugger runs as a separate application, but you can start it either from the development environment, or directly from Windows. Starting from Windows, with the development environment unloaded, means more system resources are available for your application and the debugger. The 32-bit debugger can debug multiple programs at the same time.

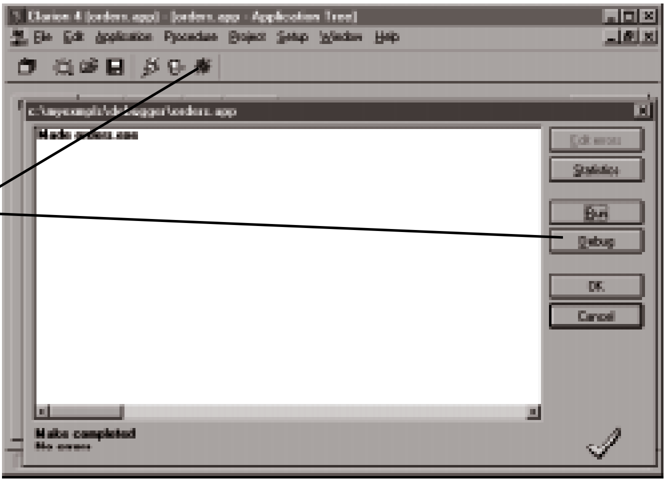
Start the debugger from the development environment

1. Choose **Project ➤ Debug** *or* press the button on the toolbar.

   The environment checks the project information to determine if your application is 16-bit or 32-bit, and starts the corresponding debugger.

   or...

2. Compile and link your application by pressing the button, then, with the compile results dialog still open, press the **Debug** button.


Clarion 4 [orders.app] - [orders.app - Application Tree]
File Edit Application Procedure Project Setup Window Help
Edit errors
Statistics
Run
Debug
OK
Cancel

Make orders.exe

Make completed
No errors

Two ways to start the debugger.

Starting the debugger from the environment sets the working directory to the debuggee program’s directory. This ensures the debugger will use any applicable configuration (.INI) files and redirection (.RED) files.

Start the debugger from Windows

When you start the debugger from Windows, you should take steps to set the proper working directory—usually the directory that contains the debuggee. For example, create a Windows 95 shortcut that sets the “Start in” directory.

1. With your preferred method (Start menu, Explorer, Program Manager, etc.) start C5DBx.EXE.

3. Choose **File ➤ File to Debug**, then choose an .EXE file in theWindows file dialog.

> **Tip:** **When debugging, run only the debugger and the debuggee programs. By doing so, you won’t lose data in other applications if a crash occurs during the debugging process.**

Loading the Source Files

The source associated with the debuggee program is automatically loaded and is available for your examination. However, you may specify any additional source files you want the debugger to manage.

To specify additional source files

1. Choose **Window ➤ Source.**

This opens the **Select Source** dialog.

2. Highlight a source file then press the **OK** button.

Repeat for each source file you want to debug.

Setting Debugger Options

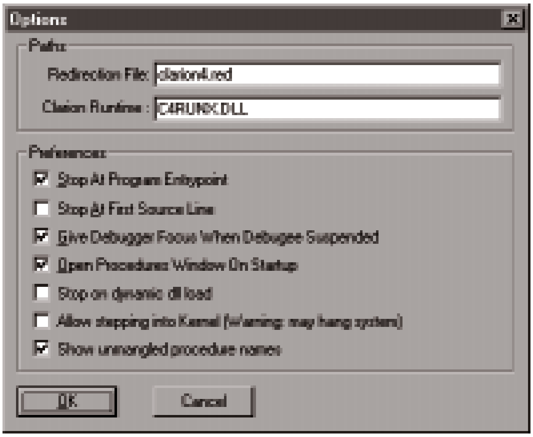
The debugger **Options** menu provides two choices: **Setup** and **Install as System Debugger**. Use **Setup** to customize the debugger.

Setup

Choose the **Options ➤ Setup** command to access the following options:

Redirection File
The debugger uses the redirection file to find project components. A redirection file is optional and follows the same conventions as the Project redirection file. See *Clarion's Development Environment—Search Paths—Redirection File*.

Clarion Run-time Dll
Specifies the Clarion dynamic link library (DLL) linked into the .EXE being debugged.



Options

Paths

Redirection File: clarion4.red

Clarion Runtime : C4RUNX.DLL

Preferences

☑ Stop At Program Entrypoint
☐ Stop At First Source Line
☑ Give Debugger Focus When Debugee Suspended
☑ Open Procedures Window On Startup
☐ Stop on dynamic dll load
☐ Allow stepping into Kernel (Warning: may hang system)
☑ Show unmangled procedure names

Stop At Program Entrypoint
Tells the debugger to stop the debuggee program at its entrypoint upon initial program load. Initial program load (and start) occurs when you choose **File ➤ File to Debug** and select the .EXE file from the Windows file dialog.

Checking this option lets you survey the status of your program at the earliest possible point of execution without explicitly setting a breakpoint.

Stop At First Source Line

Tells the debugger to stop the debuggee program at its first line of executable code upon initial program load. Initial program load (and start) occurs when you choose **File ➤ File to Debug** and select the .EXE file from the Windows file dialog.

Give Debugger Focus When Debuggee Suspended

When the debuggee is suspended at a breakpoint, focus immediately returns to the debugger.

Open Procedure Window on Startup

Tells the debugger to open the **Procedures In** window on debugger startup. See *Debugger Windows*.

Stop on dynamic DLL load

Tells the debugger to suspend debuggee execution when a dynamic DLL load (demand load) is detected. This gives you the opportunity to examine the newly loaded code and set breakpoints before anything else happens.

Allow stepping into Kernel

Lets the debugger examine the Windows Kernel32.dll. This setting can cause system lock ups.

Show unmangled procedure names

Check this box to display procedure names as they appear in source code. Clear the box to show the mangled names that allow procedure overloading.

Install as System Debugger

Installs the 32-bit debugger as the system debugger. In this configuration, the debugger is automatically invoked whenever a program crashes.

Debugger Windows

The debugger contains a collection of child windows which track information about the debuggee program. These windows are:

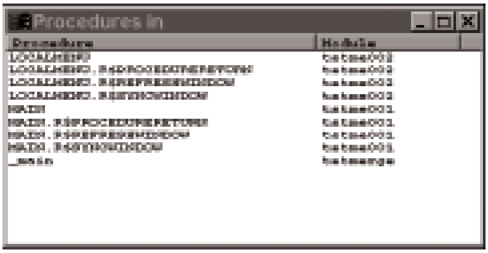
- The **Procedures In** window
- The **Globals** window
- The **Stack Trace** window
- The **Watch** window
- The **source** window
- The **disassembly** window
- The **memory** window

After you start the debugger, take a moment to arrange the various windows in a format that is comfortable for you. Position the most important windows where you can quickly scan for the information you need. Iconize or close unneeded windows. Use the **Window** menu to open windows of special interest.

By default, the debugger initially opens three windows: the **Procedures in** window, the **Globals** window, and the **Stack Trace** window. A fourth window, the **source** window, is opened as soon as you click on a procedure in the **Procedures in** window.

The Procedures In window

The **Procedures in** windowlists the procedures in the debuggee and their associated source modules. CLICK on a procedure name to display its associated source or assembler code.



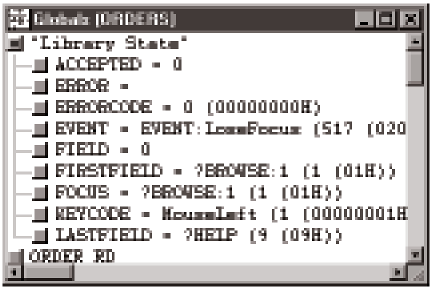
Procedures in

Procedure	Module
LOCALMENU	batma002
LOCALMENU.R$PROCEDURERETURN	batma002
LOCALMENU.R$REFRESHWINDOW	batma002
LOCALMENU.R$SHOWWINDOW	batma002
MAIN	batma001
MAIN.R$PROCEDURERETURN	batma001
MAIN.R$REFRESHWINDOW	batma001
MAIN.R$SHOWWINDOW	batma001
_main	batmamge

Tip: Use the Procedures In window to navigate through your source code.

The Globals window

The **Globals** window displays the current value of each global variable, as well as various Library States (Clarion runtime library functions such as ACCEPTED, EVENT(), FIELD(), etc.).



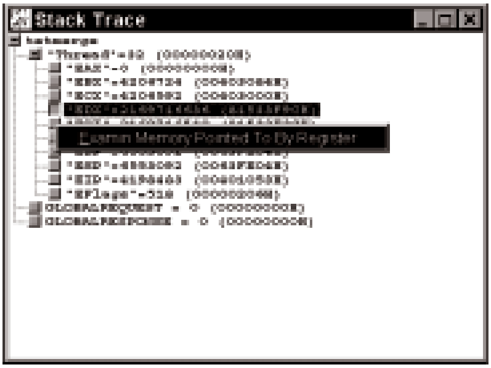
```
Globals (ORDERS)
"Library State"
  ACCEPTED = 0
  ERROR =
  ERRORCODE = 0 (00000000H)
  EVENT = EVENT:LoseFocus (517 (020
  FIELD = 0
  FIRSTFIELD = ?BROWSE:1 (1 (01H))
  FOCUS = ?BROWSE:1 (1 (01H))
  KEYCODE = MouseLeft (1 (00000001H
  LASTFIELD = ?HELP (9 (09H))
ORDER RD
```

The **Globals** window contains expandable tree controls, so you can hide variables you don't want to see. Variables with a (+) button are expandable by CLICKING on them. Variables with a (-) button are contractible by CLICKING on them.

Tip: RIGHT-CLICK on a variable to change its value.

The Stack Trace window

The **Stack Trace** window shows the current register and local variable values. The variable name is on the left and its value in decimal format then in hexadecimal format is on the right. This information is for the current thread only.



```
batmerge
  "Thread"=32 (00000020H)
    "EAX"=0 (00000000H)
    "EBX"=[illegible] ([illegible])
    "ECX"=[illegible] ([illegible])
    "EDX"=[illegible] ([illegible])
    [illegible]
    [illegible]
    [illegible]
    "EBP"=[illegible] ([illegible])
    "EIP"=[illegible] ([illegible])
    "EFlags"=[illegible] ([illegible])
  GLOBALREQUEST = 0 (00000000H)
  GLOBALRESPONSE = 0 (00000000H)
```

The **Stack Trace** window contains expandable tree controls, so you can hide variables you don't want to see. Variables with a (+) button are expandable by CLICKING on them. Variables with a (-) button are contractible by CLICKING on them.

The **Stack Trace** window has a step locator: type a letter to search for variables beginning with that letter.

Tip: **The Stack Trace window has the following special functionality:**

RIGHT-CLICK on a variable to change its value.

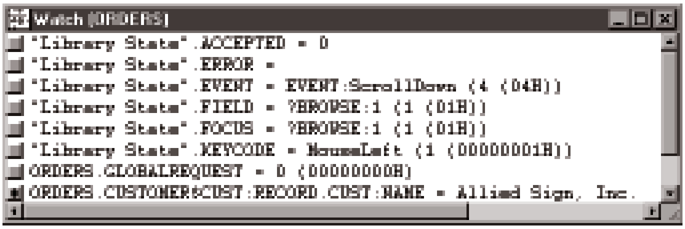
RIGHT-CLICK on a variable to monitor its value.

RIGHT-CLICK on a call to locate its corresponding source line or assembler line.

RIGHT-CLICK on a register to examine the memory pointed to by the register.

The Watch window

The **Watch** window lets you pick specific variables (local, global, or both) and "Library States" to monitor during the debugging process. That is, rather than searching through the **Stack Trace** window and the **Globals** window with each debugging cycle, you can add the variables to the **Watch** window and examine all the pertinent values in one place.



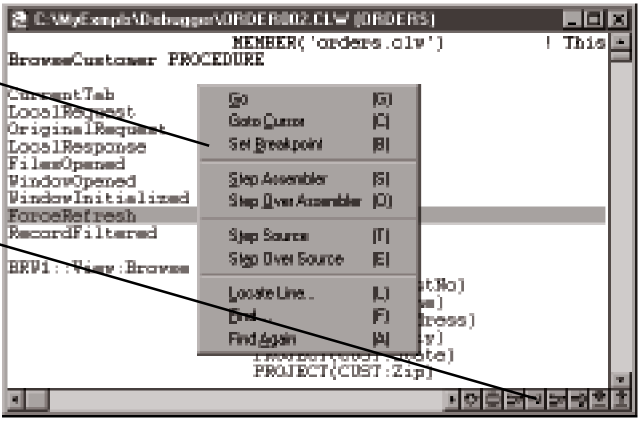
Watch (ORDERS)

```
"Library State".ACCEPTED = 0
"Library State".ERROR =
"Library State".EVENT = EVENT:ScrollDown (4 (04H))
"Library State".FIELD = ?BROWSE:1 (1 (01H))
"Library State".FOCUS = ?BROWSE:1 (1 (01H))
"Library State".KEYCODE = MouseLeft (1 (00000001H))
ORDERS.GLOBALREQUEST = 0 (00000000H)
ORDERS.CUSTOMER&CUST:RECORD.CUST:NAME = Allied Sign, Inc.
```

To place variables in the **Watch** window: in the **Stack Trace** window or the **Globals** window, RIGHT-CLICK on an item (variable or Library State), then choose **Watch variable.** This opens the **Watch** window and places the selected item in the window.

The Source window

Displays a source module. There may be multiple **source** windows open showing different source modules. The title bar shows the module name. The cursor is green. This cursor simply marks a line for your use. It may or may not mark the program’s current position. Breakpoint lines are red. If the current line is also a breakpoint line, it is yellow.



C:\MyExmpls\Debugger\ORDER002.CLW (ORDERS)

```
                              MEMBER('orders.clw')          ! This
BrowseCustomer PROCEDURE

CurrentTab
LocalRequest
OriginalRequest
LocalResponse
FilesOpened
WindowOpened
WindowInitialized
ForceRefresh
RecordFiltered

BRW1::View:Browse
                                                tNo)
                                                e)
                                                dress)
                                                y)
                                                ate)
                                PROJECT(CUST:Zip)
```

The Source Window popup menu and keyboard shortcuts.

The Source Window Taskbar.

Use the **source** window’s task bar buttons to control the execution of the debuggee and to set and remove breakpoints. The taskbar buttons correspond to the options on the popup menu which can be accessed by RIGHT-CLICKING anywhere in the window. See *Running the Program* below for a description of each command.

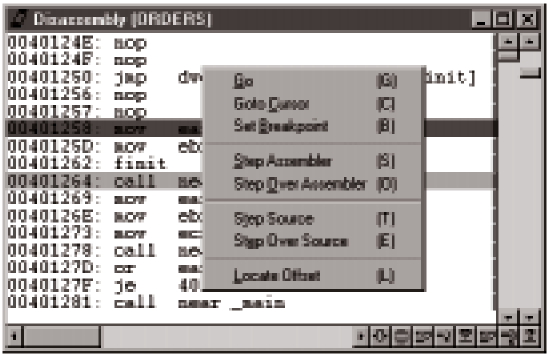
> **Tip:** RIGHT-CLICK **anywhere in the window to access the popup menu.**

The Disassembly window

Displays assembler code. There may be multiple **disassembly** windows open. The title bar shows the .EXE name. The cursor is green. This cursor simply marks a line for your use. It may or may not mark the program’s current position. Breakpoint lines are red. If a line is both the cursor and a breakpoint line, it is yellow.

Blue text has a corresponding source statement associated with it. Moving the cursor to a line with blue text moves the cursor in the **source** window to the corresponding source line.

Use the **disassembly** window's task bar buttons to control the execution of the debuggee, and to set and remove breakpoints. The taskbar buttons correspond to the options on the popup menu which can be accessed by RIGHT-CLICKING anywhere in the window. See *Running the Program* below for a description of each command.



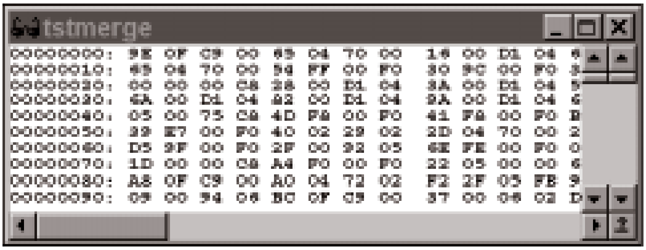
```
0040124E:  nop
0040124F:  nop
00401250:  jmp    dw                  init]
00401256:  nop
00401257:  nop
00401258:  mov    ma
0040125D:  mov    eb
00401262:  finit
00401264:  call   ne
00401269:  mov    ma
0040126E:  mov    eb
00401273:  mov    ec
00401278:  call   ne
0040127D:  or     ma
0040127F:  je     40
00401281:  call   near _main
```

The **disassembly** window has two vertical scroll bars. The left bar scrolls 64K of code at a time, the right bar scrolls 1 display line at a time.

Tip: **RIGHT-CLICK anywhere in the window to access the popup menu.**

The Memory window

Displays memory allocated to the debuggee. The title bar shows the .EXE name. The **memory** window has two vertical scroll bars. The left bar scrolls 64K of memory at a time, the right bar scrolls 1 display line at a time.



```
00000000:  9E 0F C9 00 65 04 70 00  14 00 D1 04 6
00000010:  45 04 70 00 54 FF 00 F0  30 9C 00 F0 3
00000020:  00 00 00 C8 28 00 D1 04  3A 00 D1 04 5
00000030:  6A 00 D1 04 A2 00 D1 04  9A 00 D1 04 6
00000040:  05 00 75 C8 4D F8 00 F0  41 F8 00 F0 B
00000050:  39 E7 00 F0 40 02 29 02  2D 04 70 00 2
00000060:  D5 9F 00 F0 2F 00 92 05  6E FE 00 F0 0
00000070:  1D 00 00 C8 A4 F0 00 F0  22 05 00 00 6
00000080:  A8 0F C9 00 A0 04 72 02  F2 2F 05 FB 9
00000090:  09 00 94 04 BC 0F C9 00  37 00 06 02 D
```

Setting Breakpoints

Normally, when debugging an application, you identify a small part of the program which produces incorrect output, or crashes. The debugging process for this situation will probably require running just that part of the program, and stopping it at one or more points to check its status.

Breakpoints allow you to automatically halt execution at the line of code at which (or near which) you think the problem occurs. Your program runs up to the breakpoint, then halts and turns control back to the debugger. You can then check the contents of variables to identify the cause of the problem, and step through from that point on.

When you set a breakpoint, the line where the breakpoint occurs appears in red in the source and disassembly windows.

Tip: Breakpoints appear yellow when you first create them because both the red breakpoint color and the green cursor color are present.

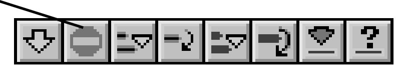
To set a breakpoint

1. Navigate to the source or assembler code where you want the debugger to break.

   CLICK on a procedure name in the **Procedures In** window to jump to that procedure. Or RIGHT-CLICK in the **source** window to access the **Find** command to find a text string.

2. Highlight the line of code to break on.
3. Press thebreakpoint ⊖ button.

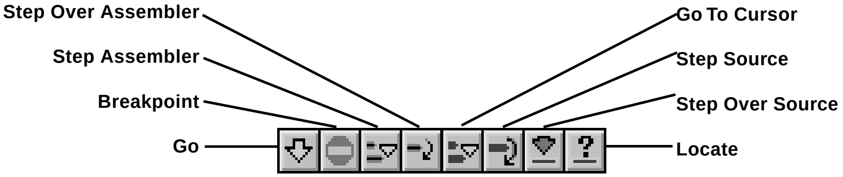
   The breakpoint button appears on the **source** and **disassembly** window taskbars as a round red icon. The breakpoint button acts as a toggle. Pressing it a second time removes the breakpoint.


The Breakpoint button.

Running the Program

The taskbar buttons on the **source** windows and the **disassembly** windows control execution of your program. Similar taskbars appear on each **source** and **disassembly** window. It makes no difference which taskbar you use. *Program execution always continues from the point at which it stopped.*

Alternatively, you can use the popup menus available in each window. The taskbar commands are duplicated on the respective popup menus of the **source** and **disassembly** windows. RIGHT-CLICK anywhere in the window to access the popup menu.



Go Advances the program from its current position to the next breakpoint. If no breakpoints are encountered, the program keeps running.

Step Assembler
Advances the program from its current position, one line of assembler code at a time.

Step Over Assembler
Advances the program from its current position to the next assembler breakpoint, without executing any statements in between.

Step Source
Advances the program from its current position, one line of source code at a time.

Step Over Source
Advances the program from its current position to the next source breakpoint, without executing any statements in between.

Go Cursor
Advances the program from its current position to the cursor. This has the effect of making the cursor a temporary, one-time-only breakpoint.

Locate Line/Offset
Advances the *cursor* (not the program) to the line number (or offset for assembler) you specify.

Find
Advances the *cursor* (not the program) to the source string you specify (source window only).

Find Again
Advances the *cursor* (not the program) to the source string specified for the previous **Find** command (source window only).

Editing Variables at Run Time

Examining Variable Values

The best way to examine variable values at run-time is to look for them in either the **Globals** window or the **Stack Trace** window. Global variables are shown in the **Globals** window and local variables are shown in the **Stack Trace** window in both decimal and hexadecimal format.

Both windows contain tree controls, so that you can expand only the variables you want to examine. Controls containing a (+) are expandable by CLICKING on them. Controls containing a (-) are contractible by CLICKING on them.

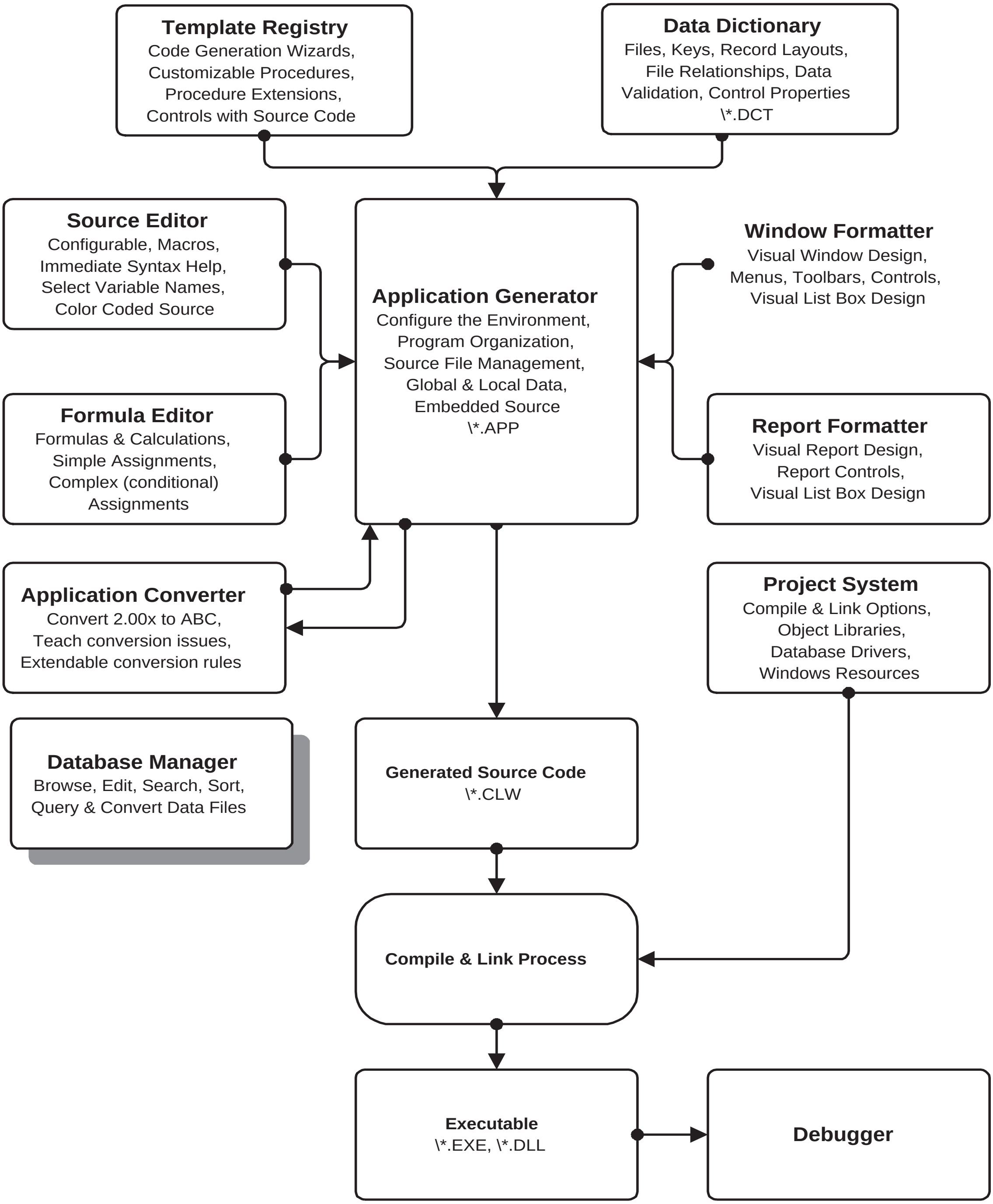
The **Stack Trace** window also shows machine register values and locates the memory area the register points to. RIGHT-CLICK the register, or highlight it and press ENTER, to examine the correct memory location in the memory window.

