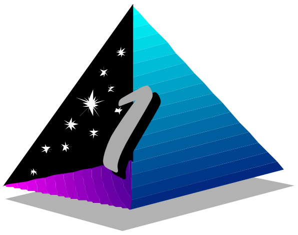
INTRODUCTION



This chapter provides:

- A “welcome” to Clarion for Windows.
- An overview of what you'll find in the *User’s Guide*.
- A guide to the information covered in the other books included with Clarion for Windows.
- Typeface and other document conventions.
- A reminder about product registration.
- A summary of our technical support programs.
- Information about Clarion's fax-back system for quick technical support.

WELCOME TO THE FAST TRACK FOR WINDOWS APPLICATION DEVELOPMENT

Welcome to Clarion for Windows. You have just purchased what TopSpeed Corporation believes is *the most powerful Windows application development tool on the market!* You can now build sophisticated Windows applications faster than you ever thought possible. This revolutionary development environment will dramatically increase your productivity. The executable programs you create with it run as fast as those tediously crafted in languages such as C, and you can connect to practically any existing database effortlessly.

You now have both a flexible Rapid Application Development (RAD) platform, *and* the power of the underlying Clarion language to create complex Windows applications. Its *Point and Click* development process removes you from the complexity and tedium of typical Windows programming platforms.

Clarion for Windows' Application Generator builds fully customized Windows programs in a fraction of the time other programming environments require, yet *makes coding optional*. The template driven environment provides extensive code reusability without the steep learning curve of object oriented programming.

The underlying Clarion language is a powerful yet easy to understand business oriented fourth generation programming language. Combined with our high performance database drivers, Clarion for Windows delivers the shortest development cycle *and* fastest executable for *your* project.

◆ Template Driven Rapid Application Development

Pre-written standard procedures—templates—provide customizable, reusable support for a wide range of functions such as browsers, forms, and reports. Just pick a template from a list, and fill in the prompts.

The templates are fully customizable to the way you want your applications to be. You can easily add your own, or add other third party templates to your template suite!

◆ WYSIWYG (What You See Is What You Get) Formatters

Use on-screen formatters to edit or add window and report controls to the template defaults. Easily customize list boxes, automatically associate database fields with entry controls, and choose actions for standard menu items, as you integrate them into the application.

Essential procedures such as data validation and referential integrity checks, time-consuming tasks on other platforms, are coded for you *automatically*. Windows development projects which normally take months using other tools take a fraction of the time with Clarion.

◆ Immediate Results—Build *Your* Application in 1 Hour

Clarion for Windows' Quick Start Wizard, Quick Load, and all the Procedure Wizards will build a dictionary, choose templates, then create an application for updating, maintaining and reporting data within 60 minutes after you tear the shrink wrap off this package. You'll create the *Quick Start Tutorial* application in chapter three of *Getting Started* without writing a single line of code.

◆ 4GL Development—C Performance

Unlike other RAD platforms, the executables you build with Clarion for Windows are *fast*. The TopSpeed compiler technology produces true Windows executables that *fly*.

The Clarion language has the power and flexibility to support any application development project. It's a structured, compact, extensible language that supports the best of component-oriented development. Use Visual Basic Custom Controls (.VBX's) in your application. Create dynamic link libraries which applications written in other languages can call.

For "hand-coders"—those writing applications with the Clarion language from scratch, rather than using templates—the development environment provides a wealth of tools. Window and Report formatters let you switch between graphically editing windows and reports, or editing their data structure declarations as text. The Text Editor provides color syntax highlighting. The Project Editor compiles and links at top speed with *TopSpeed* compilers, and the world-class Clarion Debuggers help your project reach perfection.

- Database Independence

  New high performance database drivers, rewritten as dyna-link libraries, support popular databases and accounting packages; including Xbase formats, Clipper, FoxPro, Btrieve, and others. On a local drive, they're far faster than the extra ODBC layer many Windows database applications use (which Clarion for Windows supports as well).

  Store your data in (or convert existing data to) our new TopSpeed file format for incredibly fast performance. It's efficient too; you can store multiple data files inside one physical DOS file, saving unnecessary use of end user file handles.

WHAT YOU'LL FIND IN THIS BOOK

The *User's Guide* shows you how to create applications using the Clarion for Windows *development environment*. It focuses on the development environment interface, tools, and templates. In other words, the *User's Guide* shows you how to build applications largely without writing code. The following lists the parts of this book and summarizes their content:

Introduction	The chapter you're reading now.
Setup	Lists system requirements and provides installation instructions.

Development Flow	An introduction to Clarion, describing the functional parts of the development environment, and an overview of the steps you take to create your applications.
Dictionary Editor	Introduces the concept of the data dictionary which defines the application's data files, their fields, keys, and much more. Clarion provides automatic entry options and data validity checking which saves many lines of code, via the data dictionary.
Application Generator	Describes the Application Generator, with which you design and generate code for your application's procedures and functions. Use the Application Generator by choosing procedure templates which most closely match your requirements, then customize them with your own controls, data, formulas, and embedded source.
Procedure Templates	Describes the procedure templates, their uses and how to incorporate them into your applications.
Other Templates	Describes the control, code, and extension templates as well as their uses and how to incorporate them into your applications.
Window Formatter	Describes how to use the Formatter to visually set the properties of your application windows, including size, appearance, menus and controls. The Window Formatter also allows you to add controls from Visual Basic (VBX) libraries.
Menu Editor	You access the Menu Editor through the Window Formatter. It allows you to create menus for your application. This chapter also explains how to create tool bars for your application.
Controls	Controls are the user interface elements you place in the application using the Window Formatter, Report Formatter, and Text Editor. These include entry fields, push buttons and graphical elements such as bitmaps. This chapter shows you how to customize each type of control.

List Formatter	Describes how to use the list box formatter to customize the appearance of your list box controls. It also provides instructions on adding functionality such as drag and drop to list box controls.
Report Formatter	The Report Formatter is the main means of creating reports with Clarion. You can place controls in the report in the same manner you place controls in a window. This chapter describes how to build reports for your applications.
Text Editor	Describes how to directly edit your source code files. The Text Editor features color coded syntax highlighting as well as other programmer conveniences, such as Search and Replace.
Formula Editor	Describes how to use the Formula Editor to quickly generate a statement assigning an expression to a value.
Project System	The Project System manages the compile and link process for your application. This chapter walks you through setting up a new project and explains the components of the project tree. This section also provides information on the files you need to distribute with your finished applications.
Debugger	Describes how to use the Clarion Debuggers to eliminate bugs from your programs.
Database Manager	Describes how to use this programmer’s tool to directly access data files without having to produce an application.
Appendices	Provides additional information on subjects related to but not integral parts of the Clarion for Windows development environment. These include a discussion of Windows design issues, file driver specifications, Multi-Programmer development, Open Database Connectivity (ODBC), plus a glossary.