Changing Variable Values

RIGHT-CLICK on a variable, or highlight it and press ENTER, in either the **Globals** window or the **Stack Trace** window to change its value.

Tip: RIGHT-CLICK **on a variable to change its value.**

14 - DATABASE MANAGER


Template Registry
Code Generation Wizards, Customizable Procedures, Procedure Extensions, Controls with Source Code
Data Dictionary
Files, Keys, Record Layouts, File Relationships, Data Validation, Control Properties
*.DCT
Source Editor
Configurable, Macros, Immediate Syntax Help, Select Variable Names, Color Coded Source
Application Generator
Configure the Environment, Program Organization, Source File Management, Global & Local Data, Embedded Source
*.APP
Window Formatter
Visual Window Design, Menus, Toolbars, Controls, Visual List Box Design
Formula Editor
Formulas & Calculations, Simple Assignments, Complex (conditional) Assignments
Report Formatter
Visual Report Design, Report Controls, Visual List Box Design
Application Converter
Convert 2.00x to ABC, Teach conversion issues, Extendable conversion rules
Project System
Compile & Link Options, Object Libraries, Database Drivers, Windows Resources
Database Manager
Browse, Edit, Search, Sort, Query & Convert Data Files
Generated Source Code
*.CLW
Compile & Link Process
Executable
*.EXE, *.DLL
Debugger

Overview

The Database Manager provides you direct access to data files without the need of creating an application. With the database Manager you can :

- Interactively browse through your data files.
- Add, delete, or change records.
- Add, delete, or change memos.
- Examine data files
- Print data
- Sort data
- Use Query-by-Example to Filter data
- Search data
- Convert data files to new formats

Database Manager is designed to allow application developers free access to their data files. The only entry constraint is the picture assigned to a column. The controls for Data Integrity and Referential Integrity in your Data Dictionary and application are not used.

Normally, Data Integrity is enforced in the end-user applications by the Validity Checks specified in the Database Dictionary, allowing the user to input only valid values in the field to which it applies.

Referential Integrity is ensured in generated applications by the Relationship Constraints you specify in the Database Dictionary. Changing the values in fields which link records in two files, or deleting a Parent record with existing Child records can compromise the Referential Integrity of your Database. This is discussed further in the *Dictionary Editor* chapter.

Browsing Data Files

There are several ways to browse data files with the Database Manager including:

- With the Dictionary Editor's **File ➤ Browse FileLabel** menu command.
- By choosing **File ➤ Open** (or pressing the **Open button** on the **Pick List)**
- By choosing **File ➤ Browse Database...** .

The method you use to call the Database Manager affects its behavior. If you open a file through the Dictionary Editor (with the appropriate .DCT file open) the Database Manager uses all the information in the dictionary. If you open a file from any other area, only the information stored in the file itself is available. This offers maximum flexibility by allowing you to browse a file without a Data Dictionary file (.DCT). The information stored in a file varies with different file systems.

From the Dictionary Editor

This is the best method to call the Database Manager because it provides the most information about the file.

1. Open the appropriate dictionary file (.DCT).
2. In the **Files** list, highlight the desired file.
3. Choose **File ➤ Browse <FileLabel>.**

   The <FileLabel> is the Clarion Label for the file as specified in the dictionary. The File menu display this choice based on the highlighted file.

   If the file does not exist, a dialog appears asking if you want to create it. With the Database Manager, you can create a file even if the file does not have the CREATE attribute (the Enable File Creation check box in the File Properties).

   If the file exists but does not match the layout in the dictionary, a dialog appears asking if you want to convert the file to the current layout. See *Converting Data Files* for more information.

   The Database Manager opens and displays the file.

File ➤ Open

To open an existing data file:

1. Choose **File ➤ Open** (or press the **Open** button on the **Pick List**).

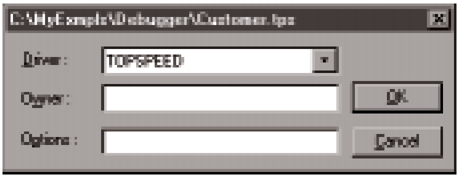
This opens the Windows file dialog.

2. Select Database Files in the **Files of type** drop-list.

   This filters the files list to files that have default database file extensions.

3. Highlight the file to open, then press the **Open** button.

   The Database Manager prompts for the File Driver and owner information.



C:\MyExample\Debugger\Customer.tps

Driver: TOPSPEED

Owner:

Options:

4. Select the Database Driver from the **Driver** drop-down list.

5. Optionally, specify the **Owner** name and **Options.**

   The **Owner** name is a password for access to the file. For an ODBC database, this is the Data Source, user ID, and password separated by commas.

   The **Options** are additional instructions to pass to the database driver (driver strings). See the *Database Drivers* in the *Application Handbook* for more information on valid driver strings for each driver.

6. Press the **OK** button.

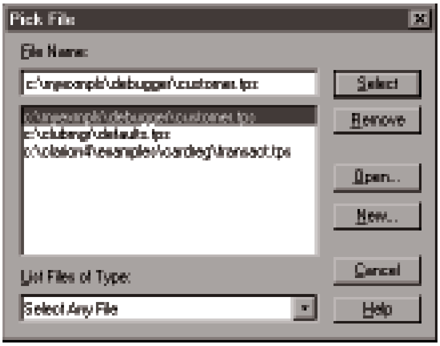
   The Database Manager opens and displays the file.

File ➤ Browse Database

The Browse Database menu command displays a specialized pick list displaying recently opened data files or tables.

1. Choose **File ➤ Browse Database.**

   This opens the Database Manager's Pick List, displaying recently opened files.



Pick File

File Name: c:\myexample\debugger\customer.tps

c:\myexample\debugger\customer.tps
c:\clubmgr\defaults.tps
c:\clarion4\examples\cardlog\transact.tps

List Files of Type: Select Any File

2. Highlight a file in the list and press **Select** or press the **Open** button to choose one from a standard File Open dialog.
3. If you are opening a file for the first time, you are prompted to supply the File Driver to use. Select the driver, then press the **OK** button**.**
4. If you are browsing an ODBC Data Source or a TopSpeed superfile with multiple tables, the Database Manager prompts you to select the table to browse.

   The Database Manager opens and displays the file.

Closing Data Files

Database Manager asks if you want to creates a backup copy before modifying any data in a file. Creating a backup file lets you cancel changes you make while browsing a file. However, some file drivers do not support the creation of a backup. When using one of those file systems, you are not prompted for a backup.

Note: If you do not make a backup copy of the file when modifying it, you will not be able to revert the file to its original state.

To close a file:

1. Choose **File ➤ Close**.

   If you have modified the data, a dialog appears asking if you want to save your changes.

   Yes Saves your changes.

   No Reverts the data file to its last saved state.

   Cancel
   Returns you to the Database Manager.

Sort Order

Once a file is open, you can change the sort order by specifying a different key.

1. Choose **Browse ➤ Order** (or press CTRL+O).

   This opens the **Select File Order** dialog, listing the available Keys and “Record Order”.
2. Highlight the key (or Record Order) you want, then press the **Select** button.

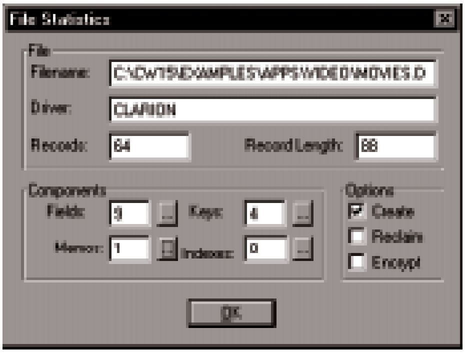
The file is displayed in the selected sort order, and ready for any Database Manager operation.

File Statistics

The File Statistics command lets you examine file information.

To view File Statistics

1. Choose **File ➤ File Statistics** to open the **File Statistics** dialog.



Filename: C:\CW15\EXAMPLES\APPS\VIDEO\MOVIES.D
Driver: CLARION
Records: 64
Record Length: 88
Fields: 9
Keys: 4
Memos: 1
Indexes: 0
☑ Create
☐ Reclaim
☐ Encrypt

Filename
The filename and path for the data file.

Driver
The File System the file uses.

Records
The number of records in the file (including deleted records).

Record Length
The size of each record in bytes.

Fields
The number of fields in the file. Press the ellipsis (...) button to display the field layout.

Keys
The number of keys in the file. Press the ellipsis (...) button to display the key components.

Memos
The number of memos (and BLOBs) in the file. Press the ellipsis (...) button to display the memo field layout.

Indexes
The number of indexes in the file. Press the ellipsis (...) button tp display the index components.

Options
These check boxes indicate whether the corresponding (CREATE, RECLAIM, and ENCRYPT) attributes are present. See the *Language Reference* for more information.

Working with Columns

Database Manager lets you specify which columns (fields) you wish to see on your screen, the size of those columns, and the order in which columns are displayed.

Hiding Columns

Hiding columns removes a column from view. This does not affect the data file in any way. By hiding columns, you see only the desired data columns.

1. Highlight the column to hide.
2. Choose **Column** ➤ **Hide** (or press CTRL+I).

   The column is no longer displayed.

Showing Columns

Once columns are hidden, you can restore them to view or “unhide” them.

1. Choose **Column** ➤ **Show**.

   This opens the **Select Columns to Show** dialog which lists the hidden columns.
2. Highlight the column to restore then press the **OK** button (or press the **All** button to restore all columns).

   The **Show Fields** dialog reopens, allowing you to select another field to restore. Repeat the last step for any other fields you wish to show.
3. When you have all the desired fields displayed, press the **Cancel** button.

Reformatting Columns

The Reformat command lets you quickly define the desired view. You can specify the fields to hide or show and set the order of the columns all at once.

1. Choose **Column** ➤ **Reformat.**

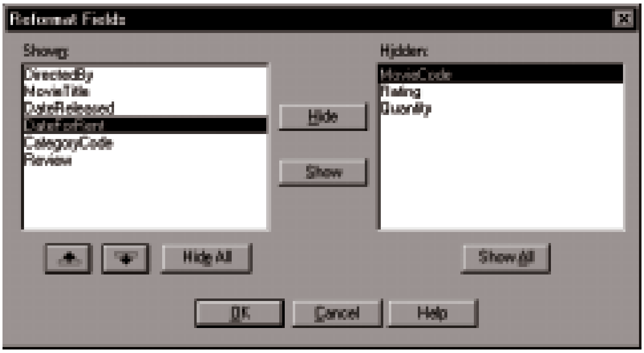
   This opns the **Reformat Fields** dialog.

❏ To show all fields, press the **Show All** button.

❏ To hide all fields, press the **Hide All** button.

❏ To hide individual fields, highlight the field in the **Shown Fields** list box, then press the **Hide** button.

- ❏ To Show individual fields, highlight the desired field in the **Hidden Fields** list box, press the **Show** button. The field reappears in its original location.



Column Justification

You can specify left, right, center, or decimal justification for individual columns.

1. Choose **Column ➤ Justify** (or press CTRL+J).
2. Select the justification type from the drop-down list.

   Left Places the beginning of the display value against the left edge of the display field.

   Right
   Places the end of the display value against the right edge of the display field.

   Center
   Centers the display value in the display field.

   Decimal
   Aligns numeric data on the decimal point.
3. Press the **OK** button.

Column Width

You can adjust the column display width for individual fields.

1. CLICK-AND-DRAG the grid line to the right of the column.

Column Display Pictures

You can change any column’s display picture. This lets you view data in any supported format.

1. Highlight the column to change.
2. Choose **Column ➤ Picture** (or press CTRL+P).
3. Type the desired Picture Token in the **Picture** field.

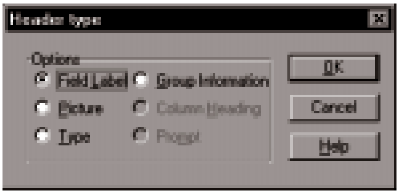
   The data is displayed in the specified format. See *Picture Tokens* in the *Language Reference* for more information.

Column Headers

You can change any column’s header.

1. Choose **Column ➤ Header.**

   This opens the **Header** sub-menu which lists the valid options. A check mark next to an option indicates it is enabled. Choose from:



Field Label
Displays the field’s label from the data dictionary.

Picture
Displays the field’s display picture from the data dictionary.

Type
Displays the field’s data type from the data dictionary.

Group Information
Displays the field’s GROUP information from the data dictionary.

Column Heading
Displays the Default Column Heading from the data dictionary.

Prompt
Displays the default prompt from the data dictionary.

Working with Data Files

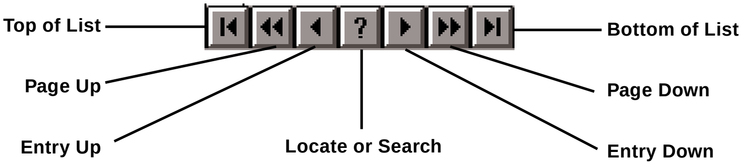
This section describes how to use Database Manager to work with data files.

Navigating Through a File

Database Manager uses the following keystroke conventions to navigate through files:

- In browse mode, LEFT and RIGHT ARROWS move between columns. In edit mode, LEFT and RIGHT ARROWS move between characters.
- UP ARROW and DOWN ARROW, scroll bars, or VCR buttons move between records.
- CTRL+LEFT ARROW and CTRL+RIGHT ARROW swaps columns.
- HOME and END keys move to first and last columns, respectively.
- PAGE UP and PAGE DOWN scroll up and down.
- CTRL+PAGE UP and CTRL+PAGE DOWN moves to the first or last record.
- INSERT adds a record.
- DELETE deletes a record.
- In browse mode, ENTER edits the highlighted field of the current record. In EDIT mode, ENTER accepts your entry in the current field.

Database Manager also provides VCR controls to navigate through files:



Locate (Key) Command

The locate command searches for the first record containing the value you specify in the key field(s). This option is only available when the data file is displayed in a keyed sequence, not in Record Number order. This command only searches fields which are components of the selected key. To search other fields, use the **Search** command.

1. Choose **Edit ➤ Locate.**

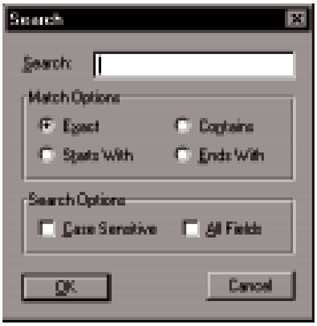
2. Type the search values in the Key fields.
3. Press the **OK** Button.

   The highlight bar is positioned on the first occurrence of the specified values in the key fields. If the value entered does not exist, the next highest match is highlighted

Search and Find Next

The Search command searches for the first record containing the value you specify. The search may be limited to one field or all fields in the record. You may search for:

- An exact match
- A record with a field beginning with the value specified
- A record with a field ending with the value specified
- A record with a field containing the value anywhere within it.



To begin a search

1. Choose **Edit ➤ Search** (or press the **?** VCR button).
2. Type the search value in the **Search For** field.
3. Select the appropriate radio buttons for the desired type of search. Choose from:

   Exact match
   Searches for values that match the specified search string exactly.

   Starts With
   Searches for values that begin with the specified search string.

   Contains
   Searches for values that contain the specified search string.

Ends With

Searches for values that end with the specified search string

4. If you want the search to match case, check the **Case Sensitive** box.
5. If you want the search to consider all fields, check the **All Fields** box.
6. Press the **OK** Button.

   When searching large files, the Search Status window reports the search's progress by displaying the number of records searched and provides the opportunity to abort the search. When the search is complete, the highlight bar is positioned on the first record that matches the search criteria.

To cancel a search in progress

1. Press the **Cancel** button on the Search Status dialog.

To continue a search

1. Choose **Edit ➤ Find Next** (or press CTRL+N).

   Database Manager searches forward from the current record.

Sending Driver Strings

Database Manager lets you communicate with the file driver for the open file. This is equivalent to issuing a SEND command. See *SEND* in the *Language Reference* form ore information. See *Database Drivers* in th *Application Handbook* for valid driver strings for each file driver.

Saving File Definitions as Source Code

Database Manager lets you export the file definition to Clarion source code. This code can later be "pasted" into another code segment or used as part of a source code procedure.

To export a file definition

1. Choose **File ➤ Save as Source**.
2. Specify the file label and filename for the source code.
3. Press the **OK** button.

   The source code for the file definition is created.

Using Query-by-Example

Query-by-Example (QBE) is a powerful tool to find information in a data file. This lets you ask questions of your database based on examples of the desired results. Query-by-Example filters records, allowing you to display a subset of the records based upon a specified example. The filter is in the form of an expression. Most often this expression will compare a specific value to a field.

You specify your query in a QBE list box. A filter expression is built based on expressions entered in this list box. Each column represents a field, and each row represents logical groupings. Expressions entered in different columns on the same row have the effect of an AND operator. Expressions in separate rows have the effect of an OR operator. The filter expression displays below the list box as you enter expressions in the list box.

For example, to find all records with an ID number between 10 and 100, with a last name of Smith or Smythe, you create a query:

IDNumber	FirstName	LastName
>10&<100		=’Smith’
>10&<100		=’Smythe’

Use the ampersand character (&) to represent the AND operator and the vertical bar (|) to represent the OR operator when used in the same field. The example above can also be represented in this fashion:

IDNumber	FirstName	LastName
>10&<100		=’Smith’ | =’Smythe’

Both examples produce a filter expression of (IDNumber > 10 OR IDNumber < 100) AND (LastName = ‘Smith’ OR LastName = ‘Smythe’). The expression displays below the QBE list box.

Note: Although the expression created and displayed in a query is not optimized, the run-time evaluator performs its own optimization. Thus performance is not affected.

The Query is stored in the .INI file when you exit. The next time you open the file in the Database manager, you can filter the records with the same QBE filter.

Editing Data

One of Database Manager's primary functions is the ability to update records without creating procedures to do so. For example, you may need to create a file of twenty choices which are unlikely to change. You could use the Database manager to create the file, enter the twenty records into the data file, and ship the file with your application.

> **Note: Caution should be used when making any changes to data files with the Database Manager. This is a programmer's tool for data file examination and correction, not an end user's tool for data maintenance. There are no controls to prevent you from making changes which could compromise the Data Integrity (invalid data values) or the Referential Integrity ("orphan" Child records) of your Database.**

Database Manager asks if you want to create a backup copy before modifying any data in a file. Creating a backup file lets you cancel changes you make while browsing a file. However, some file drivers do not support the creation of a backup. When using one of those file systems, you are not prompted for a backup.

> **Note: If you do not make a backup copy of the file when modifying it, you will not be able to revert the file to its original state.**

Editing Records

You can edit any field of any record in Database Manager.

1. Highlight the field to change.
2. Choose **Edit ➤ Change** (or press ENTER).

   You may now "edit-in-place." New data is entered in either insert or overwrite mode, depending on the last setting used.
3. To move between fields, press TAB to edit the next field, or SHIFT+TAB to edit the previous field.

Adding Records

Database Manager lets you enter new records in a file.

1. Choose **Edit ➤ Insert** (or press INSERT).

   A new record is added at the bottom of the list. The cursor is positioned in the first field.

2. Type the information you want to enter in a field, then press TAB to edit the next field (or SHIFT+TAB to edit the previous field). Repeat for all fields in which you want to enter data.
3. When you have completed all data entry for the record, press ENTER.

The Database Manager adds the record to the file and updates the keys.

Editing Memos

You can edit a memo in ASCII Text or Hexadecimal format (Hex Mode). Hex Mode editing is useful for memos which contain binary data.

To Edit a Memo in Text Mode

1. Highlight the record.
2. Choose **Edit ➤ Edit Memo** (or press CTRL+E).
3. If the file has more than one memo field, a list box appears. Select the appropriate memo field.
4. Edit the memo.

To Edit a Memo in HEX mode

1. Highlight the record.
2. Choose **Edit ➤ Hex Edit Memo** (or press CTRL+X).
3. If the file has more than one memo field, a list appears. Select the appropriate memo field.
4. Edit the memo.

Showing Deleted Records

By default only active records are shown; however, you can display deleted records. You can use this feature to browse recently deleted records or undelete deleted records.

You can view deleted records in any file that does not have the RECLAIM attribute. If a file does have the RECLAIM attribute, you may still view a deleted record unless new a record has been added in its place.

1. Choose **Window ➤ Show Deleted.**

   A check mark appears next to the menu choice to indicate deleted records are displayed.

   When a deleted record is highlighted, the word **Deleted** displays at the bottom of the window.

Undeleting Records

You can undelete a record in a file— if the file system supports it and the file does not have the RECLAIM attribute. If a file does have the RECLAIM attribute, you may still undelete a record unless new a record has been added in its place.

1. Make sure your view includes deleted records (choose **Window ➤ Show Deleted).**
2. Choose **Edit ➤ Undelete** (or press CTRL+DELETE).

Holding and Releasing Records

Holding a record arms record locking in a multi-user environment. Generally, this excludes other users from writing to, but not reading, the record. The specific action HOLD takes is file driver dependent. See HOLD in the *Language Reference* and see *Database Drivers* in the *Application Handbook*.

To hold the highlighted record

1. Choose **Edit ➤ Hold** (or press CTRL+H).

To release the highlighted record

1. Choose **Edit ➤ Release** (or press CTRL+R).

   When a held record is highlighted, the Held box below the list box is checked.

   You can also hold or release the highlighted record by checking or clearing the Held check box at the bottom of the window.

Converting Data Files

The file conversion utility lets you convert the records in an existing data file to a new file format. For example, when you modify a data dictionary, you can use the conversion utility to convert your existing data to the new format.

The file conversion utility provides two methods of converting a file:

- Immediate Conversion, and
- Generating Source for File Conversion.

Tip: It is always a good idea to make backup copies of your files before running any conversion process.

Immediate Conversion

Immediate conversion handles simple conversions (same file driver, same field names, new or changed fields and keys) between files with the same file system. The conversion is done by the Database Manager, but the Dictionary Editor must also be active.

Note: If you change the name of a field or if you change file drivers, you must generate source code and edit the source code to make the field assignments.

1. Open the data dictionary that describes the file you wish to convert.
2. Modify the data file definition as needed (add or change fields or keys).
3. With the modified file highlighted, choose **File ➤ Browse Filelabel** to load the data file in the Database Manager.

   A message appears, warning that the physical file structure does not match the dictionary declaration.
4. Press the **Yes** button to convert the file.

   The Database Manager converts the file and displays its contents.

Generating Source for File Conversion

Generating Source for File Conversion creates a source code file and a Project file, so you can handle more complex conversions, and you can run the program as often as you want. Generating and compiling source creates an executable file that you can ship to end users to convert their data files to the new format. The Database Manager generates the source code.

Tip: **If there are several files that your customers must convert on-site, you can combine the generated conversion programs into a single conversion application by adding each program to the application as an external source module or as a source template. This way, your customers will have access to all the conversion programs in an easy-to-use menu driven application that is consistent with their other applications.**

Backup the Original File Description

1. Open the data dictionary that describes the file you wish to convert.
2. Copy the data file definition to a new name. To copy a file definition, highlight the file to copy in the Files List and press CTRL+C, then press CTRL+V to paste it. You are prompted to supply a new name and prefix (for example, copy Customer to OldCustomer).

   An alternative would be to copy the entire data dictionary to a new name. You might use this method if there are several files to convert in one session.