WHERE TO FIND MORE INFORMATION

There are four books, plus extensive on-line help for Clarion for Windows.

- The *Getting Started* manual provides two step by step tutorials. The first uses the Clarion for Windows Quick Start Wizard to create an application *quickly*. The second provides a more extensive walk through the development environment, creating an Order Entry system.
- The *User's Guide* is the book you're reading now. It provides a task-oriented description of the development environment, arranged by its major components. It describes the templates that ship with this product, and provides additional appendices with information on topics such as the Clarion for Windows file drivers.
- The *Language Reference* is the complete guide to the Clarion language. It provides descriptions of all statements and functions, with examples for each. The Language Reference is organized by functional topics.
- The *Template Language Reference* documents the Clarion Template Language, clearly demonstrating how to write your own templates. To obtain a printed copy of the *Template Language Reference*, please see the product registration card.
- The on-line hypertext help appears when you press the F1 key, a **Help** button, or choose one of the commands on the **Help** menu. The on-line help is arranged by dialog box, to provide you with the precise help text, when you need it.

  The full text of the *Language Reference* is also on-line. When working with the Text Editor, place the insertion point on a Clarion language statement or function, then press the F1 key to view help for the item.

  The full text of the *Template Language Reference* is on-line too. Access this document through the main table of contents for the help system.

  Important: if any part of the on-line help text conflicts with the printed documentation, the on screen help takes precedence. TopSpeed Corporation makes every reasonable effort to ensure the printed documentation is up to date. However, lead-time required by printers may create a lag in the documentation; while we can update the help files that ship concurrently with a product revision, printed materials must 'catch up' later.

DOCUMENTATION CONVENTIONS

The documentation uses the typeface and keyboard conventions which appear below.

Typeface Conventions:

Italics	Indicates what to type at the keyboard, such as *Type This*.
SMALL CAPS	Indicates keystrokes to enter at the keyboard, such as ENTER or ESCAPE.
Boldface	Indicates commands or options from a pull down menu or text in a dialog window. Note: this style also uses a different typeface to match the helvetica bold face which Windows uses as the system font.
`LETTER GOTHIC`	Used for diagrams, source code listings, to annotate examples, and for examples of the usage of source statements.

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

Indicates critical information. If you read nothing else in this chapter, please read this.

Keyboard Conventions:

F1	Indicates a keystroke. Press and release the F1 key.
ALT+X	Indicates a combination of keystrokes. Hold down the ALT key and press the X key. Then release both keys.

REGISTERING THIS PRODUCT

Before you begin using Clarion for Windows, fill out and mail in the registration card that came in the package. This Business Reply Card makes you eligible to receive several important benefits. Once registered, you can use TopSpeed's Technical Support services, and you automatically receive new product announcements and update alerts.

TECHNICAL SUPPORT

You can receive unlimited free technical support for Clarion for Windows on CompuServe Information Service. Once connected to CompuServe, type GO TOPSPEED. TopSpeed employees, as well as TopSpeed Certified Support Partners (known as Team TopSpeed), will answer your questions in a timely manner. Additionally you will get advice and answers from other Clarion for Windows users. We strongly recommend that our customers take advantage of this service.

Paid telephone technical support is also available from TopSpeed Corporation. You can access our pay-per-call support by calling (900) 884-0444. Various paid support programs are also available. Call TopSpeed Corporation customer service at (800) 354-5444 or (305) 785-4555 for more information.

THE TOPSPEED FAX RETRIEVAL SYSTEM

TopSpeed also offers customers phone/FAX access to most often requested technical and marketing documents. Documents on-line include product brochures, technical documents, article reprints, price lists, and even a "What's Hot" update on TopSpeed products.

To request specific documents, dial (305) 785-4555, press 53, and listen for the FAX Retrieval System's instructions. You may also dial (305) 785-4556 (TopSpeed Standard Support Line), and press 1 to access the system. The menu is interactive and user-friendly. First time callers can request a list of available documents to review before making a selection. You can then enter the document code number, and the material will be immediately delivered to you. You can access the system directly from your FAX machine or from any touch-tone phone.

SETUP

This chapter explains the Clarion for Windows system requirements, setup process, and setup options. It also provides a brief introduction to the environment as you'll see it, the first time you start Clarion for Windows.

SYSTEM REQUIREMENTS

You can run the Clarion development environment on any system that meets the minimum system requirements for Microsoft Windows 3.x, Windows 95™, or Windows NT 3.51.

- Windows 3.x, 8 Megabytes of RAM recommended.
- Windows 95, 12 Megabytes of RAM recommended.
- Windows NT 3.51, 16 Megabytes of RAM recommended.
- Minimum of 8 to 20 Megabytes free hard disk space, depending on the Setup options you select.

The applications you develop with Clarion for Windows will execute comfortably on computers that meet only the minimum requirements for these operating systems.

THE SETUP PROGRAM

The Setup program, on disk one of your installation disks, decompresses and copies the Clarion for Windows files to your hard drive.

- For all the target operating systems, it provides you with options for installing the various components, such as the example files.
- It *asks* before updating the PATH statement in your AUTOEXEC.BAT file to include the Clarion for Windows directory.

- In Windows 3.x, it installs Program Manager icons for the Clarion development environment, Debugger, Help files, and ReadMe files.
- In Windows 95, it installs the Clarion development environment, Debugger, Help files, and ReadMe file icons to the **Start ➤ Programs** menu.

Starting Setup

To start the Clarion for Windows Setup program in Windows 3.x:

1. Insert disk one of the installation disks into your floppy drive.
2. From Program Manager, File Manager, or other shell program capable of launching a program, choose **File ➤ Run**.
3. Type *A:\SETUP* (or *B:\SETUP*) in the **Run** dialog, and press the **OK** button.

   The Setup program provides an introductory screen and other text information. It prompts you for your desired Setup Options.

To start the Clarion for Windows Setup program in Windows 95:

1. Insert disk one of the installation disks into your floppy drive.
2. From the Start menu, choose **Settings ➤ Control Panel**.
3. Choose **Add/Remove Programs** then press the **Install** button.

   The Windows 95 Wizard will direct you through the installation process.

Setup Options

After starting Setup, you'll see a screen displaying a number of options.

1. Choose the Setup options by checking the boxes you want, then press the **OK** button.
2. Specify the target drive and directory, then press the **OK** button.

   Setup will install the main components of the Clarion Development Environment to a BIN subdirectory one level below the target directory you specify in the dialog.

   The Clarion for Windows Setup program installs *all* files to the target directory, and subdirectories beneath it. It installs *no* files to any other directory.