Make the New File Description

1. Make any necessary changes (add fields, change the file driver type, etc.) to the definition with the original name. In the example above, the Customer file is the file to modify.
2. Close the Dictionary and save it.

Open the Data File

1. Choose **File ➤ Open** from the menu.
2. Select Database files from the **Files of type** drop-down list.
3. Navigate to the file to convert, then select it.

   The Database Manager prompts you to specify a file driver
4. Select the file's driver from the **Driver** drop-down list.

   The Database Manager opens the file and displays it's contents.

Generate Conversion Source Code

1. Choose **File ➤ Convert File** (or press CTRL+V).

   This opens the **File Convert** dialog, prompting for the information below:
2. The **Source Filename** field defaults to the file you are browsing.
3. In the **Source Dictionary** field, specify the dictionary which contains the source file definition.

4. In the **Source Structure** field, specify the structure within the source dictionary which defines the file to convert (e.g., OldCustomer).

5. In the **Target Filename** field, specify the name of the new file.

   This defaults to the current file name. Specify a new file name if you have not backed up your data file!

6. In the **Target Dictionary** field, specify the dictionary which contains the new (target) file definition.

7. In the **Target Structure** field, specify the structure within the target dictionary which defines the target file.

8. In the **Generated Source** field, press the ellipsis button to specify the full pathname for the generated source code.

Note: By default, the conversion utility writes CONVERT.CLW to the subdirectory of the active application or project file, not necessarily to the subdirectory containing the data or the data dictionary.

9. Press the **OK** button.

   This generates a source file and a corresponding project file (.PRJ) which you can now compile and link to create an executable program to perform the file conversion.

Make and Run the Conversion Program

1. Press the **Exit** button to close the data file in the Database Manager.

Note: Prior to executing the generated conversion program, you must close the data file open in the Database Manager.

2. Load the conversion program by choosing **File ➤ Open**.

3. Select *CONVERT.CLW* (or the file name you specified) in the Windows file dialog.

4. Edit the source code as required to make the field assignments.

   See *Editing Source Code to Make Field Assignments.*

5. Choose **Project ➤ Set** to load the project file.

   Navigate to the project file and select it. This defaults to CONVERT.PRJ.

6. Press to Make and Run the conversion program.

Note: If you set Runtime Library to LOCAL, the conversion program may "lose" the database driver. It issues a "File not found" message. If this happens, re-add the database driver dll to your project.

Finish Up

1. Check the file that has just been converted by opening it with the Database Manager.
2. Delete the “old” file definition from the active dictionary, or archive it into a backup dictionary file.
3. If the converted file is located in a different directory, you may now copy it into the working program directory. If you renamed the file, you may rename it to the original file name at this time.

   The conversion process is now complete. This example creates CONVERT.EXE which you may be ship to end users to convert their files.

Editing Source Code to Make Field Assignments

The File Conversion Utility creates source code to convert a file to a different specification. The conversion is automatic except in the following cases:

- If a field’s label is changed
- If a field is split into two separate fields.
- If two or more fields are combined.
- If a date is converted to a Clarion standard date from some other file system date format.

In these cases you must modify the source code to handle the field assignments. The portion of the source code you need to examine is the *AssignRecord ROUTINE*. This is where field assignments are made. Here is an example:

```
AssignRecord      ROUTINE
  CLEAR(CUS:Record)
  CUS:NUMBER = IN::NUMBER
  CUS:FIRSTNAME = IN::FIRSTNAME
  CUS:LASTNAME = IN::LASTNAME
  CUS:ADDRESS = IN::ADDRESS
  CUS:CITY = IN::CITY
  CUS:STATE = IN::STATE
  CUS:ZIP = IN::ZIP
  CUS:PHONENUMBER = IN::PHONENUMBER
  CUS:ENTRYDATE = IN::ENTRYDATE
```

If you examine the source code, you’ll see that the first line in the routine clears the record buffer. Next, each field in the output file is assigned the value from the matching field in the input file.

Changing a Field’s Label

However, if the field labels do not match, no assignment is made. For example, if you change the LastName field to Surname, the File Conversion utility generates a comment to alert you of an assignment you may need to make:

```
AssignRecord      ROUTINE
  CLEAR(CUS:Record)
  CUS:NUMBER = IN::NUMBER
  CUS:FIRSTNAME = IN::FIRSTNAME
  ! CUS:SURNAME = ‘’
  CUS:ADDRESS = IN::ADDRESS
  CUS:CITY = IN::CITY
  CUS:STATE = IN::STATE
  CUS:ZIP = IN::ZIP
  CUS:PHONENUMBER = IN::PHONENUMBER
  CUS:ENTRYDATE = IN::ENTRYDATE
```

To assign the values from the original file, edit the line containing the assignment to assign the value of LastName to the SurName field as shown below:

```
  CUS:SURNAME = IN::LASTNAME
```

Splitting a Field into Two Fields

Writing the assignment statements to split the contents of a field into two fields involves a little more work, but string slicing minimizes the effort. For this example, let’s assume you had a single field in the original file for a phone number and area code. You now want to store the area code in one field and the phone number in another. Assuming that these fields are numeric data types, you need to temporarily assign the value to a string, then “slice” the string to assign the desired portion to each new field.

In this example the original PhoneNumber field is a ten-digit number, the area code is a three-digit number, and the new PhoneNumber field is a seven-digit number. The AssignRecord ROUTINE in the generated file conversion source code looks like this:

```
AssignRecord      ROUTINE
  CLEAR(CUS:Record)
  CUS:NUMBER = IN::NUMBER
  CUS:FIRSTNAME = IN::FIRSTNAME
  CUS:LASTNAME = IN::LASTNAME
  CUS:ADDRESS = IN::ADDRESS
  CUS:CITY = IN::CITY
  CUS:STATE = IN::STATE
  CUS:ZIP = IN::ZIP
  ! CUS:AREACODE =
  CUS:PHONENUMBER = IN::PHONENUMBER
  CUS:ENTRYDATE = IN::ENTRYDATE
```

Notice there is an assignment from the original PhoneNumber field to the new PhoneNumber field. However, since the new field only stores seven digits, you should change this. To handle the field assignments, create an implicit string variable, assign to it the value of the original PhoneNumber field, then use string slicing to assign the desired portions to the new fields, as shown below:

```
AssignRecord      ROUTINE
  CLEAR(CUS:Record)
  CUS:NUMBER = IN::NUMBER
  CUS:FIRSTNAME = IN::FIRSTNAME
  CUS:LASTNAME = IN::LASTNAME
  CUS:ADDRESS = IN::ADDRESS
  CUS:CITY = IN::CITY
  CUS:STATE = IN::STATE
  CUS:ZIP = IN::ZIP
  TempPhoneNumber" = IN::PHONENUMBER
  CUS:AREACODE = TempPhoneNumber"[1:3]
  CUS:PHONENUMBER = TempPhoneNumber"[4:10]
  CUS:ENTRYDATE = IN::ENTRYDATE
```

For more information on String Slicing, see *Implicit Arrays and String Slicing* in the *Language Reference*.

Combining Two or More Fields

In this example the original PhoneNumber field is a seven-digit number, the AreaCode is a three-digit number, and the new PhoneNumber field is a ten-digit number. The AssignRecord ROUTINE in the generated file conversion source code looks like this:

```
AssignRecord      ROUTINE
  CLEAR(CUS:Record)
  CUS:NUMBER = IN::NUMBER
  CUS:FIRSTNAME = IN::FIRSTNAME
  CUS:LASTNAME = IN::LASTNAME
  CUS:ADDRESS = IN::ADDRESS
  CUS:CITY = IN::CITY
  CUS:STATE = IN::STATE
  CUS:ZIP = IN::ZIP
  CUS:PHONENUMBER = IN::PHONENUMBER
  CUS:ENTRYDATE = IN::ENTRYDATE
```

Notice there is an assignment from the original PhoneNumber field to the new PhoneNumber field, but the original AreaCode is omitted. To handle the field assignments, create two implicit strings and assign them the original AreaCode and PhoneNumber values, then concatenate the strings and assign the result to the new PhoneNumber as shown below:

```
AssignRecord      ROUTINE
  CLEAR(CUS:Record)
  CUS:NUMBER = IN::NUMBER
  CUS:FIRSTNAME = IN::FIRSTNAME
  CUS:LASTNAME = IN::LASTNAME
  CUS:ADDRESS = IN::ADDRESS
  CUS:CITY = IN::CITY
  CUS:STATE = IN::STATE
  CUS:ZIP = IN::ZIP
  TempAreaCode" = IN::AREACODE
  TempPhoneNumber” = IN::PHONENUMBER
  CUS:PHONENUMBER = CLIP(TempAreaCode”) & TempPhoneNumber”
  CUS:ENTRYDATE = IN::ENTRYDATE
```

For more information on String Slicing, see *Implicit Arrays and String Slicing* in the *Language Reference*.

Converting a Date

Let’s assume the ENTRYDATE field contains a string date in MM/DD/YY format. We want to convert this string date into a Clarion standard date so we can perform calculations against it and display it in various other formats.

We use the DEFORMAT function to perform the conversion as shown below. The @D1 represents the Clarion Date Picture that corresponds to the MM/ DD/YY format. See *Date Pictures* in the *Language Reference* for more information.

```
AssignRecord      ROUTINE
  CLEAR(CUS:Record)
  CUS:NUMBER = IN::NUMBER
  CUS:FIRSTNAME = IN::FIRSTNAME
  CUS:LASTNAME = IN::LASTNAME
  CUS:ADDRESS = IN::ADDRESS
  CUS:CITY = IN::CITY
  CUS:STATE = IN::STATE
  CUS:ZIP = IN::ZIP
  ! CUS:AREACODE =
  CUS:PHONENUMBER = IN::PHONENUMBER
  CUS:ENTRYDATE = DEFORMAT(IN::ENTRYDATE,@D1)
```

Converting Legacy Data

As a developer, you may often be asked to write a program to support existing or “legacy” data. Although you initially have no data dictionary describing the data, you can easily convert this legacy data by creating a data dictionary then following the steps described in *Generating Source File for Conversion*.

1. Import the file into a data dictionary by following the steps in *Dictionary Editor—Importing File Definitions*.
2. Follow the steps described in *Generating Source File for Conversion* and *Editing Source Code to Make Field Assignments*.

Printing Data

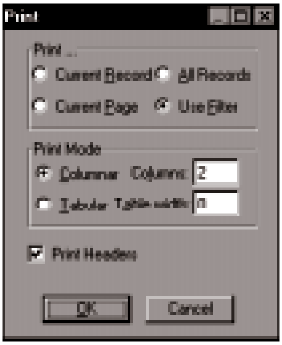
You can print data from Database Manager. You can use these simple reports to look at the data in your files.

To print data

1. Choose **File ➤ Print.**

   This opens the **Print** dialog, allowing you to select the report's format.

2. Select the appropriate radio buttons from the **Print...** group to specify the records to print. Choose from:



○ Current Record ○ All Records
○ Current Page ● Use Filter
● Columnar Columns: 2
○ Tabular Table width: 0
☑ Print Headers

   Current Record
   Prints only the currently highlighted record.

   Current Page
   Prints only the records currently displayed on screen.

   All Records
   Prints all records in the file.

   Use Filter
   Prints only those records which match the filter created in the Query-by-Example dialog.

3. Select the appropriate radio buttons from the **Print Mode** group to specify the report format. Choose from:

   Columnar
   Prints the records in a "spreadsheet" type of format in which each field in the record is a separate column.

   Tabular
   Prints the records in a "form" type of format in which each field in the record is on its own separate print line.

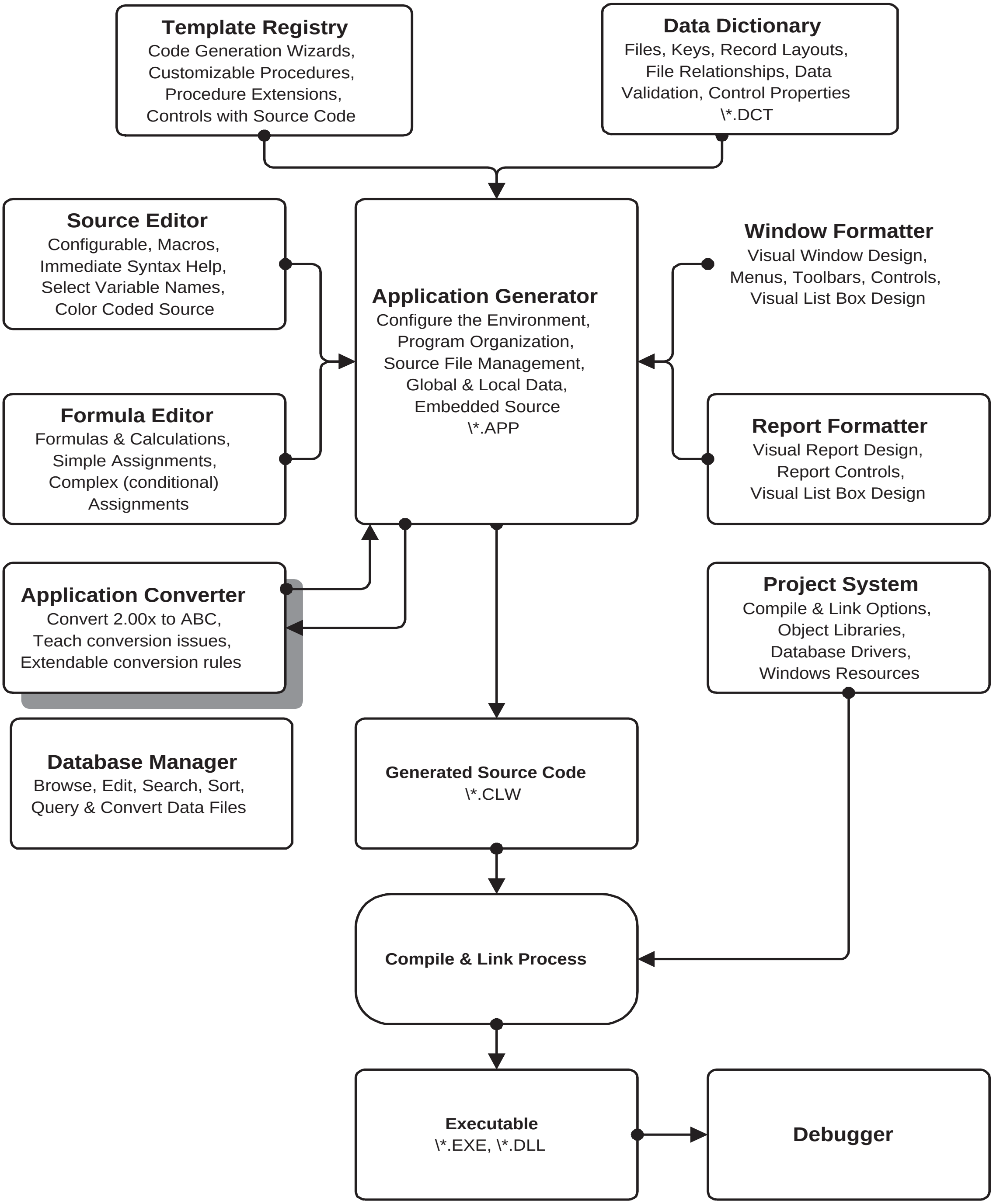
4. If you selected **Columnar,** specify the number of records to print side-by-side in the **Columns** field.

5. If you selected **Tabular,** specify the total number of characters to print for one record in the **Table width** field.

6. If you want to print column headings in Columnar Mode or field labels in Form Mode, check the **Print Header** box.

7. Press the **Print** Button to print the selected record(s).

15 - APPLICATION CONVERTER



Overview

The Application Converter is a utility that takes a Clarion application (.APP) or text application (.TXA) as input, applies conversion rules that you specify, and produces a converted application or .TXA as output. By default, the Converter leaves the original application completely intact.

Interchangable, Modifiable Conversion Rules

The conversion rules are encapsulated from the reading, parsing, and writing of the application files so that conversion rules can be easily added, modified, and deleted.

Clarion 5 supplies two sets of conversion rules. One set of rules converts Clarion for Windows 2.00x applications to Clarion 5 ABC applications. The other set of rules converts Clarion 4 beta applications to Clarion 5 applications.

TopSpeed will provide additional rule sets with future releases of Clarion to help developers upgrade their applications. In addition to the TopSpeed rule sets provided, you may write your own rules to convert applications based on your own templates or third-party templates. See *Writing Your Own Conversion Rules* for more information.

Wizard Interface

The Application Converter provides a “wizard” style interface that guides you through the application conversion in a friendly, straight-forward way. The conversion rules participate in the wizard interface in two ways. First, each rule provides a switch that determines whether or not to apply the rule (see *Configure the Conversion Rules*). Second, if the rule applies, it explains the underlying reason for the change, and it provides an opportunity for you to confirm each proposed change (see *Dispose of Proposed Changes*).

This approach, coupled with TopSpeed’s conversion rules makes the Application Converter more than a conversion utility. The Application Converter is a teaching tool that not only converts your application from one version to another, it also *shows you what* it is changing, and *tells you why* it is changing it.

Application Converter Goals

Initial Scope

Converts Clarion for Windows 2.00x applications to Clarion 5 ABC applications, and converts Clarion 4 beta applications to Clarion 5.

Speed

Speeds up the conversion of your applications, especially where many applications are involved.

Teach

Teaches developers the most commonly encountered conversion issues and their solutions. To this end the program is visual and interactive.

Reusability

Provides an extendable, configurable utility to port Clarion applications between *any* two template sets, including non-TopSpeed templates.

Using the Conversion Wizard

By default, the Converter leaves the original application completely intact. So you may run it as many times as you want without affecting the original application. The original application can stay up and running until you are completely satisfied that the converted application is ready for prime time.

Tip: The ABC Templates must be registered to succesfully convert applications with the Application Converter. The ABC Templates are preregistered when you install Clarion. See *Application Generator—Registering Templates*.

Starting the Conversion Process

Start the Application Converter

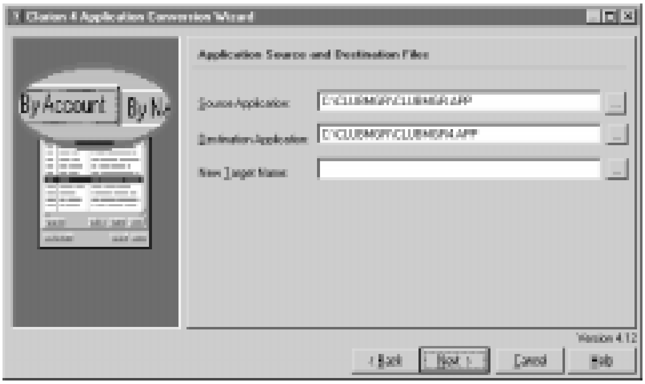
From the main menu, choose **File ➤ Convert Application** to start the Application Converter. You may also run the Application Converter independent of the Clarion environment by running C5conv.exe. This starts the Application Converter and opens the Welcome page. Press the **Next** button to open the **Application Source and Destination Files** page.

Set Source and Target Applications

The **Application Source and Destination Files** page provides the following prompts.

Source Application

Type the pathname of the source .APP or .TXA file, or press the ellipsis button to select the source from the Windows file dialog. By default, the Application Converter only reads from the source application.


Clarion 4 Application Conversion Wizard
Application Source and Destination Files
By Account
By N
...
...
...
Version 4.12
< Back
Next >
Cancel
Help

Source Application: C:\CLUBMGR\CLUBMGR.APP

Destination Application: C:\CLUBMGR\CLUBMGR4.APP

New Target Name:

Destination Application

The Application Converter provides a default target pathname derived from the source pathname. To change the default, type the pathname of the target .APP or .TXA file, or press the ellipsis button to select the target from the Windows file dialog. We recommend naming a target application other than the source application.

New Target Name

Type the pathname of the target application's target file, or press the ellipsis button to select the target file from the Windows file dialog. By default, the converted application retains the same target file as the source application, so making the converted application overwrites the existing target file, unless you specify a new target name.

Configure the Conversion Rules

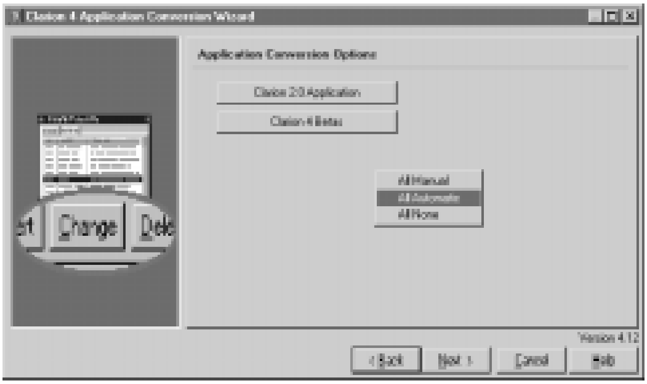
Press the **Next** button to open the **Application Conversion Options** page. This page lets you configure the conversion rules applied to the source application to create the target application.For each rule in each family (except the Hints rule), you may apply the rule automatically, manually or not at all.

None

The converter ignores the conversion rule.

Manual

The converter applies the rule interactively by showing you each affected instance of code and the proposed changes to the code. You have the option of ignoring, marking, applying, or editing and applying the proposed changes on an instance by instance basis. The *Manual* rule application is the teaching mode, and we recommend it for your initial conversions because it gives you the most control over the conversion process and the most information about the conversion process. You can see each change that is made (original source and proposed source) along with an explanation of *why* the change is needed. If you think the change is not needed or you want to postpone the change until later, you may do so. See *Dispose of Proposed Changes*.



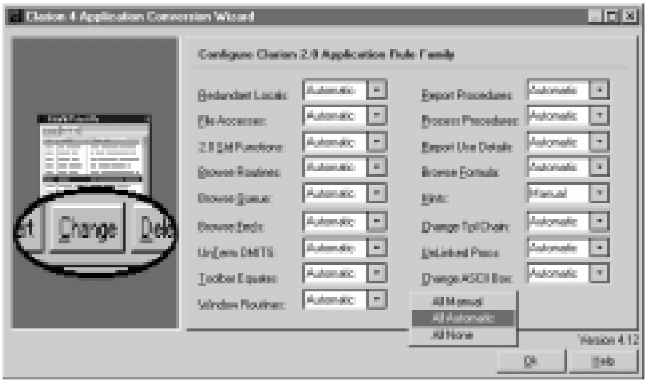
Automatic
The wizard unconditionally applies the rule as it is written.

Tip: RIGHT-CLICK on this page to configure *all* the rules in *all* rule families.

CW 2.00x to Clarion 5 ABC Rules

Press the **Clarion 2.0 Application** button to configure the Clarion 2.00x to Clarion 5 ABC conversion rules. Each drop-list represents a separate rule. Some of the rules such as *Browse Queue, Browse FEQ's* and *Toolbar Equates* apply new naming conventions or new references to data structures, while other rules such as *Files Accesses, 2.0 Std Functions,* and *Browse Routines*, replace calls to CW 2.00x procedures and routines with calls to their Clarion 5 ABC counterparts.

Tip: RIGHT-CLICK on this page to configure *all* the rules in the Clarion 2.00x to Clarion 5 ABC family.


Clarion 4 Application Conversion Wizard

Configure Clarion 2.0 Application Rule Family

Redundant Locals: Automatic
File Accesses: Automatic
2.0 Std Functions: Automatic
Browse Routines: Automatic
Browse Queue: Automatic
Browse Ends: Automatic
UnZero DMTS: Automatic
Toolbar Equates: Automatic
Window Routines: Automatic
Export Procedures: Automatic
Process Procedures: Automatic
Report Use Details: Automatic
Browse Formula: Automatic
Hints: Manual
Change Tpl Chain: Automatic
Unlinked Procs: Automatic
Change ASCII Box: Automatic

Version 4.12

The **Hints** rule does not offer an *Automatic* application, because it does not apply any changes. It only offers suggestions for ambiguous circumstances where it is not completely clear that a change is needed.

Press the **OK** button to return to the **Application Conversion Options** page.

Clarion 4 Betas Rules

Press the **Clarion 4 Betas** button to configure the Clarion 4 Beta to Clarion 5 Gold conversion rules. Each drop-list represents a separate rule. These rules are designed to convert Clarion 4 Beta1 and Beta2 applications to Clarion 5.

Tip: RIGHT-CLICK on this page to configure *all* the rules in the Clarion 4 Beta to Clarion 5 Gold family.

You need not apply these rules to Clarion 2.00x applications.