   During the installation, progress bars will display as Setup copies the files.
3. Choose **Yes** or **No** when Setup asks whether to modify the PATH for you.

   For Windows 3.x, the Clarion for Windows development environment requires that the BIN subdirectory be listed in the PATH environment variable. If you choose **No**, you *must* edit the AUTOEXEC.BAT file manually.

   The only other change to any of your system files is that Clarion for Windows adds its own section to WIN.INI (Windows' initialization file) when you run it for the first time.
4. Choose **Yes** or **No** when Setup asks whether to display the ReadMe file.

   If you don't wish to read it right away, you'll find an icon for it in the Program Manager group (or the **Start ➤ Programs** menu) which Setup creates for you. We recommend reading it as soon as Setup has copied all the files.
5. Press the **OK** button when Setup is done.

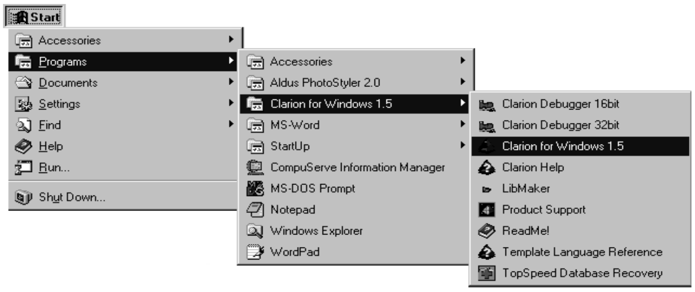
STARTING CLARION FOR WINDOWS

To start Clarion for Windows, locate the Clarion for Windows icon (Purple Pyramid) in the Clarion for Windows program group, and DOUBLE-CLICK on it:

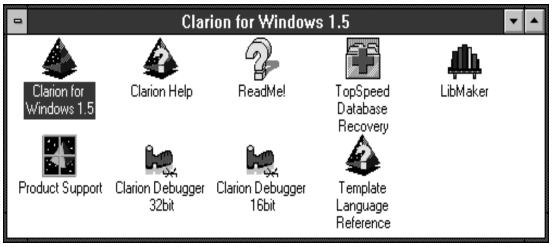
The Clarion for Windows development environment appears, ready for you to begin work.

- ◆ You'll find a quick guide to the development environment parts—such as a diagram of the tool bar icons—in chapter three of this book.

To run Clarion for Windows, navigate to Clarion for Windows with the Start menu:



or, locate the Clarion for Windows icon in the Clarion for Windows program group, and double-click it:



The Clarion for Windows development environment appears, ready for you to begin work.

- ◆ You'll find a quick guide to the development environment parts—such as a diagram of the tool bar icons—in the next chapter.

QUICK ACCESS TO FILES AND FUNCTIONS

Probably the first thing you'll notice when you start Clarion for Windows is the tool bar. The tool bar provides "short cuts" to commonly used commands which manage your files and/or projects.

The Development Environment Tool Bar

The Development Environment tool bar includes the following icon buttons:

The Pick button.

This button opens the Pick list for all types of files, as described further in the section below. Pressing this button is equivalent to choosing **File ➤ Pick** from the menu.

The New button.

This button calls the **New** dialog, which allows you to create a new application, Clarion source code file, data dictionary, project file or text file. Pressing this button is equivalent to choosing **File ➤ New** from the menu.

The Open button.

This button calls the standard **Open** file dialog, allowing you to "walk" the directory tree to locate a file of your choice. Pressing this button is equivalent to choosing **File ➤ Open** from the menu.

The Save button.

This button saves the current file, of whichever type is open. When you run Clarion for Windows for the first time, this button is disabled, because there are no open files. Pressing this button is equivalent to choosing **File ➤ Save** from the menu.

The Make button.

This button compiles and links the current project or application. The Make window details the progress of the operation. Pressing this button is equivalent to choosing **Project ➤ Make** from the menu.

The Run button.

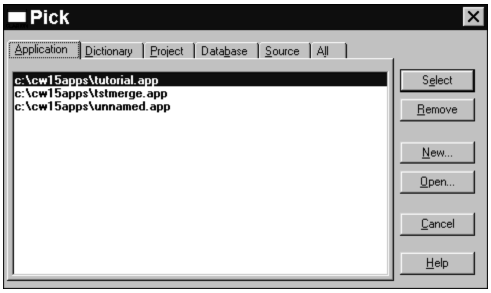
This button compiles and links (if your current settings include **Auto Make Before Run**) the current project or application, then runs it. Pressing this button is equivalent to choosing **Project ➤ Run** from the menu.

The Debug button.

This button compiles and links (if your current settings include **Auto Make Before Run**) the current project or application, then loads the Clarion for Windows Debugger. Pressing this button is equivalent to choosing **Project ➤ Debug** from the menu.

The Pick List

The **Pick** dialog lists all the most recently used files in a list box categorized by Application, Dictionary, Project, Database, Clarion Source, and All. Each of these tabs displays a pick list of up to twenty of the most recently used files of that type:

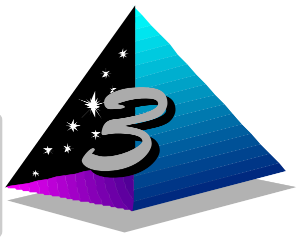


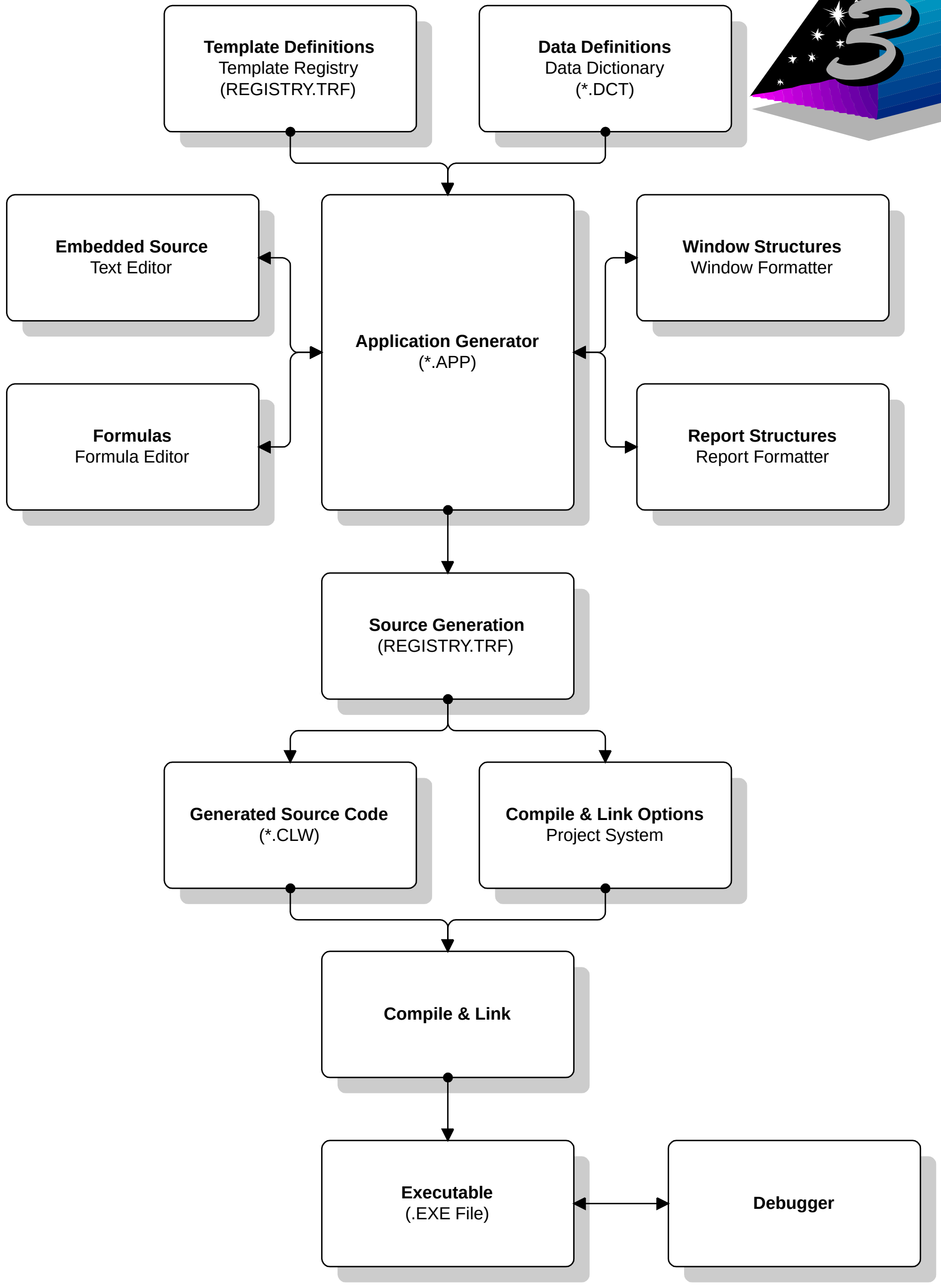
c:\cw15apps\tutorial.app
c:\cw15apps\tstmerge.app
c:\cw15apps\unnamed.app

The **Pick** dialog provides the following buttons:

Select	Opens the currently selected file.
Remove	Removes the currently selected file from the Pick list.
New	Allows you to create a file.
Open	Allows you to open a file not on the Pick list.

DEVELOPMENT FLOW





This chapter provides an overview of how the Application Generator ties everything—the Clarion language and the parts of the development environment—together.

The Application Development Flow chart on the previous page depicts how the working parts of the development environment connect with each other when you use the Application Generator to develop your application.

CLARION PROGRAMMING

Clarion is a fourth-generation (4GL), business oriented *programming language* specially designed for Rapid Application Development (RAD).

Clarion for Windows is a complete *development environment* that helps you do everything from designing your data dictionary, to generating Clarion source code, to supplying reusable code, to managing the compile, link, and distribution of your files.

As implemented in Clarion for Windows, the Clarion language automatically handles the Windows “housekeeping” chores that many other Windows programming languages leave to you.

File driver independence is built into the language; Clarion for Windows contains dynamic link library drivers for most popular PC database formats, plus other drivers are available as add-ons.

Template Driven

Clarion’s Application Generator is template driven. The various templates provide many of the benefits of object oriented programming, especially reusability, yet without the overhead of learning an object oriented language.

The template registry (REGISTRY.TRF) stores pre-written executable code and data structures which can be customized and reused. You can modify the default Clarion templates and store your modifications in the template registry. You may also add third party templates and use them *in addition to, and along with,* the Clarion templates.

Procedure Templates

A *procedure* is a stored series of Clarion language statements which perform a task. A *Procedure template* is an *interactive tool* that requests information from the developer, then generates a procedure customized for just the task the developer needs to accomplish.

Clarion for Windows provides a rich assortment of Procedure templates with which you can rapidly develop database applications. In *Getting Started*, the *Quick Start Tutorial* introduces a few procedure templates; the *Hands On Tutorial* introduces more. You identify the procedure template that generates code closest to the task you want to perform, then customize it with the other development environment tools. The procedures can include elements such as browse windows to view groups of records, and form windows, to edit one record at a time.

To incorporate a procedure into an application, you select a procedure template from the registry and use it to add the new procedure to the .APP file (the file where the Application Generator stores all procedures). If the procedure drives a window with a menu, the menu actions are automatically added to the application and marked as "ToDo."

The usual way to customize a procedure is to call one of the formatters—the Window Formatter or Report Formatter—and add a new window or a new control to a dialog box. The formatters are visual design tools: to place a command button in a dialog box, you pick the button tool from a toolbox, then click in the dialog box under construction to place the button.

Another way you may customize a procedure is to add embedded source code. The Application Generator displays a tree diagram showing the main locations where source can be embedded, including before, during, and after the procedure, plus at each event the window or entry fields in the procedure may generate. You can pick a precise spot to execute the code, then "hand code" it, or use "code templates" to generate the code for you.

Control Templates

A *control* is almost anything you see on a window or a report. For example, a check box, a push button, an entry field, and a list box are all controls.

Control templates **create controls** ***and*** **the executable code for maintaining them. The source code they generate can, for example, load the data from a file into a QUEUE, then display the data in a list box.**

Code and Extension Templates

Code templates are executable code fragments with functionality related to a procedure rather than to a specific control. For example the DateTimeDisplay template can display a clock on an action bar, or a date on a window. Each typically provides you with on screen instructions on how to incorporate its functionality into the application.

The Application Generator generates your application’s source code from the templates, plus any customized code (embedded source code) you provide. The Project System then compiles and links to create the application.

THE DEVELOPMENT PROCESS

At a very high level, application development requires analysis of a situation, followed by design and implementation of a solution.

Analysis requires identification and segmentation of the data and of the processes that manipulate the data.

Implementation of a good solution requires providing a cost effective method of performing the processes identified during analysis.

Clarion for Windows is designed to facilitate the implementation of efficient data processing solutions. Each of the parts of the development environment plays a specific role in the implementation process.

CLARION’S DEVELOPMENT ENVIRONMENT

The development environment contains seven main functional parts, all of which are accessible from the others. When using the Application Generator, buttons in the various dialogs lead to the other parts. The Application Development Flow chart at the beginning of this chapter pictures how the parts interact with each other and the template registry, with the Application Generator at the center of the whole process.

This section provides a description of each part, in the order that a typical programmer using the Application Generator might encounter it. Each contains dialog boxes which the programmer fills out to “describe” the Application’s functionality to the Application Generator. On your command, the Application Generator generates the specified source code, and the Project System compiles and links it to make an executable program.

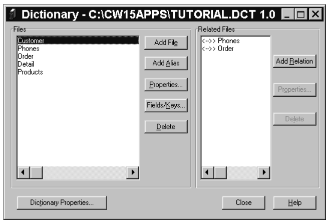
Programming in Clarion for Windows is, in many ways, a personal journey through a series of dialog boxes. There is no mandatory sequence in which you must "fill in" the dialogs, though some are prerequisites for others. If you know which dialogs do what, it makes building your application that much quicker.

The Dictionary Editor

The Data Dictionary (a .DCT file), maintained by the Dictionary Editor, holds a description of the database, including its files, keys, indexes, database drivers, fields, file relations, field validation rules, referential integrity constraints, and more. *It's the first file you create when you design your application.*

You can create the file definitions "from scratch" (using Quick Load or not), or import definitions from existing data files.

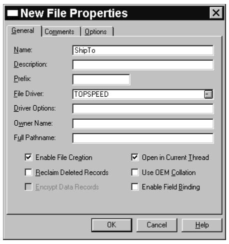
The other parts of the development environment use the information in the dictionary to let you, for example, easily place data fields in a dialog box you design for the end user. The Application Generator creates code for all the statements that access the data files based on how you construct the Data Dictionary.


Dictionary - C:\CW15APPS\TUTORIAL.DCT 1.0

Files:
Customer
Phones
Order
Detail
Products

Related Files:
<-->> Phones
<-->> Order

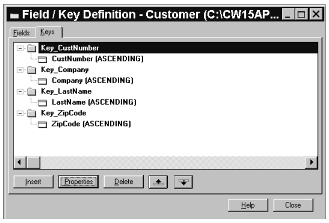
Start a new dictionary with the **File ➤ New** command, then select Dictionary. This leads you to the **Dictionary** dialog. Define your application's data files, aliases, and views in this dialog. It also shows the relationships between files. Buttons lead to the **New File Properties**, the **New File Alias**, and the **New Relationship** dialogs.


New File Properties

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

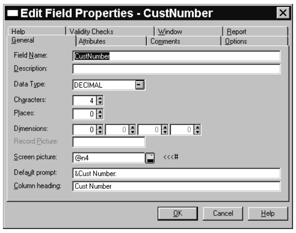
[✓] Enable File Creation
[✓] Open in Current Thread
[ ] Reclaim Deleted Records
[ ] Use OEM Collation
[ ] Encrypt Data Records
[ ] Enable Field Binding

Specify the name and file driver for each data file, one by one, in the **New File Properties** dialog. It also allows you to set options such as Threaded, which specifies that each execution thread accessing the file gets its own record buffer. This is useful for MDI applications.



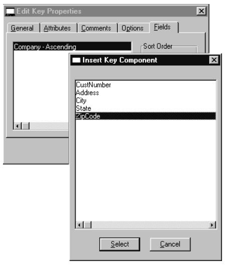
Key_CustNumber
CustNumber (ASCENDING)
Key_Company
Company (ASCENDING)
Key_LastName
LastName (ASCENDING)
Key_ZipCode
ZipCode (ASCENDING)

From the **Field/Keys Definition** dialog, press the **Insert** button to specify fields, keys, and index files. All the information is arranged hierarchically.



Field Name:	CustNumber
Description:	
Data Type:	DECIMAL
Characters:	4
Places:	0
Dimensions:	0 0 0 0
Record Picture:	
Screen picture:	@n4 <<<#
Default prompt:	&Cust Number:
Column heading:	Cust Number

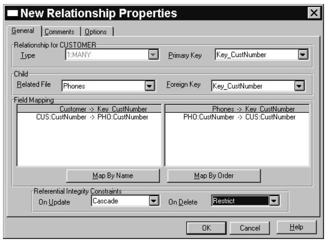
Define fields, their data types and sizes in the **New Field Properties** dialog. You can pre-define control properties, such as text justification. Two buttons provide shortcuts to all the properties dialogs in the Window and Report Formatters. You can also “back up” to the previous dialog to define keys and relationships.



Company - Ascending

CustNumber
Address
City
State
ZipCode

Specify the key components in the **Key Components** dialog. Clarion for Windows automatically builds the key correctly even if you specify multiple field types. From here, you “back up” to the **Dictionary** dialog to define relationships.



Relationship for CUSTOMER
Type: 1:MANY
Primary Key: Key_CustNumber

Child
Related File: Phones
Foreign Key: Key_CustNumber

Field Mapping

Customer -> Key_CustNumber	Phones -> Key_CustNumber
CUS:CustNumber -> PHO:CustNumber	PHO:CustNumber -> CUS:CustNumber

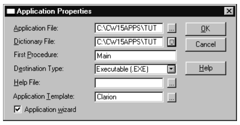
Referential Integrity Constraints
On Update: Cascade
On Delete: Restrict

Define relationships in the **New Relationship Properties** dialog. You can also specify Referential Integrity Constraints from controls in this dialog. With the major parts of the dictionary defined, you save the dictionary and move on to the .APP file.

The Application Generator

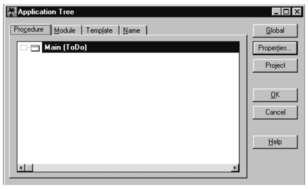
The Application Generator generates your application’s source code, one procedure at a time, *based on the templates you pick from the template registry*. It allows you to add global and local memory variables, and customize the procedures with visual design tools and embedded source code.

The Application Generator provides access to the other parts of the development environment so you can customize the look and functionality of the windows, menus, reports and other user interface elements.



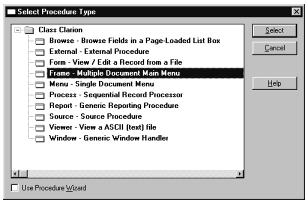
Application File: C:\CW15APPS\TUT
Dictionary File: C:\CW15APPS\TUT
First Procedure: Main
Destination Type: Executable (.EXE)
Help File:
Application Template: Clarion
☑ Application wizard

Start a new application with the **File ➤ New** command, then select **Application**. This allows you to set the basics—application name, data dictionary name, help file and the application template—in the **Application Properties** dialog. This creates the .APP file and displays the Application Tree.