Press the **OK** button to return to the **Application Conversion Options** page.

Start the Conversion Process

Press the **Next** button on the **Application Conversion Options** page to open the **Start Conversion Process** page. This page explains that the rules are applied automatically, manually, or not at all. If all rules are applied automatically, the conversion proceeds with no further input. If some rules are applied manually, the Application Converter opens the **Confirm Conversion** window, where you can see the proposed changes, the reason for the proposed change, and you can decide how to handle each proposed change.

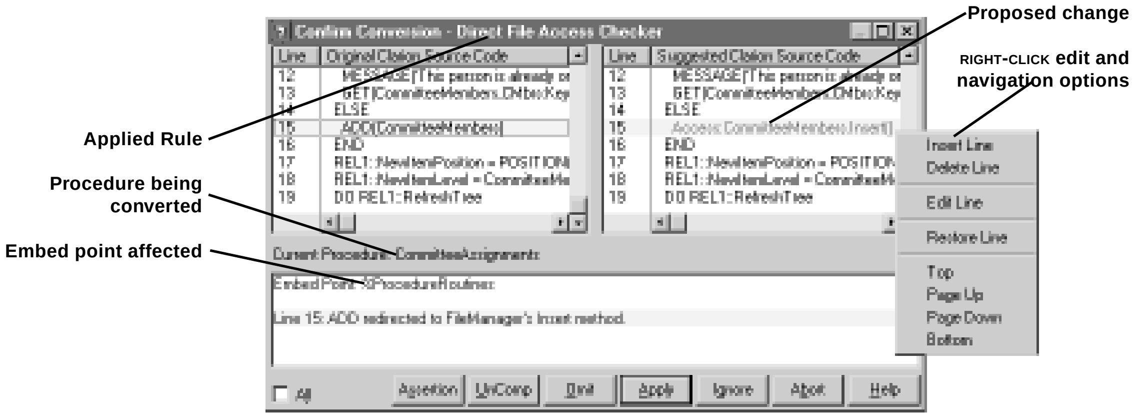
In either case, the Application Converter displays progress indicators to advise you of the conversion's progress. Press the **Proceed** button to start the conversion or press the **Cancel** button to stop now.

When the Application Converter has applied all the conversion rules, it creates the target file (.APP or .TXA). If the target file is an application, it loads the file in the Clarion environment so the application is ready to make and run.

> **Tip:** **The Application Converter cannot load the new application if its data dictionary is unavailable. Place a copy of the associated data dictionary in the target directory before you convert the applicaion.**

Dispose of Proposed Changes—Confirm Conversion Window

If any rules are applied manually, the Application Converter opens the **Confirm Conversion** window, where you can see the proposed changes, the reason for the proposed change, and you can decide how to handle each proposed change.



Line	Original Clarion Source Code
12	MESSAGE('This person is already or
13	GET(CommitteeMembers,CMbr:Key
14	ELSE
15	ADD(CommitteeMembers)
16	END
17	REL1::NewItemPosition = POSITION
18	REL1::NewItemLevel = CommitteeMe
19	DO REL1::RefreshTree

Line	Suggested Clarion Source Code
12	MESSAGE('This person is already or
13	GET(CommitteeMembers,CMbr:Key
14	ELSE
15	Access:CommitteeMembers.Insert()
16	END
17	REL1::NewItemPosition = POSITION
18	REL1::NewItemLevel = CommitteeMe
19	DO REL1::RefreshTree

Current Procedure: CommitteeAssignments

Embed Point: %ProcedureRoutines

Line 15: ADD redirected to FileManager's Insert method.

The **Confirm Conversion** window identifies the rule being applied as well as the procedure and the embed point the rule is applied to. In the **Confirm Conversion** window, proposed code changes are shown in red in the right-hand list box. The original code is in the left-hand list box. A brief explanation of each proposed change is in the bottom list box.

This window is resizable so you may enlarge it to reveal as much code as you want.

The proposed code changes are editable in a number of ways: you may click one of the buttons to dispose of a single proposed change or for all proposed changes for the rule. You can RIGHT-CLICK a specific line of code to edit or delete that line. You can even add new lines of code.

Choose from the following disposition buttons:

All Check this box to apply a button's action to *all* the proposed changes for this rule. Clear the box to dispose of each proposed change individually.

Assertion
The proposed (red) code replaces the original code in the new application. The converter inserts a runtime marker (ASSERT(False)) before the new code so the program offers to GPF prior to executing the new code.

Uncomp
The proposed (red) code replaces the original code in the new application. The converter inserts a compile time marker (***) before the new code so the compiler issues a message locating/identifying the new code, and so the program will not successfully compile.

Omit
The proposed (red) code replaces the original code in the new application. The converter inserts an OMIT statement and terminator around the new code so it is neither compiled nor executed.

Apply
The proposed (red) code replaces the original code in the new application.

Ignore
The original code remains in place in the new application.

Abort
Stops the conversion process without generating a new application. The Application Converter returns to the **Application Conversion Options** page so you can start over, or cancel.

Writing Your Own Conversion Rules

The Application Converter conversion rules are encapsulated from the reading, parsing, and writing of the application files so that conversion rules can be easily added, modified, and deleted.

Clarion 5 supplies two sets of conversion rules. One set of rules converts Clarion for Windows 2.00x applications to Clarion 5 ABC applications. The other set of rules converts Clarion 4 beta applications to Clarion 5 applications. You can use these rules as models for writing your own rules.

TopSpeed encourages you to write your own rules to convert applications based on your own templates and applications. Toward that end, the source code for the TopSpeed conversion rules and for the Application Converter are installed (by default) to the \Clarion5\Convert directory.

If you write your own rules, you should recompile them using the C5Conv.pr project file, and you should update the C5Conv.INI file (installed by default to the \Clarion5\bin directory) to include your new rule .DLLs.

Compiling Conversion Rules

If you write your own rules, you will need to compile them with the C5Conv.pr project file (installed by default to the \Clarion5\Convert directory).

You may make the C5Conv.pr project just as you would any other Clarion project. You should update the C5Conv.INI file (installed by default to the \Clarion5\bin directory) to include your new rule .DLLs.

APPENDIX A - WINDOWS DESIGN ISSUES

About This Chapter

This chapter provides an introduction to:

- The design principles which Microsoft suggests for designing the user interface of Windows-based applications.
- Event driven programming and how it should influence your application's design process.
- The types of windows, controls, cursors, and other objects common to Windows applications.
- The standard menus and menu commands recommended for Windows applications.
- The use of color and how it relates to the user.

Design Principles

If you make your program look and act like other Windows programs, your users will learn it more quickly, and feel more confident using it to accomplish the tasks you designed it to do. The Graphical User Interface (GUI) environment demands that you address your users on their terms. The program design must reflect that the user is in control, both visually and in the underlying structure of program flow.

Apple, in its *Human Interface Guidelines*, IBM, in its *Common User Access: Advanced Interface Design Guide,* and Microsoft, in *The Windows Interface: an Application Design Guide* have all published detailed design principles for software designers working in popular GUI environments. Creating a standard for program design offers these advantages:

- Given consistency between applications, users learn new applications more quickly and easily, minimizing the need for training.
- Consistency between applications increases the level of confidence in the user, resulting in increased productivity.

There are two especially important Windows design principles for the Clarion programmer to keep in mind when designing the user interface. The first is *user control*—providing a real-world based metaphor for the program's organization, and maintaining a consistent look and feel for all parts of the program, builds the user's confidence. The second is to remember that Windows programming is *event driven*—the user decides the next action. The programmer's responsibility is to provide a visual list of options the user can act upon.

User Control

Metaphors

Your users may not have years of experience using a wide variety of programs. Providing a *metaphor* from the real world will help provide a 'setting' or group of expectations to apply to your program. A word processing program, for example, uses a 'paper' metaphor—the document is like a piece of paper to write on, erase characters from, etc. Many database programs use Rolodex card metaphors. By establishing a relation to the real world, you increase the comfort level of your users, and actively engage them in the work of the program.

Visual Consistency

Visual consistency is very important. As much as possible, an application should use a single way to implement actions. A user learning a new procedure in your program builds upon prior experience with other procedures in the same program. Creating a standard look for your dialog boxes, and making screens for similar tasks look like one another, also reduces the time it takes you to design the different screens your program requires.

Directness

Directness makes a user feel they are in charge. Moving a document from one folder to another, or moving one icon to another, such as the Recycle Bin icon, can seem to a user to be a real action. Clarion provides drag-and-drop server support, which lets you create, for example, an icon with a picture of a person on it, representing an employee record. A user might drag the icon to another representing money, to open a dialog box displaying the employees's payroll records.

Simplicity

Simplicity is usually the best design. Cluttering the screen with too many windows, buttons, icons, lists and other objects can confuse the user. Dialog boxes especially, usually must fit in a small space, so their messages must be simple. The same is true for colors. Limit the use of colors to areas where they are needed to provide emphasis, such as red text for an important message.

Feedback

Let the user know what's going on—provide *feedback*. When the user chooses a command or begins an operation, visual feedback should confirm it is being carried out. The confirmation may be graphical, such as a cursor change, or simply a progress bar or message on the status bar. If there will be a delay while the program finishes another operation, inform the user, and advise, if possible, how long it will take.

Plain Language

Your application should use *plain language*. When designing an application for a corporate environment, technical terms are often essential. Watch out for the times when plain English is better. Of special importance to the programmer, who may be new to Windows programming—beware of "Window-isms." It's very easy to create two buttons marked "OK" and "Cancel." Yet "Yes" and "No" are far better button labels when the question is: "Do you wish to delete this record?"

Event Driven Programming

Related to user control issues, and the most important concept for new Windows programmers, is that you must design Windows programs to be *event-driven*. In a multi-tasking environment, a program simply cannot direct the sequence of program action—the user directs it.

The underlying structure of Windows is such that the operating system informs the program with messages just what it is the user wants the program to do. This is the opposite of DOS programming.

Windows messages tell a program when a menu is selected, when a window is selected, when the user wants to shut down the operating system—the Windows 3.1 Software Development Kit documentation devotes over 200 pages just to listing and explaining the different messages. Most programming languages require elaborate message handling procedures to branch program flow upon receiving different types of Windows messages.

Clarion automatically handles the housekeeping associated with message handling. The ACCEPT loop frees the Clarion programmer from worrying about system messages. Yet you still must consider the event-driven model when designing your program.

A Windows program must constantly look for input—in the form of data entry in an edit box, for example. In a window with several fields, the user can TAB or CLICK with the mouse to make any of them active. You must plan your input dialogs accordingly.

Additionally, your users will expect a complete set of user interface elements, including menus, multiple windows and graphical controls, *all available simultaneously*.

You may create an application which allows the user to open two windows, with markedly different functions. Clarion will automatically handle events generated within each window. Yet what about the menu commands, or the tool bar? You should plan these so that the commands will act on any active window. Clarion helps you do this automatically when creating an application using Multiple Document Interface. There may be times, however, when you may need to manually disable and re-enable menu commands.

Finally, the event-driven model should influence the windows you design for your application. Plan on using your main application window as the backbone for your program. Generate another window only upon some user action in the main window.

Background Processing

Background processing is another important concept for Windows programmers. In a multi-tasking environment, your program should always leave the user in control, especially when it is doing a lengthy process such as reading or writing a large file.

At the language level, Clarion supports background processing through the invocation and detection of TIMER events. That is, specifying a TIMER for a WINDOW tells the operating system to generate a timer event for the Window at regular intervals. The TIMER:Events may be detected and acted upon within the WINDOW's ACCEPT loop. Your program performs a small portion of a larger process for each timer event. The net effect is that your program can work concurrently with other programs that employ a similar technique.

> **Note: The YIELD statement accomplishes a similar result for programs that do not contain an ACCEPT loop.**

See the *Language Reference* for more information on TIMER, ACCEPT, EVENT(), and YIELD.

At the template level, all the Clarion templates automatically support background processing with the TIMER technique described above. The ABC Templates dynamically adjust records per cycle to optimize sharing of machine resources.

You can control the sharing of machine resources by adjusting the amount of time between timer events and the amount of processing that occurs for each timer event. For example, a small TIMER interval with a high volume of processing per interval hogs resources, whereas a large TIMER interval with a small volume of processing per interval lets the machine sit idle.

A reasonable TIMER interval ranges from 1/100 to 30/100 of a second depending on the process performed and the hardware doing the processing.

```
MyWindow  WINDOW(‘With a Timer’),TIMER(1)   !1/100 of a second
  ...
  CODE
  ...
  MyWindow{PROP:Timer} = 30                 !30/100 of a second
```

A reasonable number of iterations or LOOP cycles per interval may range from 1 to 1000, depending on the length of the timer interval, the process performed, and the hardware doing the processing. See the RecordsPerCycle variable generated by the Process template.

Your choice of 16-bit or 32-bit applications or 16-bit or 32-bit operating systems has no impact on the technique you use to implement background processing.

Windows

This section describes common window types. Choosing the right window for the right job helps your users get the most out of your program.

Multiple Document Interface (MDI)

Clarion enables Multiple Document Interface (MDI) programs. This manages multiple documents or multiple views of the same document, each in a separate window which appears inside the main application window. If the user resizes the application window to make it smaller than the document window, the document window will appear clipped.

There are three types of windows in a typical MDI program: application, document and dialog boxes.

Application Window

The *application window* should usually "contain" all the other windows your program may generate. If you open a browse window, it should appear inside the application window. If a user resizes an application window to a smaller size, you may allow your other windows to appear partially outside it. Additionally, you may allow users to move moveable dialog boxes outside the application window.

Application windows should always contain a title bar that contains the name of your application. Microsoft recommends that application windows also contain resizable frames, a Control-menu box, and Minimize and Restore buttons. In Clarion, you add these attributes to the window with the **Application Properties** dialog.

To create an application window with Clarion, you select *Frame - Multiple Document Main Menu* from the **Select Procedure Type** dialog, or select *MDI Parent Frame* in the **New Structure** dialog, or declare an APPLICATION structure in your source code.

Document Windows and Dialog Boxes

Document windows and dialog boxes are very similar in that they are both defined as Clarion WINDOW structures. They differ in the conventional context in which they are commonly used and the conventions regarding appearance and attributes. In many cases, the difference is not distinguishable and does not matter. The generic term for both document windows and dialog boxes is "window" and that is the term used throughout this text.

Document Windows

Microsoft recommends each document window contain a title bar with the name of the document, a system menu, and a Maximize button. Optionally, you can include a minimize button. Document windows usually display data. For example, in the Windows environment, the “Main” program group window that appears when you DOUBLE-CLICK on the “Main” icon in the Program Manager’s desktop, is a document window.

Dialog Boxes

The Windows design guidelines for dialog boxes are very flexible. They may be movable, or maintain a fixed position. They may have a single size, or a ‘**More >>**’ button to make it unfold and offer additional options. They may be modal or modeless. They may present a brief message with only an ‘**OK**’ option, or provide complex choices, controls, and entry options.

Here are a few pointers which will help you design windows (Document Windows and Dialog Boxes) that are easier to use.

- ❏ Using either the caption bar or static text in the window, briefly explain the function of the window, or indicate which command caused it to appear.
- ❏ Set as many controls to the default setting as possible, so that you require the least amount of entry on the part of the user.
- ❏ Place the most important information at the upper left, the least important at the lower right. Your users read from left to right, top to bottom—this is the natural way in which they expect to enter information.
- ❏ Set the focus to the first entry box. This lets the user type a word or words at the keyboard without repositioning the cursor.
- ❏ Place default buttons—the most likely user choice—on the right. This gives the user to opportunity to ‘read’ the choices on the left before presenting the ‘decision.’
- ❏ When presenting a brief message, take advantage of the default icons available with the MESSAGE function. The Stop, Information, and Question icons are familiar to users from other applications.

Window Elements

This section describes common window elements. Choosing the right element for the job at hand helps your users get the most out of your program.

Buttons

A button initiates an action. When the user presses a button, the button appears to be depressed. When a button action is unavailable, the button label should be dimmed.

Clarion lets you use text, graphical labels (picture buttons), or both. If you are using picture buttons you should include tool tips to allow the user to see, in words, what action the button will initiate. Stick to standard bitmaps (such as the icons many bestselling applications use for **File ➤ Open**, **File ➤ Save**, etc). Too many picture buttons in a window can be confusing to the user. Some reviewers have accused programs of "iconitis"—having so many graphical buttons on the screen at once that it's impossible for users to remember which button executes which action.

Buttons can perform several types of actions.

- A button may initiate an action.
- A button may close the active dialog box, then open another, related dialog.
- A button may open another dialog on top of the current one, without closing the current one.
- A button may "unfold" or resize the dialog window to include more options.
- A button can turn the "page" on a wizard dialog.

Always designate one button as the default. When the user presses ENTER, it will initiate the default button action. Microsoft also suggests that you *not* assign a mnemonic—for example, "**&OK**," which appears as **OK**—to a default button.

Check Boxes

Check boxes control true/false, yes/no, on/off or logical variables. The user toggles the state by CLICKING on the box, or pressing the SPACEBAR. If a check box option is unavailable, the label should be dimmed.

Radio Buttons

Radio buttons, also referred to as option buttons, present the user with a single choice in a mutually exclusive set of choices. The user may select only one at a time. If space is at a premium, and the number of choices is greater than four, consider a drop-down list box, which takes up less space.

List Boxes

List boxes display choices for the user. If a choice is unavailable, in general you should drop it from the list. If the choice is important enough that the user should know it is unavailable, Microsoft recommends it appear in the list box as a dimmed selection.

Always allow your list boxes enough room. Try to allow vertical space for three to eight choices; horizontal space for the average length of selection text plus several extra spaces.

Remember that a list box can present a user with a great deal of information—keep it simple!

Combo Boxes

A combo box is a combination of text box and list box. They are appropriate where the data lends itself to possible responses, but allows the user to type in a response not in the list.

The design guidelines for list boxes apply to combo boxes.

Drop-Down List Boxes

Drop-down single-selection lists perform the same function as list boxes, but take up less space. When closed, the drop-down is only tall enough to show one selection. Opened, the list will show more items, like a standard list box.

Novice users often have much more difficulty selecting an item from a drop-down than from a normal list box. Whenever space permits, use radio buttons (for four choices or less) or normal list boxes.

Text Boxes

Text boxes allow the user to type in information. They may be single line or multi-line. Multi-line edit boxes should usually provide a vertical scroll bar.

The standard Windows accelerator keys for copy, paste, etc., are active by default. This is useful, because it enables the user to copy, for example, a paragraph from another application, then paste it directly into a multi-line text box in your application. For this reason, we recommend you do *not* reassign the default windows editing shortcut keys—such as CTRL+C for copying or CTRL+V for pasting—to alternate commands in your Clarion application.

Fixed-length, auto-exit text boxes may speed up data entry. As soon as the user fills the text box (by typing the maximum allowable characters), the focus moves to the next control. Microsoft recommends applications use this type of text box sparingly, as the shift of focus may be disconcerting if it catches a user by surprise. We recommend using this type of text box when there are many fields to enter in a dialog. For dialogs with only a few fields, the programmer should try to anticipate what the end user will expect, and choose accordingly.

Clarion lets you select the font for text boxes. We suggest using the default system font. Microsoft specifically chose this font for menus and other system items because it is especially easy to read on a monitor.

Spin Boxes

Spin boxes are specialized text boxes with a pair of arrows (spin buttons) attached to the right of the text box. Spin boxes accept a limited set of values, which the user may type in, or by using the arrows, increase or decrease the value by a specified unit. Spin boxes can provide an alternative to drop-down lists when the set of values is limited in quantity and fits into a natural progression; for example, ‘Spring, Summer, Fall, Winter.’

Besides increase or decrease controls for simple numbers or choices, you may use spin boxes to manipulate values that consist of several components. You may display time in hours, minutes and seconds, for example. Be sure to visually separate each component with a relevant separator character, such as a colon.

Static Text

Static text should present read-only information, such as directions for entering data in the other controls in the dialog box. You should also use static text to label controls not automatically labelled by the Window Formatter, such as a text box.

Clarion lets you select the font for static text. We suggest using the default system font. Microsoft specifically chose this font for menus and other system items because it is especially easy to read on a monitor. You should certainly feel free to make the text bigger, as in creating a ‘title’ for a dialog box ‘form.’

We also advise you to resist the temptation to use odd or too many different colors for static text. You never can tell what the default window background will be.

Group Boxes

Group boxes provide a visual grouping for related controls. They consist of a rectangular frame with a label at the upper left.

A group box can guide the user directly to the controls that are most important to the task at hand. If your application requires a dialog with more than ten controls or fields, we highly recommend taking a moment to consider whether some of the controls fit into logical groups.

Sheets and Tabs

The Property Sheets with tabs provide another method of grouping related controls, by allowing you to place controls with similar or related functions on separate “pages.”

Tabbed dialogs can “flatten” your application, by reducing the number of visible controls and displaying only those that are most important to the task at hand. If your application requires a dialog with more than ten controls or fields, you should consider a “multi-page” approach.

Keep in mind that Required Entry fields should be on the first visible tab.

Wizards

The WIZARD attribute on a Property Sheet control lets you control the user’s movement through the tab “pages.” This lets you present a series of dialogs in a linear fashion. Optionally, you can control the next “page” based on the answers the user provides in previous pages.

Wizards have become increasingly popular because they let users answer only one question at a time, decreasing the chances of confusion or error.

Control Labels

The Window Formatter automatically supplies labels for many, but not all controls. You may supply labels for the other controls using static text. Not only will this identify the control for the user, but it also will allow keyboard users to quickly direct the focus to the control.

When the user keys in the mnemonic (such as the "S" in '**Daily Sales**'), Windows automatically directs the focus to the *next* control after the static text label. Thus, you may place 'Daily &Sales' to the left of a drop-down box. When the user presses ALT+S, the combo box will receive the focus. The keyboard user will merely have to press the DOWN ARROW key to view the choices in the list.

Microsoft suggests the following guidelines for control label text:

❏ Capitalize the first letter of each word, except for articles (e.g., a, an, the), coordinate conjunctions (e.g., and, or, for), prepositions (e.g., by, with) or the word "to" in infinitives.

❏ Try to use the first letter of the first word for the mnemonic. Since the mnemonics need to be unique, however, this isn't always possible. Alternately, use another letter if it allows a stronger mnemonic link: such as the "x" in "**Exit**". If the first word in the control label is less important than another, use the other (e.g., "**by Ascending Order**").

❏ Choose consonants over vowels: they are more distinctive and more easily remembered.

Microsoft also suggests the following positioning for control labels for the following controls:

- Command buttons: inside the button.
- Check boxes, radio buttons: to the right of the control.
- Text boxes, spin boxes, lists, combo boxes: above or to the left of the control. Place a colon after the last word of the label. Left align the label with the section of the dialog box in which it appears.

Cursors

In a graphical environment such as Windows, the mouse cursor, or screen pointer, is the means by which the user shows the application what to do next. For example, the I-bar, or insertion point, may tell a word processing application where the next characters typed by the user should appear. This is a key part of the 'event driven programming' referred to earlier in this chapter.

Though Microsoft has not set specific guidelines for the use of each system cursor, the following uses have evolved into standards across GUI platforms:

Arrow: selects controls and menu commands.

I-beam: selects and inserts text.

Crosshairs: draws and manipulates graphics.

Plus sign: selects fields in an array.

Hourglass: shows that a lengthy operation is in progress.

Menus

Menus display the range of commands available for the user to execute. Windows users are accustomed to standard menus and commands which appear in many different applications. If you use these same menus, new users may learn your application more easily, and the sense of familiarity will increase all users' confidence and productivity levels.

When designing additional, custom menus and commands, bear in mind that the model for GUI design is the 'Noun-Verb.' principle. Apple fancifully refers to this as 'Hey you—do this!'

In the 'Noun-Verb' model, the user points to something—for example, an on screen object such as text. This is the noun. The model assumes that the next command action the user chooses will tell the application what to do to the noun. The action is the verb. If you word your menus and commands in a way that the menu - command is a short 'do this' sentence—such as "**Insert ➤ Record**," "**View ➤ Transaction**," "**List ➤ Activity**," or "**Select ➤ Current Group**," your menus will gain added clarity.

This guideline should not severely limit you. There are times when it is most appropriate to use a single menu item to initiate complicated instructions, such as bringing up a dialog box with many different preferences and options for the user to set. When doing so, add an ellipsis (**...**) following the last word of the menu command.

The following discusses the standard menu implementations recommended by Microsoft:

File Menu

Many database applications do not naturally lend themselves toward allowing the user to open and close external document files. In the simplest case, a database or databases open automatically with the application, and user editing is limited to editing individual records. Clarion programmers may wish to limit commands on the file menu to those that affect the global operation of the application—**Printer Setup** and **Exit**, in the most extreme case. At the very minimum, your Clarion applications should have a **File ➤ Exit** command: this is how users expect a Windows application to allow them to quit the program.

New or **New...**

Creates a new file with a default name such as 'Untitled.' This is sometimes a problematical menu item when creating database applications. Unless your application allows the user to create a new database, or creates separate editable external files (such as text files) Clarion programmers may wish to drop this command.

Open

Displays the **File Open** dialog, from the Windows common dialog library. Allows the user to open external files.

Save

Saves the active document. For a new file, calls the **Save As** dialog.

Save As

Displays the **Save As** dialog, from the Windows common dialog library.

Print or **Print...**

Prints the active document, or leads to a dialog allowing the user to set print options.

The **Print** command can be an interesting one in a database application. Many times, a database application allows the user only to print pre-formatted reports.

Other ‘docu-centric’ Windows applications may simply go ahead and print the current document in its entirety—but a database application can hardly be expected to print a 30,000 record database as the default print preference.

One solution some popular applications use is to drop the print command entirely and provide a separate **Report** menu. This is a good solution for an application with a limited number of reports. Alternatively, an application with a limited number of reports might also use a cascading menu, attached to the **File ➤ Print** command.

For an application with a large number of pre-formatted reports, one solution might be to present a list box in a dialog window when the user selects the **File ➤ Print** command.

Print Setup

Displays the **Printer Setup** dialog, from the Windows common dialog library.

This dialog allows the user to change the active printer and/or specify settings for the selected printer.

Exit Closes all application windows and terminates the application. If don’t have a **File** menu in your application, place your **Exit** command on the leftmost application menu, as the last command on the menu.

Edit Menu

The Edit menu usually provides commands for reversing the user’s last action, plus the clipboard editing commands: cut, copy and paste.

Undo
The **Undo** command should reverse the user's last action. It must always be the first command on the **Edit** menu, if your application supports undo.

Clearly, database programs present special problems for Undo. In general, Windows applications disable the **Undo** command after a file operation, such as when a **File ➤ Save** command saves an edited document to disk. A database application may easily present a situation in which it writes data to disk every few seconds when, for example, a user enters a group of new records.

Cut Transfers a selected object to the clipboard and deletes it from the field.

Copy
Places a copy of a selected object in the clipboard.

Paste
Places a copy of an object previously placed in the clipboard into the current field.

Clarion automatically enables clipboard support for **Cut**, **Copy** and **Paste** when in an edit box. The default accelerator keys for these actions are CTRL+X, CTRL+C and CTRL+V respectively.

View Menu

Microsoft defines the View menu as optional, and states that it includes commands for changing how the program presents the data to the user, without changing any of the data. As such, it presents a natural means for a database application to allow different browse options on a single database.

The View menu may also present options for displaying various interface elements such as toolbars, status bars, and other special controls that are part of the application window. There are no specific command text suggestions for the View menu.

Window Menu

This is an optional menu. If you choose to support the Multiple Document Interface (MDI) in your application, the Window menu allows the user to manipulate entire child windows.

The commands for this menu are flexible. Common commands include:

Tile Arranges child windows end-to-end, so that all are visible.

Cascade

Arranges child windows in an overlapping fashion, so that the title bar of each is visible.

The Window menu may also contain a numbered list of up to nine open child windows. A number should precede each child window name. When the user chooses a window from the list, the window should receive the focus.

Help Menu

The Help menu provides the user with access to the Windows Help system. It should always be the last menu on the right. The Help menu usually contains the following commands:

Contents
Loads the Windows Help system, then opens the external Help file to the main contents page.

Search for Help On
This loads the external Help file, then automatically opens the Search dialog. This allows the user to type in a word; if the word appears as a topic title, the Help system jumps to the title.

How to Use Help
This opens the Windows Help system, and displays the instructions for using it. The file “*WINHELP.HLP*,” which Windows automatically installs, contains the instructions.

Accelerator Keys

A number of commands have gained standard accelerator (or alert, or hot) keys. When creating your application, should you use any of the following commands, we recommend you use the following keys:

Command	Accelerator
File ➤ New	CTRL+N
File ➤ Open	CTRL+O
File ➤ Save	CTRL+S, or SHIFT+F12
File ➤ Exit	ALT+F4
Edit ➤ Undo	CTRL+Z
Edit ➤ Cut	CTRL+X
Edit ➤ Copy	CTRL+C
Edit ➤ Paste	CTRL+V
Edit ➤ Select All	CTRL+A

Color

Color can greatly affect how your user works with your application. Microsoft does not publish standard guidelines on color usage—yet. When designing your application, the following guidelines may help you:

- ❏ Windows allows users to select default colors for window text and background. It’s best to accept these default colors for the parts of the program which require the most data entry: the user has expressed a preference, so you should respect it!
- ❏ Without forgetting the first point, you may choose to accentuate windows and screen elements by using color. Color can set off specific areas in a window—it can be more effective than a group box.
- ❏ Use color to discriminate between different parts of your program. For example, you may associate one window background color for dialog boxes related to accounts receivable data entry, and another for payables.
- ❏ Use color to visually relate similar parts of the program. For example, you may associate one window background color with phone number data.
- ❏ Use standard cultural associations for special alerts. In western culture, the most ‘meaningful’ colors are probably the ones on the traffic lights: red, yellow and green. You may use red to signal a halt in a procedure. You may use yellow to signal a warning. Green, of course, means go, all clear.
- ❏ When adding color to text elements, remember that most colors look best against a neutral grey background. If you don’t use grey, be sure there is a high contrast between the text and the background color. In dim lighting, color tends to wash out.
- ❏ Bear in mind that 8% of males in Europe and America have some degree of color blindness. The most common type reduces the ability to distinguish red and green from gray. In a less common type, the user cannot distinguish between yellow, blue and gray.
- ❏ Remember that on monochrome LCD screens, light blue is very hard to distinguish from gray and white.

APPENDIX B - MAKING API CALLS

Overview

This appendix provides an introduction to Application Programming Interface (API) calls. This appendix is meant only to provide an outline of what API calls are, what they do, and to provide some basic examples to get you started.

API calls provide a method for your program to call functions external to your application. In other words an API call is simply calling a function from someone else's dynamic link library (DLL). A DLL is a file that contains executable code that is linked into your .EXE *at run-time*. You can use API calls to implement Object Linking and Embedding (OLE) and multimedia processing in your applications.

Generally, making API calls from Clarion involves two steps: *prototyping* the API functions, and *linking* the API functions into your program. However, many Windows API calls are already linked for you. See *Linking API Functions* below.

Prototyping API Functions

Each API function you wish to call must first be prototyped in the Clarion MAP structure. Functions written in a language other than Clarion can be referenced in a Clarion program by creating an equivalent Clarion prototype.

The prototypes are placed in a MODULE structure which identifies the name of the DLL's library as the MODULE parameter. For example, if the DLL name is WIN32.DLL then the module structure and prototype for the *GetWindowsDirectory* function is:

```
MAP
  MODULE('WIN32.LIB')
    GetWindowsDirectory(*CSTRING,USHORT),USHORT,RAW,PASCAL
  END
END
```

In order to proceed with your prototyping, you will need a technical reference describing the DLL's functions, purposes, and parameters.

For Windows API calls, we have provided some prototype examples in \LIBSRC\WINDOWS.CLW as well as the WINAPI.APP in the \RESOURCE\WINAPI folder. If you need to call the Windows API from your Clarion application or program, the WINAPI program can help you build an include file with the necessary Windows API prototypes and associated data declarations. The prototypes use the OMIT compiler directive to accomodate transparent toggling between 16-bit and 32-bit applications.

You may also want to read *How to use DLLs not created in* Clarion in the *How Do I...?* section of Clarion's on-line help. See also *Procedure Prototypes* in the *Language Reference.*

There are several issues to consider when creating Clarion prototypes which depend upon a DLL's source code language. A primary consideration is finding equivalent data types between the two languages. You can determine equivalent data types by considering the underlying machine representation of the data. For example, the Clarion data type SREAL stores a four-byte signed floating point in Intel 8087 format, while a BFLOAT4 stores a four-byte signed float in Microsoft Basic format.

Here are some Clarion and C or C++ data type equivalents:

C/C++	Clarion
`unsigned char`	`BYTE`
`short`	`SHORT`
`unsigned short`	`USHORT`
`long`	`LONG`
`unsigned long`	`ULONG`
`float`	`SREAL`
`double`	`REAL`

```
struct {                      Struct1 GROUP
    unsigned long ul1;      ul1       ULONG
    unsigned long ul2;      ul2       ULONG
}   Struct1;                          END
```

A second important prototyping consideration is the function calling convention used by another language. Clarion provides support for three different calling conventions: PASCAL, C, and TopSpeed's Register Based.

Linking API Functions

In order to call an API function, you must first link the function into your program. This can be accomplished in several ways. Some functions (WIN16.LIB and WIN32.LIB) are automatically linked. Other .DLL functions must be explicitly linked from a corresponding .LIB file. Finally, functions can also be dynamically linked using Clarion’s CALL function.

Windows API Functions

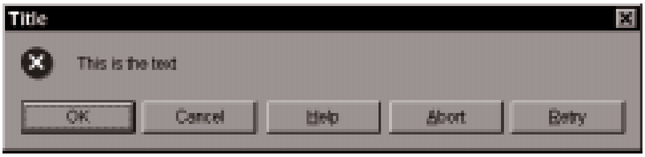
From a Clarion Language perspective, API calls can be divided into two categories: Windows API calls and other API calls. Windows API calls are calls to the functions that live in the three main Windows libraries (USER.EXE, GDI.EXE and KERNEL.EXE).

Because the Clarion Language makes extensive use of Windows API calls, many Windows API functions are *already linked* into Clarion’s run-time libraries. This means you can make Windows API calls from your Clarion programs simply by prototyping and calling. The linking is already done for you.

Clarion ships \LIB\WIN16.LIB and \LIB\WIN32.LIB in this package so references to these functions can be resolved during the compile and link process. Many windows-based technical references or *Windows API Bible* can provide information on the functions available in these Windows libraries.

Here is an example of how to call a Windows API function:

```
  PROGRAM
  MAP
    MODULE(‘WIN16.LIB’)
      MessageBox(USHORT,*CSTRING,*CSTRING,USHORT),PASCAL,RAW
    END
  END

Caption         CSTRING(18)
MessageText     CSTRING(32)

  CODE
    Caption = ‘Title’
    MessageText = ‘This is the text’
    MessageBox(0,MessageText,Caption,30)
```



Title

This is the text

Other API Functions

For API functions not in WIN16.LIB or WIN32.LIB, you must have a library file (.LIB) that corresponds to the .DLL. Once you have a .LIB that corresponds to the .DLL, add the .LIB to your Project File so the linker can resolve the external reference during the compile and link process. Prototype the functions your application calls, then compile and link as usual. The function can then be dynamically linked into your program at run-time.

Creating a .LIB from a .DLL

You can make and use a .LIB that corresponds to a .DLL as follows:

1. Create an Export (.EXP) File for the DLL.
2. Create a Library (.LIB) File for the DLL.
3. Reference the Library (.LIB) File in the Project System.

Create an Export File for the DLL

The TopSpeed Tech Kit includes a program (TSIMPLIB.EXE) that can be used to create .LIB files from .DLLs. The TopSpeed Tech Kit ships with TopSpeed C, C++, Modula-2, and Pascal.

You can also extract the set of accessible DLL function names from a DLL by using the EXEHDR.EXE DOS command line utility program. This program appears on most DOS diskettes earlier than version 6.0. If this utility program is not available then some other utility program or method may be substituted which provides the same list of names.

Append the extracted function names (stripped of any surrounding text) to the export file header information provided below. Substitute the appropriate DLL name for the word “dllname” on line 1 of the header information. Save the Export file under the same file name as the DLL with the extension .EXP.

```
——————— Start of EXPORT File ————
LIBRARY dllname
CODE MOVEABLE DISCARDABLE PRELOAD
DATA MOVEABLE SINGLE PRELOAD
HEAPSIZE 1024
STACKSIZE 32678
SEGMENTS
     ENTERCODE MOVEABLE DISCARDABLE PRELOAD
EXETYPE WINDOWS
EXPORTS
  function and function names go here (one name per line)
——————— End of EXPORT File —————
```

Create a Library .LIB File for the DLL.

Once you have created an Export (.EXP) file, you can create a .LIB file for the DLL using the #implib project system command. The #implib command creates or updates a Library (.LIB) file based on the information contained in an Export (.EXP) file. The command's syntax is:

```
#implib   <library file name>   <export file name>
```

where

<*library file name*> is the DLL name with the extension .LIB

<*export file name*> is the DLL name with the extension .EXP.

Using a text editor, create a Clarion Project File (.PRJ) and enter a single line in the file containing the #implib project system command with the appropriate parameters. Note the #implib must be in lowercase. Save the project file under an appropriate name (i.e. the DLL file name with the .PRJ extension). See the *Programmer's Guide* for more information on #implib.

Under Clarion, set the project file you just created as the current project:

1. Choose **Project ➤ Set**.
2. Select the project file and press the **OK** button.
3. Make the project by pressing the Make button on the Toolbar.

If the Make was successful and the Library (.LIB) file was created then a confirmation window appears with a green check mark in the bottom-right corner and appropriate completion messages display. The Library (.LIB) file is ready to use.

Reference the .LIB File in the Project System.

Place the Library (.LIB) file in the Project Tree (under 'Library and Object files') of any Project when you use the associated DLL's functions. During the link phase of the Make, the linker recognizes any referenced functions in the Library (.LIB) file.

CALL

If you do not have a .LIB file that corresponds to the .DLL you wish to access, and cannot make one, the Clarion CALL procedure gives limited access to .DLL functions without explicitly linking the function. There is more overhead incurred with the CALL procedure than with calling the API function directly. CALL uses an intermediate API function to access the target API function, and requires that you know where in the .DLL the desired function resides. You cannot pass parameters to the CALLed function, nor can it return any values. See the *Language Reference* for more.

APPENDIX C - DEVELOPMENT AND DEPLOYMENT STRATEGIES

Overview—EXEs, .LIBs, and .DLLs

The way you organize your application files can have a significant impact on the efficiency of your work processes throughout the life cycle of the application. For example, developing all of your procedures within a single .APP file keeps everything under one roof. This can be a benefit for smaller applications—keeping all files in a single directory makes finding and backing up the files quite easy if there are reasonably few files. This benefit can become a problem for larger applications—backing up or compiling hundreds of files at once can be time consuming and tricky. So you can see how organization affects the development phase of your application.

The ultimate size and number of your application's executable files affects your ability to quickly and easily distribute upgrades and bug fixes to your end users. This is one way that organization affects the maintenance phase of your application.

This appendix discusses of some of the factors you should consider when deciding how to structure your application files, both for the development and maintenance phases of the application. This includes instructions on how to implement the organization that best suits your needs.

Generally speaking, the benefits of breaking large programming projects into smaller logically related pieces are:

Development Phase

- Smaller, more manageable problems.
- The ability to test and debug smaller pieces of code. The smaller the code, the easier it is to isolate problems.
- Faster compile and link times.
- The ability to genericize and reuse code.
- The ability to delegate programming tasks to multiple programmers.

Maintenance Phase

- The ability to sell and distribute discrete application components.
- The ability to deploy bug fixes and upgrades with small files.

- For reused code, the ability to affect many procedures by changing a single source file.

These benefits are most easily realized by creating multiple .APP files that are used to make separate .DLLs or .EXEs that are developed and tested independently. As the project nears completion the executable files are linked together and tested as a whole.

Some problems associated with breaking large programming projects into smaller pieces include:

Development Phase

- Managing more files: backing up, naming conventions, and synchronization of files becomes a bigger job.
- Correctly linking together related pieces of executable code.

Maintenance Phase

- Managing more files: version control is more difficult. The end user must have a complete set of compatible files.
- For reused code, unintentionally affecting many procedures by changing a single source file.
- Accidentally omitting a required file during deployment.

Multi-Programmer Development

Clarion modular approach to source code management, its procedure-oriented language, and its ability to produce .DLL and .LIB files allows your team to split the work on big programming projects.

Our recommended methods for group development assume the team is linked by a LAN which supports the ability to grant read-only or read-write privileges to individual developers. It doesn't matter whether the LAN is peer-to-peer or a more traditional network operating system. We also assume the project will have a Team Leader or Project Manager to coordinate the overall efforts of the team. Finally we assume, as per our license agreement, each Clarion programmer has a licensed copy of Clarion.

The first step to prepare for a team-development project is to create a data dictionary available to all developers, but which only the team leader may edit. An Application Generator option (**Multi user Development** check box under **Setup ➤ Application Options**) provides support for opening the Data Dictionary and REGISTRY.TRF in read only mode, so that many developers working with separate .APP files can work with the same dictionary. Prior to beginning the project, all team members should synchronize their REGISTRY.TRF and their template source files. The Team Leader should be responsible for the dictionary and the template set.

Once the data dictionary is created, there are three basic approaches your team can take to use Clarion as a group development tool:

- Procedure-oriented:

  The team divides the application into procedures, as listed in the Application Tree. These should be organized around the various windows, dialog boxes, menu items, and command buttons that form the user interface.

  The Team Leader prepares a “shell .APP,” (or master) upon which all the others build. Each team member receives a copy of the .APP file, then works on a procedure (or procedures). The Team Leader imports the completed procedures into the master .APP file for compiling. This approach is suitable for small to medium size projects.

- Module-oriented:

  The team divides the application into its target-file-level components (.DLL's, .LIB's, and executables). Each team member creates a single target file. Separate project files (.PRJ) compile the individual components. A master project file may include all the other project files, building all target files at once. This approach is suitable for medium to large size projects.

- Sub-Application:

  The team divides the application into its target-file-level components (.DLL's). Each team member creates a single application or Dynamic Link library (.DLL). A master application calls each .DLL. This approach is suitable for medium to large size projects. This method provides the most flexibility and minimizes version control concerns.

Enabling and Organizing Team Projects

This section describes how to set up the Clarion Development Environment at each workstation, and where to store the files necessary for all three group development approaches:

1. Create the data dictionary in a shared directory.

   All team members working on the project must have read rights to the directory. Those permitted to edit the dictionary should also have write privileges, though it may be best that only the Team Leader be allowed to edit it.

2. Create a shared directory for resource files.

   Provide read rights for all team members to icon, cursor, bitmap, and other resource files.

3. Within Clarion, at each workstation, choose **Setup ➤ Application Options.**

   This opens the **Application Options** dialog.

4. Check the **Multi User Development** checkbox.

   This specifies that when working in the Application Generator, the copy of Clarion residing at each workstation opens the dictionary file on the network in read-only mode. The purpose is to ensure that no one accidentally deletes a field, file, or key needed by other team members. For this reason, we recommend that only the Team Leader have write privileges to the directory containing the dictionary. To modify the dictionary, all team members must close all applications which use the dictionary. The Team Leader must clear the Multi User Development box in order to modify the dictionary and ,upon completion, check the box again.

5. Press the **OK** button to close the dialog.

6. Create a directory on each workstation's local drive to hold each team member's individual .APP and source files.

   The real work is planning how to split the development project, which is what the remainder of this chapter discusses.

> **Note: If your application's .DLLs use your application files, the FILE definitions and all Global variables must be declared in a .DLL (not the .EXE) and exported. See the *Sub-Application Approach* for more information.**

Procedure Oriented Approach

The Application Generator lets you import and export procedures from other .APP files. With careful management, a Team Leader can organize development so that each team member can compile and test a copy of the application which includes the parts he or she works on. Each views the entire menu and the application's most important dialog boxes, yet executes only the procedures for which that team member is responsible.

> **Note: This approach is only necessary If your team members are not using Enterprise Edition, or do not have Version Control/team Developer.**

To accomplish this, each team member requires a *copy* of a "master" .APP file, containing the MAIN procedure (which would most likely be an Application Frame procedure), plus other procedures inserted below it as "ToDo" procedures. Each team member then "plugs in" the procedures he or she is responsible for.

To assemble the complete application, using the **File ➤ Import from Application** command, the Team Leader imports each finished procedure into the master .APP file.

The following outlines a possible implementation of the procedure oriented approach:

1. Create the data dictionary and set up the workstations as described above.
2. Create a "master" .APP file in a directory to which only the Team Leader has write privileges.
3. Within the .APP file, edit the MAIN procedure's most important user interface elements and declare its global variables.

   The user interface elements may include any dialog boxes or windows of particular importance to the application. As you specify procedure calls to menu items and/or toolbar controls, the Application Generator automatically adds "ToDo" procedures the application tree.
4. Save and copy the .APP file to each team member's local drive.

   If the team prefers, you can rename each copy; for example, MASTER01.APP, MASTER02.APP, etc. or JIM.APP, JANE.APP, etc.

5. Team members work on the procedures for which they are responsible, using their own copy of the .APP file.

   With the .APP file containing the complete user interface, each team member can compile an interim build locally, to test their own procedures while under development.

6. Each team member synchronizes their local directory with an equivalent directory on the network at the end of each work session, or copies renamed .APP files to a "master" directory.

7. To update the master .APP file with the latest work from a developer, the Team Leader replaces a "To Do" procedure in the Application Tree with a completed procedure in a team member's .APP by importing it. The Team Leader chooses **File ➤ Import from Application**, indicating the same procedure in the .APP file in the developer's network directory.

   Any sub-procedures added by the team members will be brought along as new "To Do" procedures. When the Team Member completes these, they can be imported in the same manner. As the Team Leader's master .APP file "grows", it can be copied back to team members' individual directories (but only if *all* the work done by the individual team member was imported). This way, each team member has access to all the work completed by other members of the team. Keep in mind that each of the other member's modules will need to be compiled on the member's local drive.

   If the Team Leader is also a team member—i.e., also responsible for coding procedures—it's best to maintain a completely separate directory and copy of the master .APP file for that work.

8. After importing the updated procedure, the Team Leader checks to see if it added any new "To Do" procedures to the tree, and imports those, if ready.

   Communication at this step is vital. In fact, based on E-Mail messages within the team, the Team Leader could optionally import "works in progress."

9. The Team Leader compiles the project, so that it now includes each team member's work added through importing procedures.

10. The Team leader repeats the last three steps on a periodic basis until all work by all team members is complete, and the entire application can be tested.

Module Oriented Approach

With this approach, each team member creates a separate target file. This requires splitting the application into a "Main" executable and "secondary" executables or dynamic link libraries. The individual team members maintain separate project files (.PRJ) for each component. The Team Leader creates a master project file to build all target files at once.

The key to successfully implementing this strategy is extensively preplanning the “division of labor” between the various target files created by the application. The Notes section below provides a few helpful suggestions.

The following outlines a possible implementation of this strategy:

1. Create the data dictionary and set up the workstations as described above.
2. Each team member creates their own .APP and .PRJ files, specifying the dictionary file on the network as the data dictionary, and a directory on the local drive as the default directory for the .APP file. Each team member specifies a different target file.

   One particular .APP or .PRJ file creates the executable which launches or calls library functions or procedures in the others. To the end user, this is the .EXE program to start when working with the complete application.
3. Each team member synchronizes their local directory with an equivalent on the network at the end of each day.
4. The Team Leader creates a master .PRJ file which includes all the other .PRJ files, in a network subdirectory.

   The Team Leader inserts the name of each .PRJ file (previously copied to the network) in the **Projects to Include** item in the Project Tree.
5. The Team Leader compiles the master project, which in turn compiles all the target files one by one.
6. The Team leader repeats the last step on a periodic basis until all work by all developers is complete, and the entire application can be tested.

Notes on Splitting the Project

There are probably as many ways to split a project as there are projects; this section provides a few general suggestions.

- If a task associated with a menu command requires extensive coding, store it in its own external .DLL, so that only a single developer can work on it.

  A typical example might be an accounting program, which could store all procedures and functions associated with accounts receivable in one .DLL file, accounts payable in another, and so forth.

- Organize .DLL’s by function; for example, place utility procedures and functions such as backups and file exports in a UTILITIES.DLL.