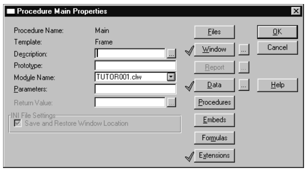
Main (ToDo)

View and maintain the parts of your application in the **Application Tree** dialog. It hierarchically displays your application's procedures, and marks the ones still to be defined as "ToDo." Press the **Global** button to define global memory variables.



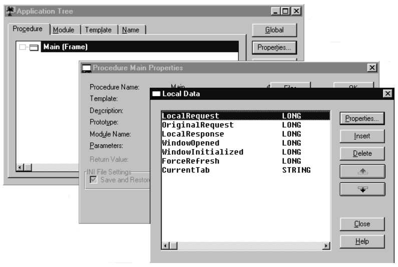
Class Clarion
Browse - Browse Fields in a Page-Loaded List Box
External - External Procedure
Form - View / Edit a Record from a File
Frame - Multiple Document Main Menu
Menu - Single Document Menu
Process - Sequential Record Processor
Report - Generic Reporting Procedure
Source - Source Procedure
Viewer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

Define the functionality for a "ToDo" procedure in the **Select Procedure Type** dialog. Procedure templates such as Browse and Form appear in a list. The **Select** button brings up the **Procedure Properties** dialog.



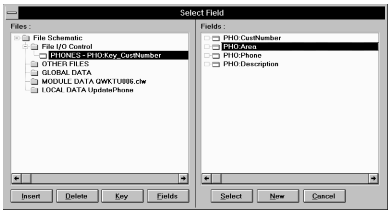
Procedure Name: Main
Template: Frame
Description:
Prototype:
Module Name: TUTOR001.clw
Parameters:
Return Value:
INI File Settings
☑ Save and Restore Window Location

The **Procedure Properties** dialog is the hub for all the other dialogs that let you customize the procedure so that your application does the job the way you want. Press the **Data** button to define local memory variables.



```
LocalRequest          LONG
OriginalRequest       LONG
LocalResponse         LONG
WindowOpened          LONG
WindowInitialized     LONG
ForceRefresh          LONG
CurrentTab            STRING
```

Define and set the order the program initializes local memory variables in the **Data** dialogs. Press the **Insert** button to define variable name, type, size etc., in a dialog box identical to the **New Field Properties** dialog.



Select Field

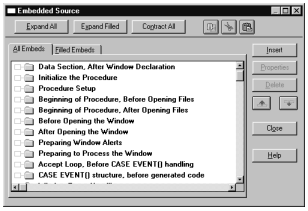
Files :

- File Schematic
  - File I/O Control
    - PHONES - PHO:Key_CustNumber
  - OTHER FILES
  - GLOBAL DATA
  - MODULE DATA QWKTU006.clw
  - LOCAL DATA UpdatePhone

Fields :

- PHO:CustNumber
- PHO:Area
- PHO:Phone
- PHO:Description

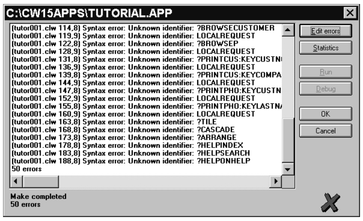
Select the files, keys, aliases, views, and fields the procedure or control will access in the **Select Fields** dialog.



Embedded Source

- Data Section, After Window Declaration
- Initialize the Procedure
- Procedure Setup
- Beginning of Procedure, Before Opening Files
- Beginning of Procedure, After Opening Files
- Before Opening the Window
- After Opening the Window
- Preparing Window Alerts
- Preparing to Process the Window
- Accept Loop, Before CASE EVENT() handling
- CASE EVENT() structure, before generated code

Press the **Embeds** button to display the **Embedded Source** dialog. This allows you to insert custom executable code at points before, during, and after the procedure, or on window and field-specific events. Select the embed point and press the **Insert** button.



C:\CW15APPS\TUTORIAL.APP

```
(tutor001.clw 114,8) Syntax error: Unknown identifier: ?BROWSECUSTOMER
(tutor001.clw 119,9) Syntax error: Unknown identifier: LOCALREQUEST
(tutor001.clw 122,8) Syntax error: Unknown identifier: ?BROWSEP
(tutor001.clw 128,9) Syntax error: Unknown identifier: LOCALREQUEST
(tutor001.clw 131,8) Syntax error: Unknown identifier: ?PRINTCUS:KEYCUSTN
(tutor001.clw 136,9) Syntax error: Unknown identifier: LOCALREQUEST
(tutor001.clw 139,8) Syntax error: Unknown identifier: ?PRINTCUS:KEYCOMPA
(tutor001.clw 144,9) Syntax error: Unknown identifier: LOCALREQUEST
(tutor001.clw 147,8) Syntax error: Unknown identifier: ?PRINTPHO:KEYCUSTN
(tutor001.clw 152,9) Syntax error: Unknown identifier: LOCALREQUEST
(tutor001.clw 155,8) Syntax error: Unknown identifier: ?PRINTPHO:KEYLASTN
(tutor001.clw 160,9) Syntax error: Unknown identifier: LOCALREQUEST
(tutor001.clw 163,8) Syntax error: Unknown identifier: ?TILE
(tutor001.clw 168,8) Syntax error: Unknown identifier: ?CASCADE
(tutor001.clw 173,8) Syntax error: Unknown identifier: ?ARRANGE
(tutor001.clw 178,8) Syntax error: Unknown identifier: ?HELPINDEX
(tutor001.clw 183,8) Syntax error: Unknown identifier: ?HELPSEARCH
(tutor001.clw 188,8) Syntax error: Unknown identifier: ?HELPONHELP
50 errors
```

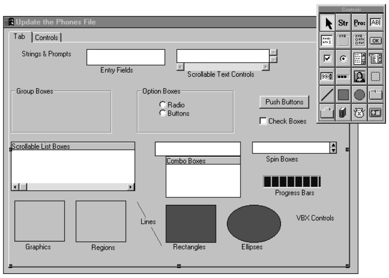
```
Make completed
50 errors
```

After you've customized the procedure template using the Window Formatter, Report Formatter and/or Text Editor, you can return to the Application Tree and generate the code!

The Window Formatter

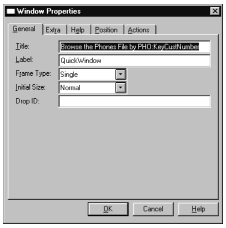
You visually design your application's windows and controls—everything the end user sees—in the Window Formatter. It automatically generates source code for the elements you visually design on screen.

When using the Application Generator, you'll call the Window Formatter by pressing the **Window** button in a **Procedure Properties** dialog.