- Store user defined functions in .LIB files; distribute the compiled .LIB files to each team member as they become available so that each may test any functions required in their own work.

Notes on File Management

Each multi-developer project has its distinct properties, so you'll undoubtedly adapt the following suggestions to fit your needs:

- Create a subdirectory for each team member on the network drive, either at the same level or below the one holding the data dictionary file. Give each developer write privileges only to their own directory, and use a network utility to synchronize the directories at the end of the day.

  This not only serves as a backup, but provides the Team Leader access to the latest work done by all members of the team.

- If the application under development creates an .INI file, a copy of it should reside in a network directory to which all team members have write privileges, so that if anyone should need to add a variable to the file, other members of the team can see it.

Sub-Application Approach

This section describes the steps to create a program using one main application and several sub-applications compiled and linked as external .DLLs. Dividing a large project into multiple .DLLs provides many benefits:

- Each sub-application can be modified and tested independently.
- Developers can work on their portion of the project without interfering with others on the development team.
- Each sub-application can be compiled as a .DLL and tested in the main program without recompiling the entire project. This reduces compile and link time.
- Dynamic Pool Limits are avoided in large projects.
- Future updates can be deployed by shipping a new .DLL, reducing shipping costs.

> **Tip: The Clarion runtime libraries assume the .EXE or .DLL where a window was most recently opened is where any referenced icons are located.**

With this approach, each Team Member creates a separate .DLL that is called by a "master" application. This requires splitting the application into a "Main" executable and "secondary" .DLLs. The individual team members maintain separate application files for each component. The Team Leader creates a master application that calls the sub-applications and a "data" application that contains (and exports) all the File definitions and Global variables. Optionally, members can call procedures from another member's .DLL.

This method also requires extensive pre-planning of the "division of labor" between the various target files created by the application. The previous section provides a few helpful suggestions.

The following outlines a possible implementation of this strategy:

1. Create the data dictionary and set up the workstations as described above.

2. Create a "data" application to store and export all data declarations. All Global variables or structures and all file definitions are defined (and exported) in this application. Use the following settings:

   In the **Application Properties** dialog:

Dictionary File:	master dictionary
First Procedure:	none
Destination Type:	.DLL

   In the **Global Properties** dialog **General** tab**:**

Generate template globals and ABCs as External:	OFF

   In the **Global Properties** dialog **General** tab**:**

Generate All File declarations:	ON
External:	NONE EXTERNAL
Export All File declarations:	ON

Note: You do not need (and should not have) OWNER, NAME or PASSWORD on those FILES which have the EXTERNAL attribute. These attributes should only be on FILEs not declared EXTERNAL.

3. Team member create their own sub-application .APP files, specifying the dictionary file on the network as the data dictionary, and a directory on the local drive as the default directory for the .APP file. Each team member specifies a different target file using the following settings:

   In the Application's Module Tree:

   Choose **Application ➤ Insert Module,** then in the **Select Module Type** dialog, select **ExternalDLL,** then in the **Module Properties** dialog, select the corresponding .LIB for the .DLL containing the data definitions.

   In the **Application Properties** dialog:

Dictionary File:	master dictionary
Destination Type:	**.EXE** for development **.DLL** for release

Note: **Changing the Destination Type enables procedures to be exported. Make sure that every procedure that is called by the master application or another .DLL has the Export Procedure check box in the Procedure Properties checked (the check box is only available after changing the destination type).**

In the **Global Properties** dialog **General** tab:

Generate template globals and ABCs as External:	ON

In the **Global Properties** dialog **File Control** tab:

Generate All File declarations:	OFF
External:	ALL EXTERNAL
All Files declared in another .App:	ON
Declaring Module:	Leave this blank

In the **Global Properties** dialog **External Module Options** tab:

Standard ABC LIB/DLL?	ON

One particular .APP creates the executable which launches or calls library functions or procedures in the others. To the end user, this is the .EXE program to start when working with the complete application.

4. Team members synchronize their local directory with an equivalent on the network at the end of each day.

5. Team Members release their compiled and linked .DLLs to the Team Leader.

   Each sub-application has a "dummy" frame (not exported) that calls the sub-application's procedures so the Team Member can test the sub-application by compiling it as an .EXE. Once it passes testing, the member compiles it to a .DLL by changing the Application Properties' Destination Type to .DLL and releases the file to the Team Leader.

Tip: **If you edit the Redirection file to include "." at the start of the *.DLL and *.LIB search paths, Clarion generates the *.DLL and *.LIB files into the local sub-application subdirectory instead of \CLARION5\BIN and \CLARION5\OBJ. This is a little safety precaution that prevents the *.DLL and *.LIB from getting into other Team Members' hands before it's ready. In addition, adding the Master directory to the end of these search paths enables the sub-application or main application to find the completed .LIB's and .DLL's belonging to other sub-applications in the master subdirectory.**

6. The Team Leader copies the released .DLLs into the master directory and creates a master .APP file which calls the entry point procedures in the .DLLs.

The Master .APP is typically just a bare bones application with just a splash screen and a main frame with a menu and toolbar. The .DLLs are called at run-time so you don’t need to compile a large Master .EXE. The Master .APP should have the same settings as the sub-applications except that it is always compiled as an .EXE.

The master .APP should have these settings:

In the **Application Properties** dialog:

Dictionary File:	master dictionary.
Destination Type:	

In the **Global Properties** dialog **General** tab**:**

Generate template globals and ABCs as External:	ON

In the **Global Properties** dialog **File Control** tab**:**

Generate All File declarations:	OFF
External:	ALL EXTERNAL
All Files declared in another .App:	ON
Declaring Module:	Leave this blank

In the Application’s Module Tree:

Choose **Application** ➤ **Insert Module,** then in the **Select Module Type** dialog, select **ExternalDLL,** then in the **Module Properties** dialog, select the corresponding .LIB for the .DLL containing the data definitions.

Choose **Application** ➤ **Insert Module,** then in the **Select Module Type** dialog, select **ExternalDLL,** then in the **Module Properties** dialog, select the corresponding .LIB for the sub-application .DLL. Repeat this step for each sub-application.

For each procedure the main application calls, edit the ToDo procedure as follows:

Template: External template.

Module name: Select the .LIB from the drop-down list.

If necessary delete any empty generated modules.

7. The Team Leader compiles the master .APP and tests the calls to the .DLLs.
8. The Team leader repeats the last step on a periodic basis until all work by all developers is complete, and the entire application can be tested.

One-Piece EXEs

A one-piece .EXE is an .EXE file that contains everything it needs to run except files that cannot be linked in (.VBXs, data files, etc.). That is, the .EXE needs no associated .DLLs, .ICOs, etc., because they are already linked into the .EXE.

When to Use One-Piece EXEs

Development Pros and Cons

Potentially none, because the final state of the .EXE does not necessarily affect the organization of the project at development time.

For example, an application may be developed using several .APP files to generate separate executables (.EXE or .LIB). Eventually the separate executables are linked together to make a one-piece .EXE.

The make time for the one-piece .EXE is greater than the make time for its individual components, so at the point in the development cycle where you link in and test all components, make times increase.

Maintenance Pros

You deploy a single file. There's no chance of forgetting a required executable file or of deploying an incompatible set of .EXEs and .DLLs. There's no chance of a conflict with a different version of a .DLL with the same name.

Maintenance Cons

Your changes require relinking the entire project, plus deploying a larger file. Larger files are generally more difficult to deploy. Your end user cannot share .DLLs between mutiple programs; that is, if two or more one-piece .EXEs execute the same code, that code is duplicated within each .EXE—an inefficient use of disk space.

Conclusion

Use a one-piece .EXE for small and medium applications that are infrequently deployed. That is, the application is reasonably stable or the application is distributed to only a few end users.

Use .LIBs when you want to separate your development process into multiple applications, but you want to deploy your application as a one-piece .EXE. You can initially make each application an .EXE so it can be independently developed and tested. When you are ready to integrate all the applications together, make the called applications into .LIBs, then link them into the parent application.

How to Implement One-Piece EXEs

Create a one-piece .EXE from one or several .APP files. The simplest approach is to create an .EXE from a single .APP file. Alternatively, you may use several .APP files to create one or more .LIBs which are linked into the one-piece .EXE.

> **Tip: You may want to create .EXEs rather than .LIBs during the development cycle in order to reduce link times and to permit isolated testing and debugging of discrete executables. Near project completion, convert to .LIBs by changing the Project's Target Type or the Application's Destination Type to LIB.**

To create a one-piece .EXE from one or more .APP files:

Set the Parent Application's Destination Type to EXE

The highest level (parent) application's **Destination Type** should be set to .EXE in the **Application Properties** dialog. The parent application is the one whose procedures are not called by any other application's procedures.

1. Open the parent application.
2. Choose **Application ➤ Properties** from the menu.
3. In the **Application Properties** dialog, choose *executable(.EXE)* from the **Destination Type** drop-down list.

> **Tip: Setting the Project's Target Type is equivalent to setting the Application's Destination Type and vice versa.**

Set the Parent Application's Run-Time Library to Local

Set the parent application's **Run-Time Library** to *Local* in the Project Editor's **Global Options** dialog. This links the Clarion runtime functions, including all referenced file drivers into the application's .EXE.

1. Open the parent application.
2. Choose **Project ➤ Properties** from the menu.
3. In the **Project Editor** dialog, press the **Properties** button.
4. In the **Global Options** dialog, choose *Local* from the **Run-Time Library** drop-down list.

> **Note: Run-Time Library in this context automatically includes all file drivers used by your application!**

If there are no LIBs, then you are ready to make your one-piece EXE. Skip to the last step in this section *Make and Run Your One-Piece EXE*.

Make the LIBs

You may make LIBs for other purposes than creating one-piece .EXEs. The following information, except *Set the LIB Application's Run-Time Library to Local*, applies for any LIBs you make.

Make and test your LIB application just as you would any other application. Note the name of your LIB and it's procedures because you will reference these names in your parent application.

When you are finished testing, remake your LIB application with the following settings to create a LIB file to link into the parent EXE.

1. Choose **Application ➤ Properties** from the menu.
2. Choose *Library(.LIB)* from the **Destination Type** drop-down list then press **OK**.

Declare the LIB's Procedures Globally

By default, the ABC Templates declare procedures locally using local MAPs. Local MAPs minimize compile times but are inappropriate when making a LIB and will produce a "Link Error: *procedure*@F is unresolved..." when making the calling application. Within the LIB's application, you should declare globally, each procedure called by the parent application. So either

1. In the **Procedure Properties** dialog, check the **Declare Globally** box for each called procedure;

or

1. Choose **Setup ➤ Application Options** from the menu, select the Generation tab, then clear the **Create Local Maps** box to declare all procedures globally.

Set the LIB Application's Run-Time Library to Local

Set the LIB application's **Run-Time Library** to *Local* in the Project Editor's **Global Options** dialog. This tells the linker that the Clarion run-time functions, including all referenced file drivers, are in the parent application's EXE.

1. Choose **Project ➤ Properties** from the menu.
2. In the **Project Editor** dialog, press the **Properties** button.

3. In the **Global Options** dialog, choose *Local* from the **Run-Time Library** drop-down list, then press **OK**.
4. Choose **Project ➤ Make** from the menu to make the .LIB.

Add the LIB Modules to Your Parent Application

The following steps tell your parent application that some external procedures are called from a particular .LIB.

1. Open the parent application.
2. Choose **Application ➤ Insert Module** from the menu.
3. In the **Select Module Type** dialog, choose *ExternalLIB*.
4. In the **Module Properties** dialog **Name** field, type the name of the .LIB then press **OK**.
5. In the **Module Properties** dialog **Map Include File** field, type the name of the include file that contains the called procedures' prototypes.

   You must create this file. You can do so by copying the procedure prototypes from the generated *appname*.clw file and saving them into a separate .inc file.

Add the External Procedures to Your Parent Application

1. Open the parent application.
2. Choose **Procedure ➤ New** from the menu.
3. In the **New Procedure** dialog, type the name of the external procedure and press **OK**.
4. In the **Select Procedure Type** dialog, select *External*.

   This opens the **Procedure Properties** dialog. The **Files**, **Procedures**, **Formulas**, and **Extensions** buttons on this dialog are not meaningful for external procedures and you should ignore them.
5. In the **Module Name** drop-down list, select the .LIB that contains the external procedure then press **OK**.

   This tells the parent application where to find the external procedure.

Add RSC Files to Your Parent Application's Project

For 16-bit applications only, add any .RSC, .ICO, .BMP, etc. files used by the .LIB to the parent application's project.

1. Open the parent application.
2. Choose **Project ➤ Edit** from the menu.
3. Highlight *Library and Object files* then press the **Add File...** button.

4. Navigate to the .RSC file then press **OK**.

   By default, the .RSC files are in the same directory as the .OBJ files for the .LIB; that is, \CLARION5\OBJ.

Tip: **You can tell the Application Generator to generate the .RSC files into your application directory by editing the redirection file (..\CLARION5\BIN\CLARION5.RED). Insert a period and semicolon in the *.rsc line as follows:**

```
*.rsc = .;\CLARION5\obj
```

See *Project System—Library, Object, and Resource Files* for more information.

Make and Run Your One-Piece EXE

1. Open the parent application.
2. Choose **Project ➤ Run** from the menu.

   The Application Generator and the Project System generate source code, then compile and link the one-piece EXE based on:

   - Application Properties' *Executable(.EXE)* Destination Type
   - Project Editor—Global Options *.EXE* Target Type
   - Project Editor—Global Options *Local* Run-Time Library
   - The .LIB Modules and external procedures you inserted into your application tree.

Note: **Do not mix 16-bit and 32-bit applications by trying to link a 16-bit .LIB or .DLL into a 32-bit .EXE, or vice versa.**

EXE Plus DLLs

.EXE Plus .DLLs is an .EXE that requires some associated files that could have been linked into the .EXE at compile time. That is, the .EXE *does* need associated .DLLs, .ICOs, etc., because they are *not* already linked into the .EXE.

The use of this configuration primarily affects the maintenance cycle of the application; however, the creation of one or more .DLLs implies the presence of multiple .APP files, which also affects the development cycle.

The .DLLs often include the Clarion run-time .DLL and one or more file driver .DLLs. Please be aware that these .DLLs may be hidden by creating a one-piece .EXE, or by linking them into another .DLL that you create! See *Hiding the Clarion Run-time Library*.

Tip: The Clarion runtime libraries assume the .EXE or .DLL where a window was most recently opened is where any referenced icons are located.

When to Implement

Use an .EXE plus .DLLs for two or more applications that share a common .DLL. This saves disk space. A good example of this is two Clarion applications that use the same TopSpeed file driver: C5TPSx.DLL. The applications and shared .DLLs should be fairly stable so you don’t have two different .DLLs with the same name on the same machine. Different .DLLs with the same name can lead to confusion and worse if the .DLL files are not managed properly.

Use an .EXE plus .DLLs for very large applications or for applications that are deployed frequently. This lets you deploy only the portions of the application that actually change.

How to Implement

To create an .EXE plus .DLLs:

Set the Parent Application’s Destination Type to EXE

Set the parent application’s **Destination Type** to .EXE in the **Application Properties** dialog.

Note: Typically, an .EXE calls .DLLs, however, a.DLL may call other .DLLs.

1. Open the parent application.
2. Choose **Application ➤ Properties** from the menu.
3. In the **Application Properties** dialog, choose *executable(.EXE)* from the **Destination Type** drop-down list.

> **Tip:** **Setting the Project's Target Type is equivalent to setting the Application's Destination Type and vice versa.**

Set the Parent Application's Run-Time Library to Standalone

The parent application's **Run-Time Library** should be set to *Standalone* in the Project's **Global Options** dialog. This allows the .EXE to link in Clarion external functions at run-time from C5RUN[x].DLL.

1. Open the parent application.
2. Choose **Project ➤ Properties** from the menu.
3. In the **Project Properties** dialog, press the **Properties** button.
4. In the **Global Options** dialog, choose *Standalone* from the **Run-Time Library** drop-down list.

If you are not making your own .DLLs, that is, if there is only one .APP file, then you are ready to make your .EXE plus .DLLs. Skip to the last step in this section, *Make and Run Your .EXE Plus .DLLs.*

Generate Global Data and ABCs EXTERNAL

> **Note:** **If your application's .DLLs use your application files, the FILE definitions and all Global variables must be declared in a .DLL (not the .EXE) and exported. See *Sub-Application Approach* above for more information.**

Generate the parent application's global data and ABC Library declarations with the EXTERNAL attribute. This allows the ABC Library objects, the GlobalRequest and GlobalResponse variables, plus global variables you define to be shared between local and external functions.

1. Open the parent application.
2. Choose **Application ➤ Global Properties** from the menu.
3. In the **Global Properties** dialog, check the **Generate template globals and ABCs as EXTERNAL** box.
4. If there is a green check beside the **Data** button, then press it.
5. In the **Global Data** dialog, press the **Properties** button.
6. In the **Field Properties** dialog, select the **Attributes** tab.
7. In the **Storage Class** drop-down list, select **EXTERNAL-DLL**.

Repeat *5* through *7* for each global data item.

Make the DLLs

Make and test your .DLL application just as you would any other application. Note the name of your .DLL and it's procedures because you will reference these names in your parent application.

Typically one of the .DLLs (the "Data DLL") contains file declarations and ABC Library objects only, with no other procedures. We recommend this configuration because it is easy to set up and to maintain. If needed (your system uses more than one data dictionary), you can make a separate DLL for each data dictionary, and a separate DLL for the ABC Library objects. See *Sub-Application Approach* above for more information.

When you are finished testing, remake each DLL application with the following settings to create a DLL file to link into the parent EXE at runtime.

1. Choose **Application ➤ Properties** from the menu.
2. Choose *Dynamic Link library(.DLL)* from the **Destination Type** drop-down list and press **OK**.

   Choosing a **Destination Type** of .DLL automatically exports (makes public) each procedure in the application.

   Tip: To optimize performance, export only the procedures called by another executable.

3. In the **Procedure Properties** dialog, clear the **Export Procedure** box for each procedure not called externally.

   Procedures that are only called within the .DLL application need not be exported.

Set the DLL Application's Run-Time Library to Standalone

Set the .DLL application's **Run-Time Library** to *Standalone* in the Project Editor's **Global Options** dialog. This lets the .DLL link in Clarion functions at run-time from C5RUN[x].DLL.

1. Choose **Project ➤ Properties** from the menu.
2. In the **Project Editor** dialog, press the **Properties** button.
3. In the **Global Options** dialog, choose *Standalone* from the **Run-Time Library** drop-down list, then press **OK**.
4. Choose **Project ➤ Make** from the menu.

Add the DLL modules to the Parent Application

The following steps tell your parent application that some external procedures are called from a particular .DLL.

1. Open the parent application.
2. Choose **Application ➤ Insert Module** from the menu.
3. In the **Select Module Type** dialog, choose *ExternalDLL*.

> **Note: When you make a .DLL with Clarion, you also generate a corresponding stub .LIB. The linker links the stub .LIB at compile time so your application can call the .DLL at run-time.**

4. In the **Module Properties** dialog, press the **Module Name** ellipsis button to select the stub LIB (or type the LIB name) that corresponds to the .DLL, then press **OK**.

Add the External Procedures to the Parent Application.

1. Open the parent application.
2. Choose **Procedure ➤ New** from the menu.
3. In the **New Procedure** dialog, type the name of the external procedure then press **OK**.
4. In the **Select Procedure Type** dialog, select *External*.

   This opens the **Procedure Properties** dialog.

   The **Files**, **Procedures**, **Formulas**, and **Extensions** buttons on this dialog are not meaningful for external procedures and you should ignore them.
5. In the **Module Name** drop-down list, press the **Module Name** ellipsis button to select the stub LIB (or type the LIB name) that corresponds to the .DLL, then press **OK**.

Make and Run Your EXE Plus DLLs

1. Open the parent application.
2. Choose **Project ➤ Run** from the menu.

   The Application Generator and the Project System generate source code, then compile and link the .EXE plus .DLLs based on:

   - Application Properties' *Executable(.EXE)* Destination Type
   - Project Editor—Global Options .*EXE* Target Type
   - Project Editor—Global Options *Standalone* Run-Time Library
   - The .DLL Modules and external procedures you inserted into your application tree.

Note: Do not mix 16-bit and 32-bit applications by trying to link a 16-bit .LIB or .DLL into a 32-bit .EXE, or vice versa.

Hiding the Clarion Run-time Library

Making a one-piece .EXE hides the Clarion run-time Library within the .EXE. You can also hide the run-time Library by linking it into another .DLL as follows:

Set the Parent Application's Destination Type to .EXE

The parent application's **Destination Type** should be set to .EXE in the **Application Properties** dialog. The parent application is the one whose procedures are not called by any other application's procedures.

1. Open the parent application.
2. Choose **Application ➤ Properties** from the menu.
3. In the **Application Properties** dialog, choose *Executable(.EXE)* from the **Destination Type** drop-down list.

Tip: Setting the Project's Target Type is equivalent to setting the Application's Destination Type and vice versa.

Set the Parent Application's Run-Time Library to External

The parent application's **Run-Time Library** should be set to *External* in the Project's **Global Options** dialog. This tells the .EXE to link in Clarion external functions at run-time from a .DLL other than C5RUN[x].DLL.

1. Open the parent application.
2. Choose **Project ➤ Properties** from the menu.
3. In the **Project Properties** dialog, press the **Properties** button.
4. In the **Global Options** dialog, choose *External* from the **Run-Time Library** drop-down list.

Note: Run-Time Library in this context automatically includes all file drivers used by your application!

Note: If your application's .DLLs use your application files, the FILE definitions and all Global variables must be declared in a .DLL (not the .EXE) and exported. See the *Sub-Application Approach* above for more information.

Generate Global Data and ABCs EXTERNAL

Generate the parent application's global data as EXTERNAL. This allows the GlobalRequest and GlobalResponse variables, plus global variables you define to be shared between local and external functions.

1. Open the parent application.
2. Choose **Application ➤ Global Properties** from the menu.
3. In the **Global Properties** dialog, check the **Generate template globals and ABCs as EXTERNAL** box.
4. If there is a green check beside the **Data** button, then press it.
5. In the **Global Data** dialog, press the **Properties** button.
6. In the **Field Properties** dialog, select the **Attributes** tab.
7. In the **Storage Class** drop-down list, select **EXTERNAL-DLL**.

   Repeat *5* through *7* for each global data item.

Make the DLL Containing Clarion's Run-time Functions

1. Choose **File ➤ New** from the menu.
2. In the **New** dialog, select the **Application** tab.
3. In the **File Name** field, type the name of the application.
4. Press the **Create** button.

   The **Application Properties** dialog appears.
5. If applicable, select a data dictionary.
6. Choose *Dynamic Link Library(.DLL)* from the **Destination Type** drop-down list and press **OK**.

Set the DLL Application's Run-Time Library to Local

1. Choose **Project ➤ Properties** from the menu.
2. In the **Project Editor** dialog, press the **Properties** button.
3. In the **Global Options** dialog, choose *Local* from the **Run-Time Library** drop-down list, then press **OK**.
4. Choose **Project ➤ Make** from the menu.

   This links the Clarion functions into your .DLL at compile time so that they can eventually be linked into your .EXE at run-time. It also generates a Module Definition file (.EXP) for your .DLL. You edit the .EXP file in the following step to export the run-time functions from your .DLL.

Export the Clarion Run-time Functions

1. Open the parent application.
2. Choose **Project ➤ Make** from the menu.

   Because the run-time functions have not been exported, the linker generates an "unresolved" error for each function. Expect several hundred to one thousand plus error messages, depending on your application.

   In the following steps we use the Clarion Text Editor to convert the error messages into EXPORT statements. You may use any text editor for this purpose.
3. Press the **Edit Errors** button, then press **OK** to edit the error messages.
4. Delete all messages except the "...function is unresolved..." messages.
5. Choose **Search ➤ Replace** from the menu.
6. In the **Find what** field, type *is unresolved in file <filename.ext>*.

   Replace <filename.ext> with the filename and extension in the first message.
7. In the **Replace with** field, type an ampersand and a question mark with no spaces, like this: @?.
8. Press the **Replace All** button.

   This converts the error messages for file <filename.ext> into EXPORT statements. Repeat steps 5 through 8 until all the ...function is unresolved... messages are converted.
9. Delete any duplicate statements.

   Select all the statements and copy them to the clipboard, then paste them into a word processor or spreadsheet that has sort capabilities. Sort the statements so that duplicate statements appear together, then delete the duplicates. Select all the remaining statements, copy them to the clipboard and save your work.

   Insert the EXPORT statements into the export file (.EXP) for the .DLL application in the following steps.
10. If you have not already done so, choose **Exit!** from the menu to exit the Text Editor.
11. Choose **File ➤ Open** from the menu.
12. In the **File Name** field, type **.exp*, then press ENTER.
13. Select the .EXP file associated with your .DLL (not your parent application), then press **OK**.

    This opens the .EXP file with the Text Editor.

14. Scroll to the end of the file then CLICK the mouse to set the insertion point.
15. Choose **Edit ➤ Paste** from the menu.

> **Tip: You can modify the .EXP file by embedding text in .EXP related Global Embed points.**

Make the DLL Application with the Exported Functions

1. Open the .DLL application.
2. Choose **Project ➤ Make** from the menu.

   This links the Clarion run-time functions into your .DLL and exports them so they can be called at run-time by your .EXE.

Make the Parent Application.

1. Open the parent application.
2. Choose **Project ➤ Make** from the menu.

   Now that the Clarion run-time functions are exported, the linker resolves calls to those functions and issues no more unresolved messages.

Making LIBs and DLLs for Other Environments

Unlike most non-Clarion development tools (Delphi, PowerBuilder, Visual C++, Visual Basic, etc.), Clarion generates executables that use register based parameters. Since Clarion uses register based parameters and these other tools do not, when making DLLs for other tools, you must prototype your exported Clarion procedures with the C or PASCAL attribute, depending on the calling environment. See C and PASCAL calling conventions in the *Language Reference* for more information.

Non-Clarion environments generally cannot link Clarion created LIBs because of memory model incompatibility. However, they can link the stub LIBs associated with Clarion DLLs.

APPENDIX D - DDE—DYNAMIC DATA EXCHANGE

Overview

This chapter introduces Dynamic Data Exchange (DDE). For a complete discussion of Clarion and DDE, see the *Language Reference*. This appendix is meant only to provide a description of what DDE can do, to demonstrate that it's easy to implement, and to suggest a tip or two to help you explore DDE.

DDE is a Windows Inter-Process Communication (IPC) protocol. A DDE "conversation" consists of two applications trading messages. Within the DDE conversation, one application acts as the *client*, the other as the *server*.

The application which starts the conversation, requesting data or services from the other, is the client. The contacted application is the server. The server must "register" with Windows that it has server capability.

Clarion lets you create both DDE clients and DDE servers. An application can be both. In fact, your application can act as both a client and server at the same time, though it requires separate DDE conversations.

See the *Programmer's Guide* for information on using Clarion as a DDE server.

Capabilities

As a DDE client, your application can:

- Initiate a DDE conversation with a DDE server with the DDECLIENT function.
- Receive data from a server with the DDEREAD statement. EVENT:DDEdata tells your application when there is data for it to read.
- Send a command string to a server with the DDEEXECUTE statement.

  Many existing Windows applications allow access to their functionality through command messages. For example, you can execute any Microsoft Excel macro statement by enclosing it in square brackets and sending it as a string parameter in the DDEEXECUTE statement.
- Send unsolicited data to a server with the DDEPOKE statement.

  Typically, you provide the server with an "item" description, and its value (string). For example, to place a value in a specific cell in an Excel spreadsheet, the item is the cell address, in R1C1 format. The value is the actual value you want to put in the cell.

As a DDE server, your application can:

- Check the "topic" which the client contacting your server application specifies.

  When a client contacts your server application, it specifies, in a string, what the conversation should be about. You code your application to check the string against a list that you specify, then take an appropriate action when the topic matches an item in your list.

  The *de facto* Windows DDE "standard topics" are the current document name, and the "System" topic. The current document is the name of any open file associated with the server application. The "System" topic usually triggers a return message, listing the available "topics" which your server supports, each separated by a comma.
- Provide automatic data updates with the DDEWRITE statement, when the "mode" is set to DDE:auto.

  This lets you specify a variable. The server will automatically send a message to the client when the value of the variable changes.
- Allow access to "commands."

  The server application retrieves the command string with the DDEITEM function. You code your application to check the string against a list that you specify, then take an appropriate action when the command matches an item in your list.

See the *Language Reference* for explanations of all DDE statements and functions. The remainder of this chapter describes these capabilities with a generalized example, in which a Clarion DDE client sends a sample Client request to Microsoft Excel, then sends unsolicited data to place in a single spreadsheet cell.

DDE Conversation—Client to Server

Starting a DDE conversation is as easy as using the DDECLIENT function. The only requirement is that both applications must already be running to open the channel.

The simplest way to ensure that the conversation takes place at run time is to use an IF structure. The DDECLIENT function returns zero if the server application isn't already running. Test its return value, and use the RUN statement to start the server application if it returns zero.

Many of the DDE procedures and functions require that you specify the DDE channel number, which is an integer that Windows returns when you open the DDE conversation. Create a local variable to hold the return value. Begin at the **Procedure Properties** dialog of the procedure you wish to contain the code for the DDE conversation.

Create a variable to hold the DDE channel number

1. Press the **Data** button in the **Procedure Properties** dialog.
2. Press the **Insert** button in the **Local Data** dialog.
3. Type *Channel* in the **Name** field.
4. Choose **LONG** from the **Type** drop-down list.
5. Press the **OK** button to close the **Field Properties** dialog.
6. Press the **Close** button to close the **Local Data** dialog.

Initializing the Conversation

You must embed the code to initialize the DDE conversation, starting the server application if it's not already started. Assuming a menu choice in your application begins the conversation, embed the code at a field event associated with the Accepted event for the menu choice.

1. Choose the appropriate field event in the **Embedded Source** list.
2. Press the **Insert** button.
3. Highlight the **Source** item in the **Embedded Source** dialog, then press the **Select** button.

   This opens the Text Editor to the selected embed point.
4. Type the following code, substituting the file name (without extension) of the Server application for "Excel."

```
Channel = DDECLIENT(‘Excel’,’System’)   ! Excel re System topic
IF Channel < 1                          ! If no contact made
  RUN(‘Excel’)                         ! Attempt to start Excel
  Channel = DDECLIENT(‘Excel’,’System’) ! And try again
ELSE
  RETURN
END
```

The code example is deliberately simplistic; it would be more efficient to LOOP through the attempt to contact twice, then warn the end user of the failure.

The code attempts to open a DDE conversation with Excel named as the server. The DDECLIENT function returns a value corresponding to the channel; it doesn’t matter what the channel number is. If it’s less than one, it failed. You must therefore start the server and try to open the conversation again.

The second parameter of the DDECLIENT function is the DDE “Topic.” It tells the server what the DDE conversation is “about.” In most cases, the topic is a file name. In this case, the code names the “System” topic, which tells Excel the conversation is not regarding a particular document file.

Sending DDE Commands

Once the DDE channel is open, you can then use the DDE functions to send commands, data, or requests to the server.

The example code below sends a command to Excel to open a new file and save it under a specified file name. This is a common DDE task when working with commercial applications. Often, the server application allows access to “document” functions only when you specify a document name in the DDECLIENT function. The document name must be a file that already exists.

In this particular case, to execute any “document” actions, such as entering a value in a cell, Excel (and many other applications) require the DDE channel “topic” to be the name of document. Therefore, if your application is providing new data it wants the server to save in a *new* document file, your application:

❏ Opens a conversation about the “System” topic.

❏ Sends a command asking the server to save a document file under a specified name.

❏ Closes the conversation.

❏ Opens a second conversation with the server, this time specifying the newly created file’s name as the topic.

❏ Sends the “unsolicited” (because the server didn’t ask for it) data and then tells the DDE Server (Excel) to execute commands or other requests for data that apply to the file.

❏ Closes the conversation.

The following therefore should execute only if the example code previously shown was successful.

Open a new (spreadsheet) file

```
DDEEXECUTE(Channel,’[NEW(1)]’)      ! Excel’s File/New command
```

The DDEEXECUTE statement takes the DDE channel number as its first parameter, and the command string as the second. Excel requires you to enclose all DDE commands in square brackets (a standard DDE convention). This command creates a blank spreadsheet.

The Excel command string enclosed by the square brackets is an Excel macro statement. Excel, and many other applications allow you to send a macro statement with the DDEEXECUTE statement. In this particular case, you don’t have to know the name of the open Excel file to execute the statement.

Tip: Many commercial applications with their own macro languages allow you to both record and edit macros. Use the application to make a “dry run” of the actions you need it to execute, with its macro recorder turned on. Edit the resulting macro, and use the clipboard to copy each macro statement to your embedded source window. Put each macro statement in the second parameter of the DDEEXECUTE statement, and you can be assured of the correct syntax for the DDE command!

Save the new (spreadsheet) file

```
DDEEXECUTE(Channel,’[SAVE.AS(“DDE_TEST.XLS”,1,””,FALSE,””,FALSE)]’)
```

Knowing the name lets you close this channel, then open another specifying the file name as the topic. Note that the Excel command string requires double-quote marks.

Close the channel

```
DDECLOSE(Channel)                        ! Close first DDE channel
```

Sending Data from Client to Server

To continue the example, to send data to Excel, you need to open another DDE conversation, this time with the newly created file name as the topic:

Open the DDE channel

```
Channel = DDECLIENT(‘Excel’,’DDE_TEST.XLS’)
                             ! New channel under known file name
```

Send the Data

To place data in a spreadsheet cell, use the DDEPOKE statement.

```
DDEPOKE(Channel,’R1C1’,’999’)
```

Following the successful placement of the value in the spreadsheet, you could then send more Excel macro statements using DDEEXECUTE. This would allow you to. for example, send additional spreadsheet data, highlight a range, then tell Excel to draw a chart.

You’ll find all the DDE commands and functions in their own section in the *Language Reference*.

APPENDIX E - GLOSSARY

All definitions should be considered general terms, except where otherwise indicated. The context for definitions marked (Clarion) pertain to the Clarion language or the Clarion development environment. Likewise for (SQL), which applies to generalized Structured Query Language usage.

ACCEPT loop

(Clarion) An event handling loop beginning with the ACCEPT statement. The loop transparently processes the Windows messages and related events which affect the application's window. A single ACCEPT loop automatically gets end user input for all controls within a given window.

accepted event

(Clarion) An event generated when an end user interacts with a window control, such as when moving the focus to a field, that results in the event being reported in the ACCEPT look.

access key

(Clarion) A specified key or index to set the order for processing records in a procedure.

active window

The document or active window which currently has the focus; Windows sends the next keyboard or mouse action to the ACCEPT loop of the active window.

alias

An alternate name for a data file, which allows multiple, independent operations on it. Clarion provides a separate record buffer for each alias, increasing the performance of the separate operations.

ANSI character set

Character set standardized by the American National Standards Institute. Many ANSI characters are different then the corresponding ASCII character set. The ANSI set contains more non-English characters. The standard Microsoft Windows character set is the ANSI character set.

API

Application Programming Interface; generally refers to the Windows API. Allows applications to dynamically link function calls to the three main Windows libraries (USER.EXE, GDI.EXE, and KERNEL.EXE), plus the external libraries such as MMSYSTEM.DLL. Just about everything that every Windows program does is accomplished with the API.

append

Add a record to a data file, usually without updating a key or index.

applet

A small, single purpose application; applets are not necessarily stand alone executable programs. The “programs” managed by the Windows Control Panel, for example, are called applets, though they are actually dynamic link libraries with specialized entry points. The accessories which ship with Windows are also known as applets.

application

A computer program designed for a specific type of work; the terms “application” and “program” are interchangeable.

application generator

A program which combines prewritten, generalized executable code modules or fragments to create an application.

application generator

(Clarion) The part of the development environment which manages pre-written template procedures, obtains customizations from the developer, and generates Clarion language source code files.

application tree

(Clarion) An Application Generator dialog which graphically depicts the hierarchy of procedures for an application.

application window

In a Multiple Document Interface application, the parent window, usually containing no controls, in which all child document windows appear.

array

A ordered series or group of dimensioned values or data items.

ASCII character set

Character set standardized as the American Standard Code for Information Interchange. The standard IBM PC character set.

assignment statement

A statement placing a value in a variable; for example, A = 6 places the value 6 in variable “A.”

attribute

(Clarion) A modifier to a data declaration which specifies an optional property.

auto-increment field

(Clarion) A key field which stores a value which increases with each successive record, and is generally not available to the end user. The application places the value in the field immediately upon appending the record.

background priority

A measure, expressed in a ratio, for the amount of CPU processing time allocated to a program or task which does not currently have system focus. In the Windows 16-bit environment, all multitasking is cooperative; therefore, all background processing is dependent on all executing applications properly yielding at regular intervals.

band view

(Clarion) A specialized layout mode within the Report Formatter. Displays the contents of each part of the report structure in separate panes.

binary memo

(Clarion) A memo field suitable for holding non-ASCII contents, such as images.

bind

(Clarion) A statement which allows a variable name to be used in a dynamic expression which is assembled and processed at run-time.

bitmap

A binary file representation of a graphic or picture; raster format defines the image by absolute pixels. Popular bitmap formats supported by Clarion include .BMP, .GIF, .ICO, .PCX, .JPG. Sometimes refers specifically to the .BMP file format, an uncompressed, but widely supported file format.

Boolean

A logical expression which evaluates to true or false, one or zero.

Border or Line Color

The color designated for the outside line of a graphical control.

break field

(Clarion) A field or variable monitored when processing a report structure. When the value in the field changes while sequentially processing records, the print engine processes the next element in the report structure (usually the group footer).

breakpoint

A debugger stopping point, relative to a source or disassembly code statement. The application executes up to the breakpoint, then halts and turns execution over to the debugger, which can then examine variables and expressions to search for bugs.

BringWindowToTop

Windows API function for forcing a window to always display on top of all other windows on the desktop. Implemented in Clarion by the TOOLBAR attribute.

Browse

A specialized list box procedure dedicated to displaying database records arranged in columns and rows.

built-in

(Clarion) Default map definitions, as provided in source code format in the BUILTINS.CLW file.

button

A control that initiates a command, or selects an option. An end user chooses a button by clicking with the mouse.

calculated field

A field created via an expression which may include one or more database fields.

cascading menu

A hierarchical submenu, sometimes called a child menu. Parent menus that lead to cascading menus usually have a right-pointing triangle at the right side of the menu item, to cue the user to the submenu.

case sensitive

A characteristic indicating whether a command treats text typed with capital (uppercase) letters differently than those typed with lower case, or a combination of both.

case structure

A control structure which branches execution to a statement (or group of statements) based upon a single condition or expression.

character string

An alphanumeric data type.

check box

A control consisting of a small square or diamond, in which an end user indicates a on/off, yes/no, or true/false choice.

child window

An MDI document window displaying a document or view within the main application window.

Clarion standard date

(Clarion) The number of days elapsed since December 28, 1800; the valid range is from Jan. 1, 1801 through Dec. 31, 2099.

class

The Clarion CLASS structure declares an object class containing properties and declaring the methods that operate on those properties. In Clarion, the properties of the class are the data members declared in the CLASS and methods are the PROCEDUREs and FUNCTIONs prototyped in the CLASS structure.

click

To place the mouse pointer on a control or window, then press and release the left mouse button.

client

A system attached to a network that accesses shared network resources.

client application

A program that makes requests of a server application using a defined interface such as DDE, RPC, or NetBIOS.

client server architecture

A network configuration by which linked workstations request services from a dedicated program running on a server.

client server networking

A network architecture in which shared resources are concentrated on powerful server machines and the attached desktop systems fulfill the role of clients, making requests across the network for centralized information.

clipboard

A temporary storage area in memory for holding data, maintained by Windows.

Close

To normally terminate processing of a window or file.

code section

(Clarion) The portion of source code containing executable code statements.

color dialog

Standard Windows dialog for choosing color.

column

(SQL) Generally refers to a list of database field contents arranged by records.

combo box

A window control consisting of a synchronized edit box and list box.

command
An executable code statement or program instruction.

comment
Text inserted in a source code file to annotate or explain the code. Clarion language comments begin with the exclamation point (!) character. Each comment terminates at the end of the line it appears on.

commit
Terminates a successful transaction and commits it to disk.

common file dialog
A standard Windows dialog for displaying drives, directories, and file names. The Clarion FILEDIALOG function displays the dialog and returns a file name to the calling application.

compiler directive
An instruction directing a compiler to build an application to meet a certain condition.

concatenate
Append two string data elements to form a longer string comprised of both.

concurrency checking
The process of guarding against two users updating the same record at the same time. Usually consists of checking the record on disk still contains the same values as when it was first retrieved for updating.

conditional statement
An IF statement which branches subsequent execution based on a logical condition.

constant
A static value.

context menu
See popup menu.

control
A fundamental object in Windows that defines the appearance and behavior of a particular visual element such as a menu, an entry field, or a scroll bar.

control alignment
The “Snap-To” behavior, as found in the Window and Report Formatters, by which you may “line up” window and report elements.

control menu
Contains commands for resizing, repositioning, or closing a window.

control properties
(Clarion) Attributes which determine the appearance and functionality of a window or report control.

cool switch
The Windows procedure for switching between active applications by holding down the ALT key and pressing the TAB key.

cooperative multitasking

An operating system scheduling technique that relies on running applications to yield control of the processor to the operating system at regular intervals.

criteria

(SQL) An expression containing a condition which limits the records for processing.

current directory

The default DOS subdirectory, in which Windows or DOS searches for files not identified with a fully qualified file name.

current record

(Clarion) The current database record in the record buffer.

cursor

The mouse pointer. Changing the cursor "shape" can indicate the type of action or selection the end user can effect on a given control or window.

data dictionary

(Clarion) ASCII file describing the individual data files which comprise the database, their structure, keys, relations, and other information describing how an application will process the contents of the database.

data file

Generally, a collection of data elements in an organized format, usually arranged by records (rows) and fields (columns).

data section

(Clarion) The section of source code containing variable and data structure declarations, such as FILE, WINDOW, REPORT, and QUEUE.

data type

A physical description of the type of storage supported by a variable; what sort of values it can hold.

data validation

An expression or the process of checking data against a condition prior to accepting the data for entry into the database.

database

A structured collection of data, contained in one or more data files, plus the key files and other information which describes the order and relations of the data elements.

database administrator

(DBA) A person responsible for designing and maintaining a multi-user database system.

database definition file

(*.DDF). A Btrieve file, separate from the data file, containing the database structure. Equivalent to the header contained internally in most other PC database file formats.

database design

The process of planning and describing the most efficient application or system for storing and managing data for a specific project.

database driver A collection of functions and procedures contained in a dynamic link library, supporting low level access to a specific database file format.

database integrity Under the relational model, database integrity consists of two general rules:

1. Each database file or table must have a primary key serving as a unique identifier for all records.

2. When a table has a foreign key matching the primary key of another table, each value in the foreign key must either equal a value in the primary key of the other table, or be null.

dBase format PC database file format popularized by dBase III.

DBMS Database Management System: generic term for a program that enables a system to perform all the functions associated with managing a database.

DDE Dynamic Data Exchange: a message protocol for exchanging data between Windows applications.

debug To test, diagnose and (hopefully) solve software bugs. The Clarion debugger offers two general modes:

1. Hard mode debugging, in which all keyboard and mouse input goes to the debugger first, before being sent to the application. This effectively suspends all other applications which may have been running prior to starting the debugger in hard mode.

2. Soft mode debugging, in which the debuggee runs as a normal windows application.

debugee The program being analyzed or debugged.

deep assignment (Clarion) Automatically assigns multiple components from one data structure to another, between elements with the same labels (but different prefixes).

default An assumed state or action, which the end user accepts or executes with little or no action.

default button A command button which is activated by default when the user presses the enter button.

default window position The default location at which a new window appears unless a position is specified. The top left corner of the new window is usually below and to the right of the top left corner of the last window, when it first appeared.

delimiter

A character marking the boundaries of one database field from another.

dependent entity

(SQL) A set of data elements dependent on other related entities in the database to identify them..

desktop

The screen area in which all windows, dialog boxes, and icons appear.

DETAIL structure

(Clarion) The portion of a report structure which usually conveys the main data within the printed report. The application loops through, updates, and prints the detail band controls with the contents of all the records being processed.

dialog

By convention, a window of fixed size, that is usually designed to interact with the user.

dialog unit

Special fractional measurement units, based on the system font. Windows automatically calculates the horizontal measurement unit in fourths of the average system character width, and the vertical in eighths of character height. The net effect supports a proportional placement of dialog box elements regardless of the resolution Windows is running in.

disabled

A window, menu, or control visible but prevented from gaining focus.

document-centric design

A design technique that focuses the user on documents and the information therein rather than on the applications generating the data that combine to form the document.

document

Any file which stores data associated with an application.

DOS buffer

A (normally) small amount of memory maintained by the operating system for short-term storage of data transferred to/from a disk drive. The size is set by the BUFFERS setting in the CONFIG.SYS file, where one unit equals 512 bytes.

double-click

To press and release the left mouse button twice, quickly. Executes the default action on a selection.

drag

To press the left mouse button, then move the mouse while continuing to hold the button down. Usually a visual cue indicates a process such as moving a selected object, or rubber-banding a region. Releasing the button completes the action.

drag and drop

To select an object in a window or dialog box, press down the left mouse button, move the mouse while continuing to hold the button down, then release the button when the pointer is on top of another object. When drag and drop is supported by the program(s), the action generally indicates the dropped object is to be processed in some way by the recipient object.

driver string (Clarion) The second parameter of the DRIVER attribute. Consists of valid codes switches that operate on file open for the particular driver.

drop-down list A list box control which only displays only the current selection when closed. When the user opens the list box, it expands to include additional choices.

dynamic link library (DLL) A library of shared functions that applications link to at run-time, as opposed to compile time.

embedded source (Clarion) Executable code statements, written by the developer, and inserted into generated source at predefined points within a procedure generated by the Application Generator.

Embeditor (Clarion) Clarion’s Text Editor opened in a special mode that lets you embed source code within the context of the surrounding generated code.

enabled Normal window, menu, or control state allowing focus and/or user input.

encapsulation Bundling the properties of a class (its data members) together with its methods (the procedures that operate on the data members) into one coherent unit.

encryption The storage on disk of data in scrambled or encrypted form, such that an unauthorized user may not access the data in an intelligible format.

equi-join (SQL) A join which takes two database files (or tables) and creates a new, wider table consisting of all possible concatenated records (or rows), where there are matching values in the join fields.

event An action that is of interest to one or more software components. Triggers a Windows message to the application’s message queue. Clarion handles most of the actual messages internally.

event driven programming A programming technique in which the application responds to events as opposed to data.

Excel format File format used by the Microsoft Excel spreadsheet application. Note: an ODBC driver exists for this format, and is available in the Microsoft ODBC 2.0 Software Development Kit.

exclusive access Opening a DOS file so that no other user in a multi-user environment may update the same file.

executable A standard .EXE application file capable of being launched by the Microsoft Windows shell.

expand To decompress, usually for installation purposes, a compressed file.

expression

A mathematical formula containing any valid combination of variables, functions, operators, and constants.

extension

A file name suffix; up to three characters in the DOS file system. Windows 3.1 matches document files to their application via the [Extensions] section in the WIN.INI file.

external name

(Clarion) An attribute which holds the native format name (such as a DOS file name) for a given data element. The Clarion source code refers to the file by the Clarion label.

external procedure

(Clarion) A procedure contained in an external library, such as a library file linked at the time the application is built, or a .DLL, linked at run time.

field

A basic data element or category which names all the values in a column of data within a database file or table.

field equate label

(Clarion) A symbolic constant which references an integer, which references a window control.

field event

(Clarion) An event generated and processed within an ACCEPT loop, specific to a control in a window structure.

file handle

An operating system pointer to a file. The “FILES=“ line in the CONFIG.SYS file sets the system limit on the total number of allowable open files at one time.

fill color

The color designated for the inside of a graphical control.

filter

An expression which isolates a subset of records for an operation.

focus

A visual cue indicating the window control which will receive the next action resulting from user input.

folder

A logical container implemented by the shell, within which the user may group a collection of items. Analogous to a file directory.

font

The family name of related type face files. For example, “Times New Roman” is the font name, and “Times New Roman plain,” “Times New Roman Italic,” “Times New Roman Bold,” and “Times New Roman Bold Italic” are the styles, which are stored in separate files.

font dialog

A standard Windows dialog for picking a typeface, style, size, and optionally, the text color.

font style

Character formatting applied to a font face, such as bold, italic, or bold italic.

foreground priority

A measure, expressed in a ratio, for the amount of CPU processing time allocated to a program or task which currently has system focus.

foreign key

(SQL) A key in one table (database file) whose values match the primary key of another table.

form

A window that displays a single record for editing. By convention there is a separate entry box for each field displayed, and fields are stacked in a vertical arrangement.

form letter

A mailmerge document containing “boiler-plate” text, in which controls reference fields from which to obtain information when creating letters to individuals.

form report style

A report format generally containing one record per page, with field labels and values arranged in a vertical format.

format string

(Clarion) A string specifying the display format for a list box or drop down list box control.

formatter

(Clarion) A specialized window which allows you to visually define the formatting for a data structure in “WYSIWYG” fashion.

function

(Clarion) A specialized procedure which returns a value. The function declaration may optionally define parameters which are passed when calling the function. A function may be used within computed or conditional fields.