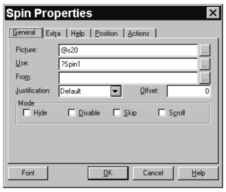
The Window Formatter provides a view of the window under development. CLICK in the toolbox, then CLICK in the window to place a new control.

You can then use the **Preview!** command to see exactly how the window appears to the end user.



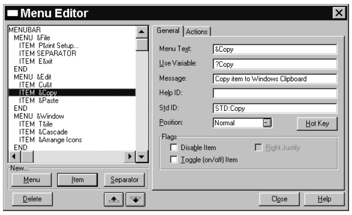
Title: Browse the Phones File by PHO:KeyCustNumber
Label: QuickWindow
Frame Type: Single
Initial Size: Normal
Drop ID:

Each window and each control in the window has an associated property dialog that defines its behavior and appearance, and if an entry field, its contents. RIGHT-CLICK the control or window to access it's property dialog. The **Window Properties** dialog sets basic elements such as system menus, caption, size, shape, scroll bars, etc.



Picture: @s20
Use: ?Spin1
From:
Justification: Default
Offset: 0
Mode
☐ Hide ☐ Disable ☐ Skip ☐ Scroll

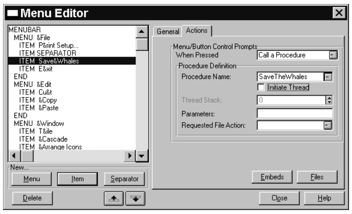
A typical control property dialog sets options such as a label, size, shape, color, font, and, for entry fields, a variable to reference its contents.



```
MENUBAR
  MENU  &File
    ITEM  P&rint Setup...
    ITEM SEPARATOR
    ITEM  E&xit
  END
  MENU  &Edit
    ITEM  Cu&t
    ITEM  &Copy
    ITEM  &Paste
  END
  MENU  &Window
    ITEM  T&ile
    ITEM  &Cascade
    ITEM  &Arrange Icons
  END
```

Menu Text: &Copy
Use Variable: ?Copy
Message: Copy item to Windows Clipboard
Help ID:
Std ID: STD:Copy
Position: Normal
Flags
☐ Disable Item ☐ Right Justify
☐ Toggle (on/off) Item

Choose **Menu ➤ New Menu** to call the Menu Editor. Edit the menu text, and new menu items with the **Item** button, and specify menu item functionality by selecting a procedure to execute when the item is selected, or choose from built in standard windows actions such as Cut, Copy, and Paste.

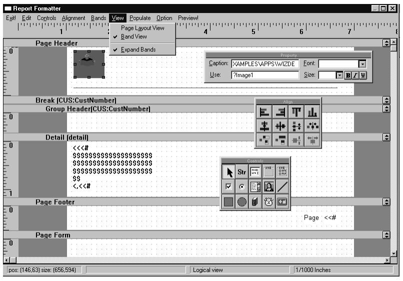


```
MENUBAR
  MENU  &File
    ITEM  P&rint Setup...
    ITEM SEPARATOR
    ITEM  Save&Whales
    ITEM  E&xit
  END
  MENU  &Edit
    ITEM  Cu&t
    ITEM  &Copy
    ITEM  &Paste
  END
  MENU  &Window
    ITEM  T&ile
    ITEM  &Cascade
    ITEM  &Arrange Icons
```

Menu/Button Control Prompts
When Pressed: Call a Procedure
Procedure Definition
Procedure Name: SaveTheWhales
☐ Initiate Thread
Thread Stack: 0
Parameters:
Requested File Action:

Use the **Actions** tab to associate a procedure call with a menu item, so when the user selects the menu command, it executes the procedure.

The Report Formatter



Caption: XAMPLES\APPS\WIZDE
Use: ?Image1
Font:
Size:

```
<<<#
$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$
$$
<,<<#
```

Page <<#

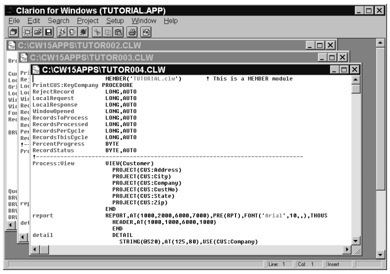
pos: (146,63) size: (656,594) | Logical view | 1/1000 Inches

The Report Formatter works in sync with the Application Generator in much the same way as the Window Formatter. You place controls in a sample report page. At run time, the print engine processes the records, handling page breaks, group breaks, headers, and footers as specified.

The Text Editor

The Text Editor is a full function programmer's editor in which you can handwrite source code.

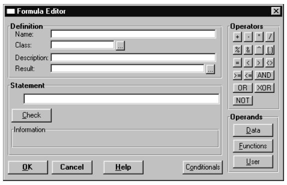
Most likely, when using the Application Generator, you'll call the Text Editor to create embedded executable source code to customize the way a procedure operates.



```
                    MEMBER('TUTORIAL.clw')        ! This is a MEMBER module
PrintCUS:KeyCompany PROCEDURE
RejectRecord        LONG,AUTO
LocalRequest        LONG,AUTO
LocalResponse       LONG,AUTO
WindowOpened        LONG,AUTO
RecordsToProcess    LONG,AUTO
RecordsProcessed    LONG,AUTO
RecordsPerCycle     LONG,AUTO
RecordsThisCycle    LONG,AUTO
PercentProgress     BYTE
RecordStatus        BYTE,AUTO
!-----------------------------------------------------------------------------
Process:View        VIEW(Customer)
                      PROJECT(CUS:Address)
                      PROJECT(CUS:City)
                      PROJECT(CUS:Company)
                      PROJECT(CUS:CustNo)
                      PROJECT(CUS:State)
                      PROJECT(CUS:Zip)
                    END
report              REPORT,AT(1000,2000,6000,7000),PRE(RPT),FONT('Arial',10,,),THOUS
                      HEADER,AT(1000,1000,6000,1000)
                      END
detail                DETAIL
                        STRING(@S20),AT(125,80),USE(CUS:Company)
```

The Editor features color coded syntax highlighting, making it easier to identify the different parts of the Clarion language statements for editing purposes. It also has full text search and replace capabilities, along with all the standard editing tools.

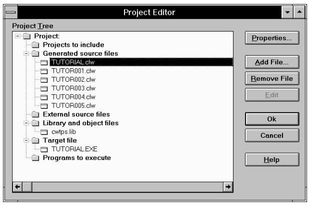
The Formula Editor



The Formula Editor helps you quickly generate and manage simple or complex assignment statements. The Formula Editor provides syntax checking, plus instant access to all the variables, functions, and operators that are used in assignments.