GDI

Abbreviation for Graphics Device Interface, the Microsoft Windows dynamic link library responsible for outputting text and images to the screen and printer.

GIF image

Graphics Interchange File format; an image format popularized by CompuServe. Generally acknowledged to offer the best compression ration for 256 color or less images. Attention: should you utilize the word “GIF” anywhere within an application or program, you must add a trademark notice: “GIF (Graphics Interchange Format) is a trademark of CompuServe Information Services.”

global variable

(Clarion) A variable accessible from all levels of a program. Global variables are allocated memory that is not released until the entire program finishes execution.

graph

A graphical representation of related data elements, on screen or paper.

Graphical User Interface

(GUI) An operating system or program environment relying heavily on images to present information to the user and to gather the user’s input.

grayed

A visual cue to the user that the window, menu, or control is unavailable or disabled.

grid snap

A series of coordinates, represented by dots, such as those used by the Clarion Window and Report Formatters, to force controls to exact positioning.

group

(Clarion) A compound data structure which allows you to reference its component variables with a single label.

groupbox

A rectangular line frame with a label at upper left, used to define related controls.

handle

In Windows, an integer serving as a pointer to the memory location for a given object, most commonly a handle to a window (HWND). The handle has approximately the same importance to most API functions as the zip code on a first class letter. In Clarion, its functionality is implemented via field equate labels. You *can* obtain the actual handle to a window or control by examining PROP:handle. The property is read only.

help context string

A unique identifier for a topic or page in a help file, which can be passed to the help engine.

help system

Comprised of the Windows help application (WINHELP.EXE) and a help document (*.HLP) distributed by individual applications. When displaying help, both the application which called it, and WINHELP.EXE are running.

help topic

A page in a Windows help document.

help compiler

A utility available from Microsoft for converting a Rich-Text-Format (.RTF) document into a Windows help (.HLP) document.

hide

Prevent a control or window from displaying on screen; the control exists but is not seen by the end user.

I-beam

A special cursor usually indicating the end user can type text into an edit control.

I/O

Input/Output. The process of moving information into and out of the system.

icon

A graphical representation of a physical object in the system, such as a printer. Also, any small image representing an action, concept or program, as when an icon appears on a command button. The normal icon file format carries the .ICO extension; one of its main features is built-in support for transparency. This enables you to display a small picture without obliterating the background.

IDE

Integrated Development Environment; a complete compiler product which includes tools for producing source code, creating resources, compiling, linking, and debugging an application.

identifier

A label uniquely identifying a variable or other program element.

implicit variable

(Clarion) A specialized variable not declared within the data structure of an application, nor defined before its first use. The compiler creates them when it first encounters them (usually within executable code) and automatically initializes them to zero.

import library

A compile time library (.LIB) used to satisfy references to external functions that will ultimately be resolved at run-time by a DLL.

include file

An external source file read and preprocessed at compile time. In Clarion, the Equates and ABC Library files in the LIBSRC subdirectory are the default include files.

independent entity

(SQL) A set of data elements sharing a set of properties independent of other related entities in the database. Independent entities have unique identifiers, and therefore, primary keys.

index file

An external key file ordered according to the contents of a specified field or expression. An index file usually must be manually updated when adding, deleting, or changing records.

INI file

A Windows Initialization file in ASCII format. The .INI file is divided into sections separated by an identifier enclosed in square brackets. Variables and their values follow, each pair separated by a carriage return, with an equal sign between the variable name and its value. Values may be stored as strings or integers.

inheritance

The mechanism that allows us to build hierarchies of classes. A derived class inherits all the properties and methods of the class from which it is derived (its base class).

insertion point

The point in a document at which the next characters typed by the end user will appear.

interface

The communication between the computer and the user; it presents information to the user and accepts the user’s input.

ISAM

Indexed Sequential Access Method; a database organization in which data files are ordered by keys, and may be retrieved in the sequence of the keys.

join

A join takes two database files (or tables) and creates a new, wider table consisting of all possible concatenated records (or rows).

JPG image

A true-color graphics file format featuring 24-bit color storage. It usually provides for adjustable lossy compression, which allows for greater compression but loss of some resolution.

Kernel

The Windows memory management, process management, and file management functions.

key

An indexed file ordered according to the contents of a specified field or fields. Keys are usually dynamically updated whenever the value in a key field changes.

key-in template picture

(Clarion) A formatting option, which when combined with the MASK attribute, restricts and verifies end user keyboard input according to a specified character pattern applied upon a variable.

keyboard accelerator

A combination of keystrokes that immediately executes a command.

keyword

A reserved word or Clarion language statement.

label

(Clarion) A unique identifier for a variable, procedure, function, routine, or data structure.

library file

A precompiled file (.LIB) containing procedures or functions which may be statically linked to the executable and utilized by a program.

license file

A proprietary key file distributed by a VBX vendor only to a licensed user of the VBX library. The license file allows an IDE to incorporate the VBX control within a window or dialog box. This file is *not* distributable to the end user.

list box

A window control presenting data arranged in rows, and optionally, columns.

literal

A constant referred to in source code by its value. For example, the literal “MyString” refers to a seven byte data item containing ASCII codes for the letters in “MyString.”

local data

Data created by, residing in memory specific to, and accessible only to a specific procedure or function.

lock

A concurrency control mechanism to prevent more than one user from updating the same record at the same time. Within Clarion, the HOLD statement arms record locking.

locked field or record

A field or record currently being updated by one user within a multi-user database, such that an attempt by another user to update the same record at the same time will fail.

logical operator

A true/false or bitwise comparison of two values; logical operators are: =, >, <, <>, >=, <+, NOT, AND, OR, and XOR.

lookup table

A database file on one side of a one to many relation, upon which a variable is searched for, and a corresponding field in the related table is returned.

LOOP structure

(Clarion) A control structure which repeats the execution of the statements it encloses for a specified count.

many-to-many relationship

A connection between two data entities in which there may exist many corresponding values in the foreign key in one database file or table, to many corresponding values in the foreign key of another table. Usually implemented via a "join" file breaking them into two 1:Many relations.

many-to-one relationship

A connection between two data entities in which there may exist many corresponding values in the foreign key in one database file or table, to only one value in the primary key of another "look-up" table. The relationship implicitly describes the direction of the relation. For example, the relation of cities to states implies many cities may belong to the same state. Also called a child-parent relation.

MASK

(Clarion) Specifies pattern editing of user input, converting data to a predefined format. The pattern is specified for an individual control, and enabled when the MASK attribute is added to the window in which the control appears.

maximize box

A window control which resizes a window to full size of the desktop, or if a child window, to the full size of the client area of the application window.

Media Control Interface

The multimedia API support component of Microsoft Windows. Managed by the MMSYSTEM.DLL library and related driver files; abbreviated as MCI.

memo

A free-form, variable length text field, suitable for storing very long strings. In most PC file formats, the memo is stored in a file separate from the fixed-length database fields. A binary memo field is a specialized type of memo field suitable for storing binary information such as graphics.

menu

An element of the user interface listing available actions which the end user may effect upon a document or selected portion of a document.

message box

A standard windows element, usually consisting of a short message string, an OK button, often a standard icon such as "stop" or "information." It may optionally contain additional buttons such as "Cancel," and "Retry."

message queue

The "place" in which Windows holds all messages for an application, which the application checks on a regular basis. The messages consist of everything the application needs to know regarding the user interface—keyboard, mouse and menu events; the system—shutdown messages, and all the other operations which may affect the application. Clarion processes the entire messaging process transparently in the ACCEPT loop.

metafile

In Windows, the representation of a graphic or line art in vector (device independent) format; defines the image as a series of lines and curves, allowing for smooth resizing. Clarion supports the .WMF (Windows Metafile) vector format. The metafile is actually a stored collection of the commands which instruct the GDI (Windows Graphics Device Interface) to display the graphic on the output device.

minimize box A window control which resizes a window to iconic size, usually at the bottom of the desktop, or if a child window, to iconic size, usually at the bottom of the application window.

mnemonic access key The underlined letter in the command names on Microsoft Windows menus. When a user activates a pull down menu, the key executes the command.

modal window A dialog or window which prevents the end user from activating controls from any other of the application's windows (or of any other application, if system modal), until processing of the modal window is completed and the window closed.

modeless dialog A dialog which remains open even while the user "works" in another of the application's document windows. The modeless dialog remains available, so that the user can utilize its functionality; as in a Search dialog, as practiced by most applications.

module (Clarion) A source or library file for a given project.

multi-tasking The capability of an operating system to execute multiple programs at the same time. Preemptive multi-tasking allots percentages of CPU time to each individual task, with the operating system automatically switching to the next task at the end of its time allotment. Cooperative multi-tasking, supported by Windows 3.1, relies upon the currently executing program to finish a task, or part of one, then yield to the next program. See also *Thread*.

multiple selection An extended list box selection, signifying the user has marked more than one item for a subsequent action.

multilevel index To speed up access to a large table or data file, a multilevel index functions as an index to an index. For example, index level one could contain pointers to four subindexes which respectively index entries beginning with A-E, F-L, M-R, and S-Z. This example describes a classic B-TREE index structure.

Multiple Document Interface (MDI) A Windows programming convention which allows an application to manage several documents, or views of documents, each in its own child window, all in an application frame window.

multi-user database A database system designed so that more than one user can access a file or record at the same time. The system requires concurrency checking so that two users don't attempt to update the same record at the same time.

natural join (SQL) A join which takes two database files (or tables) and creates a new, wider table consisting of all possible concatenated records (or rows), where the new table contains two identical columns, one of which is dropped.

nested queries (SQL) A single query consisting of both an outer and inner query. Allows for more efficient retrieval of data from large tables by combining multiple operations into one.

nesting

Placing one operation inside another, such as nesting a function within another by specifying the nested function as a parameter of the first.

non-Windows application

Any application which doesn’t require the Windows environment. Typically, a DOS program.

normalization

The representation of data entities in their simplest forms, for the purpose of quickest access and most efficient storage. The normalization process includes the elimination of redundant data groups, and the elimination of redundant data elements.

null value

A zero or empty value.

Object

An object is one instance of a Class. It has properties and behaviors inherited from the Class that define what it is.

OCX control

A custom window control for processing end user input or displaying data (a VBX that works).

ODBC

The Open Database Connectivity standard supported by many Windows applications. Provides a standard API for accessing multiple database file formats via replaceable file drivers, and Client/Server support. The ODBC SDK is published by Microsoft.

ODBC Administrator

A redistributable Microsoft application for adding, maintaining or deleting individual ODBC drivers within a system. Usually located in the Windows\System directory, the executable file name is ODBCADM.EXE.

ODBC Control Panel applet

A Windows Control Panel interface to the ODBC administrator.

ODBC driver

A driver library containing the individual functions supporting standard ODBC calls for a particular file format.

OLE

Object Linking and Embedding. A method for accessing data and documents created by one application, through a different application.

one-to-many relationship

A connection between two data entities in which there may exist one corresponding value in the primary key of one database file or table, to many identical values in the foreign key of another table. The relationship implicitly describes the direction of the relation. For example, the relation of states to cities implies a state may have many cities. Also called a parent-child relation.

one-to-one relationship

A connection between two data entities in which there may exist one and only one corresponding value in the primary key of one database file or table, to a single identical value in the foreign key of another table. For example, the relation of customer name to internet address. The data is usually split into two separate tables for storage savings; all customers have names, but only a minority have internet addresses.

option structure

(Clarion) A structure containing mutually exclusive controls, such as radio buttons.

origin

The upper left corner of a window or control, expressed in x,y coordinates (0,0).

orphan

A portion of text or data separated from its complementary preceding data by a page break.

outer join

(SQL) A join which includes all records from one database file, and only those records from another in which the values in a selected field (or fields) match those in the first.

overlay

(Clarion) A variable or field sharing the same location as another. Acts as a data “re-declaration, and provides more efficient storage. Most useful in “either/or” situations when a variable and its overlay are of similar types but utilize different pictures.

page footer

The section of a report composed after the last detail that will fit on a page has been composed.

page header

The section of a report composed before the first detail to print on a page.

page overflow

In Clarion, the point at which the report library composes enough data to complete a page; the library will either send the page to the Windows spooler at that point, or first check to verify there are no “widows,” if the application so specifies.

palette

The table of available colors which a given window may user for painting.

parameter

An argument or optional variable passed to a procedure.

PCX image

A standard graphics file format, offering moderate compression, originally developed by the Zsoft corporation. The Windows Paintbrush accessory supports this format.

pel

Equivalent to pixel; abbreviation for picture element. The smallest screen unit addressed in graphic mode; a dot.

pen

In Windows, the active drawing or painting element; you can set its color, size, etc.

picture token

(Clarion) A formatting string, which specifies a specific “picture” or masking format for displaying and editing variables. The picture token begins with the “@” character.

pixel

Equivalent to pel; abbreviation for picture element. The smallest screen unit addressed in graphic mode; a dot.

point size

A measurement expressed in points; one point equals 1/72nd inch, or 1/28 centimeter.

pointer

The mouse cursor. Or, an index entry which locates or “points” to the corresponding data record.

polymorphism

The ability of a base class method to call methods of classes derived from the base class—without knowing at compile-time exactly what method is actually going to be called.

popup menu

A menu that appears disconnected from other visual elements. Windows 95 and Clarion frequently displays popup menus when the user clicks the right mouse button. By convention, the menu is associated with the item clicked on.

prefix

(Clarion) A short identifying string for a data structure. Provides a method for resolving variable names when, for example, two database files include fields whose names are the same.

primary key

(SQL) A database field or expression which uniquely identifies each record in the table or database file.

print job

One complete task sent to the Windows print spooler (accessible from Print Manager).

print structure

(Clarion) The parts of a report structure, which include the group break structure, detail, header, footer, and form.

printer driver

An external library file containing low level instructions and functions by which the Windows GDI library sends specific commands to the printer.

printer font

A typeface resident in the printer’s RAM.

procedure

(Clarion) A set of executable statements which may be executed repeatedly.

progress bar

(Clarion) A control that displays a graphic representation of a dynamic value by progressively coloring in a rectangle as the value changes.

program MAP

(Clarion) The “layout” of modules, procedures and functions, which the compiler uses to logically assemble the file. The MAP structure contains the prototypes which declare the functions, procedures, and external source modules used in a PROGRAM or MEMBER module.

project system

(Clarion) The IDE component which tracks the modules which comprise the application to be built, including source code and external libraries. The Project System also stores the various pragma, compiler and linking options.

prompt A text label which normally appears near a screen control, to identify the control.

property (Clarion) An attribute of a window, control, or other Clarion object.

property assignment syntax (Clarion) Specific language format for setting or retrieving the value of a control property.

property sheet A dialog intended to allow the convenient grouping of closely related items in a single place.

prototype To define the parameter(s) and return data types for a procedure or function. Within Clarion, prototypes are defined within the MAP structure.

PUT statement (Clarion) A statement which executes an update to a given record, and writes it to disk.

query (SQL) An operation upon a database table which results in another table or subset of the first.

Query by Example A query built by "filling-in the blanks" in a form representing the fields in a database table. The end user types in "example elements" which represent the possible answers to the query.

queue (Clarion) A specialized memory structure containing a doubly-linked list of values.

RAD (Rapid Application Development) The construction of applications accelerated by the use of development management tools such as data dictionaries, and the reuse of programs and code wherever possible.

radio button A control for eliciting a mutually exclusive choice from an end user.

range constraint A bounds for a database operation limiting the operation to a set of records for which a given field falls within specified starting and ending values.

raster font A bitmapped typeface, stored as a pattern of dots.

read only (Clarion) A field or variable which is displayed but not modified.

RECORD (Clarion) A data structure representing one row in a database table.

redirection file (Clarion) A list of alternate subdirectories to search for source code, object or library files.

reference variable

(Clarion) An indirection to another data variable (the target). The reference variable label can substitute for the target variable anyplace in executable code. Depending upon the target data type, the reference variable may contain the address in memory of the target, or a more complex internal data structure.

referential integrity

The process by which an application “follows through” on an update to a key field in one file, to check its related record in another file. This maintains valid parent-child relationships within the database. The Application Generator can automatically generate the executable code to support referential integrity constraints when you select options in the Relate dialog.

region

A specialized control whose sole function is to provide a reference for a screen area in x,y coordinates.

registry

(Clarion) A specialized initialization file storing values and parameters in binary format. These come from the Templates and are used by the Application Generator.

relationship

A logical link between records in data files based upon a duplicate (linking) field.

report form

(Clarion) A report element defined once, when first composing the report, then printed on all pages of the report.

resource file

An external file containing data for a window control, such as an icon file.

restore button

A window control which resizes a window from a maximized state to the last size prior to maximizing.

rich text format (RTF)

A common word processing file format, originally designed for transportability between word processing systems across different operating systems. The default format for the source document for the Windows help file format.

ROLLBACK

(Clarion) To restore an earlier state of a database, undoing the effect of one or more active transactions. Restores data held in a temporary file managed by the file driver.

ROUTINE

(Clarion) A series of executable statements local to a procedure or function. Following execution of the ROUTINE, program control returns to the calling procedure or function.

run time library

A dynamic link library providing essential support for basic application functions. For example, the Clarion run-time library provides all the “housekeeping” functions such as checking message queues, and managing the allocation and deallocation of all device contexts (for windows and reports).

schema

The map or catalog of a database describing its files or tables, fields, and relations.

scope

A range of records selected for a given operation. Also, the “boundaries” beyond which a given variable is unavailable to another procedure or function.

scroll bar

Standard window control for changing the view of data within a window, displaying more of a document or application controls than currently visible.

SDK

Software Development Kit.

select

To indicate to the system that the next command should act upon an on screen object, by placing the mouse cursor over it and pressing the left mouse button.

SELECT statement

(SQL) A statement setting the fields and tables for viewing, and for subsequent operations.

SELECT statement

(Clarion) Sets the next control to receive input focus.

selected event

An event generated and sent to the ACCEPT loop when a control obtains focus.

sequential access

The ability to manipulate all the records in a database file or table in the sequence defined by the key or index.

server

A remote computer providing data storage or services to other linked computers.

SET statement

A Clarion language statement preparing a file for sequential processing upon a group of records.

SHARE.EXE

The MS-DOS executable responsible for supporting multi-user access to a single file.

sheet

(Clarion) A control that contains multiple tab controls. Designed to display multiple related “pages” of controls. See also property sheet.

sort

Physically rearrange all database records in a specified order, and store the results in a new database file or table.

source code file

(Clarion) A text file containing Clarion language statements in a structured format, which the compiler can compile and link into an executable program.

spin control

A specialized edit box control, with two “increaser” and “decreaser” controls, linked to an array of values. When the end user increases or decreases the control, it updates to display the next value in the array.

SQL

Structured Query Language; a database language for maintaining a relational database; most often utilized in mainframe and client/server applications.

stack memory

A portion of memory which usually stores the most recent parameter data utilized by procedures and commands executed by a program or application.

statement

A single executable command.

static text

A window control which displays a string constant, and never receives focus; primarily used for labeling other controls or displaying information and instructions.

static variable

(Clarion) A persistent variable, which maintains its value from one use within a procedure to the next.

status bar

An area of a window, usually found at the bottom, in which the program can display prompts and information.

standard behavior

(STD) (Clarion) A predefined set of operations associated with a menu command; the actions are automatically supported by the run-time library, without requiring specific code on the part of the application.

stream mode

A special mode for several of the Clarion database drivers which optimizes file input/output.

swap file

A system file maintained by Windows for maintaining virtual memory as required by the system.

syntax

A rule specifying the specific format of a language statement.

system colors

The default colors shared by all custom Windows palettes.

system date

The date maintained by the system clock.

tab

(Clarion) A control that defines one of several “pages” consisting of a group of other controls. These tab “pages” are designed to be displayed in a single tabbed dialog by a sheet control.

tab order

The sequence in which each control in a window gains focus upon a TAB key press.

table

(SQL) A structured collection of data, consisting of a row of fields or column headings plus zero or more rows of data. Each row contains exactly one value for each of the fields. Within Clarion, the table corresponds to a specific FILE, ALIAS, or VIEW structure.

tabular report

A listing of data labels and their corresponding values, arranged in a row of column labels, followed by additional rows of data arranged by column.

tag

For file drivers (such as FoxPro and dBase IV) supporting multiple indexes within the same index file, the indicator marking an individual index.

target file

Indicates to the project system the name of the application or library file to be built.

task

A currently executing Windows application.

template procedure

(Clarion) A pre-written source code module written in the Clarion Template Language, containing "boiler-plate" Clarion language code, instructions for processing it at code generation, plus a user interface for gathering the customization instructions from the developer.

text control

A multi-line edit control which automatically supports word wrap.

text file

An ASCII file.

text justification

A paragraph alignment style which lines up the edges of the paragraph at left, right, left and right, or centers the entire line.

third normal form

A test or measure of how closely a database meets relational theory tests for data normalization.

thread

In a multi-threaded operating system such as Windows NT, the thread is the basic entity to which the operating system allocates a slice of CPU time. The thread has access to the same code, data, and system resources as the task (program) which started it. Clarion START threads do not receive separate "timeslices" from Windows 3.1; the run time library "slices" the Clarion thread and "divides" it among the Clarion START threads.

thumb

The box control on a scroll bar.

timer

A Windows resource which can automatically send a message to an application at pre-defined intervals.

token

A structured symbol or series of symbols, recognized and parsed by the compiler. Operators, and variable names are examples of tokens.

toolbar

A horizontal or vertically arranged group of command buttons, and/or other controls, generally remaining accessible the entire time a program executes.

transaction

The logical event during which an input or entry to a database record, held for sequential management with other entries, is written to disk. Failure of any of the disk writes during the transaction would compromise the integrity of the database.

tree control

Displays a logically hierarchical list of items in collapsible outline format. In Clarion , a small square filled with a plus or minus symbol, followed by a folder, represents an expandable tree control.

untyped parameter

(Clarion) Within a function prototype, specifies the data type of a parameter is to be resolved at tun time.

USE variable

(Clarion) An attribute indicating a variable whose value should display in a window or report control.

validity check

An executable code procedure which checks end user input against an expression defining acceptable values for a given field.

VBX control

A custom window control for processing end user input or displaying data.

VCR controls

A set of icons designed for use in navigating a browse or list; the images on the controls bearing a similarity to the controls on a video cassette recorder.

vector font

A scalable typeface, such as a TrueType font.

vector graphic

A binary file representation of a graphic or line art; defines the image as a series of lines and curves, allowing for smooth resizing. Clarion supports the .WMF (Windows Metafile) vector format.

view

A virtual file containing selected fields from one or more related database files.

virtual table

A data table or view which exists in memory only, constructed from one or more tables or data files which may exist on disk.

watch variable

A variable designated for monitoring by the Debugger.

widow

A portion of text or data separated from its complementary following data by a page break.

window frame

The window boundary. Dialog window frames are not resizeable. End users can resize other windows by dragging the frame.

window pane

A specialized window which acts as a “part” of a greater window. This allows an end user to divide an active window into separate sections which may then be scrolled independently or in sync.

WinExec

The standard Windows API function for calling another application. Supported in Clarion via the RUN statement.

Wizard

A series of dialogs that guide the user through a process, supplying defaults and limiting the user options to only those still available after each decision point, thereby controlling and simplifying the process from the user’s perspective.

X axis

The horizontal axis. Used for locating controls; the leftmost pixel in a window is position zero.

Y axis

The vertical axis. Used for locating controls; the upper pixel in a window is position zero.

INDEX