The Project System

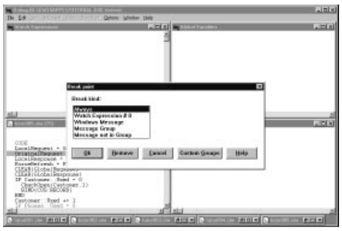
The Application Generator automatically creates the project file for the application. The project file contains compile and link options, such as whether to include debug code, optimization choices, external drive files, and so on.



The Project Tree displays the source code files, libraries and other external files included in the compile and link process. Press the **Properties** button to set specific options. When creating an application with the Application Generator, the Project file is maintained by the Application Generator.

The Debuggers

Debugging a program usually requires running the program and repeatedly stopping it to examine the value of different variables. The debuggers (16bit and 32bit) have of a number of windows which display source code, variable contents, active procedures and more.



Tell the project system to include debug information in the .EXE file, then start the debugger by pressing the **Debug** button in the **Compile Results** dialog, or choosing **Project ➤ Debug**.

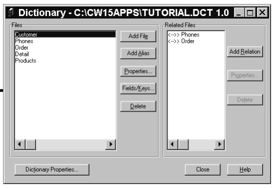
The simplest way to debug your application is to identify the part of the program that you think is producing the bug, and set a breakpoint, at that part of the code.

You can then run the program, and the Debugger will suspend it at the break point so that you can examine the values of the variables. This will help you pinpoint the problem so that your application is perfect!

USING THE DICTIONARY EDITOR

Contents

In this chapter, you will learn how to create a Database Dictionary. This defines the application's data files, data fields, keys, entry validation rules, entry options, file relationships, and referential integrity constraints.

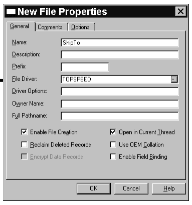


Dictionary - C:\CW15APPS\TUTORIAL.DCT 1.0

Files: Customer, Phones, Order, Detail, Products

Related Files: <-->> Phones, <-->> Order

To create the dictionary, you must define the individual files that make up the database. See Adding Files to the Dictionary.

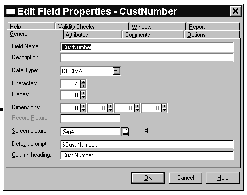


New File Properties

Name:	ShipTo
Description:	
Prefix:	
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

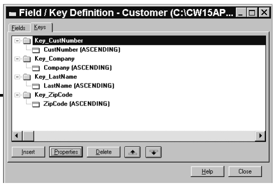
Name each file and choose its file driver. See the New File Properties dialog.



Edit Field Properties - CustNumber

Field Name:	CustNumber
Description:	
Data Type:	DECIMAL
Characters:	4
Places:	0
Dimensions:	0 0 0 0
Record Picture:	
Screen picture:	@n4 <<<#
Default prompt:	&Cust Number:
Column heading:	Cust Number

Add fields and define field properties, including Entry and Validation options. See Adding or Modifying Fields.



Field / Key Definition - Customer (C:\CW15AP...

- Key_CustNumber
  - CustNumber (ASCENDING)
- Key_Company
  - Company (ASCENDING)
- Key_LastName
  - LastName (ASCENDING)
- Key_ZipCode
  - ZipCode (ASCENDING)

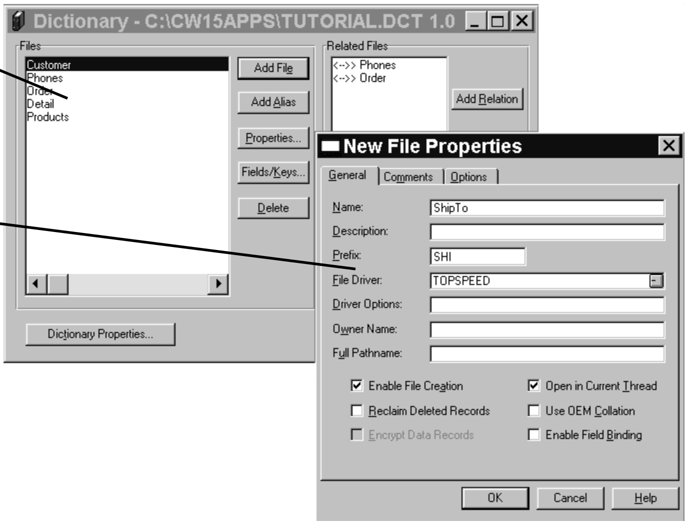
Organize records based upon the contents of a field or combination of fields by defining keys. See Adding or Modifying Keys.

This chapter shows you how to set up a data dictionary. The Application Generator generates Clarion language statements for a wide range of your application’s functionality based on how you construct your data dictionary. This chapter explains:

- What a data dictionary is and does.
- How the data files you design and options you choose in the dictionary dialogs can determine the efficiency of your application’s data storage. This includes a brief discussion of relational database theory.
- How to set data validation and entry options for end user data entry.
- How to specify default screen control options. Clarion even allows you to 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.

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

Define each data file in a separate File Properties dialog.



Files: Customer, Phones, Order, Detail, Products

Related Files: <-->> Phones, <-->> Order

Name: ShipTo
Description:
Prefix: SHI
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

WHAT A DATA DICTIONARY IS

The Data Dictionary is the central repository for information about your application’s files and the fields within those files. The information stored includes how and where the data will be stored on disk, as well as how the data will be presented to end users on reports and computer screens.

The Dictionary file (.DCT) stores

- file names
- file descriptions
- file structures
- file keys
- file indexes
- file relationships
- field data types
- field descriptions
- field prompt values (how the data is presented to the user)
- field validation
- field entry pictures
- field report column titles
- status bar messages
- 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 gives your application a consistent look and feel, so that end users have shorter learning curves.

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 standard method of handling each file and each field which will be used *every* time you reference the file or field. 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; but *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 **Edit (or New) File Properties** dialog.

- ❏ Manage fields and keys in the **Field/Key Definition** dialog.
- ❏ Define specific fields and the types of data they hold in the **Edit (or New) Field Properties** dialog.
- ❏ Define specific file relationships in the **Edit (or New) Relationship Properties** dialog.

OVERVIEW: 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 and define the fields in each file:*

1. Choose **File ➤ New** from the development environment menu, then select the **Dictionary** tab.
2. Specify the path (**Folders**) and **File Name** for your dictionary file, then press the **Create** button.
3. Press the **Add File** button, then, when asked if you want to use Quick Load, press the **No** button.

   The **New File Properties** dialog appears. See the *A Word About Quick Load* section below 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.
5. Press the **Fields/Keys** button to open the **Field/Key Definition** dialog.
6. On the **Fields** tab, press the **Insert** button.

   The **New Field Properties** dialog appears.
7. On the **General** tab, type in the field **Name**, choose the **Data Type**, specify the length in **Characters**.
8. Select the **Validity Checks** tab, and choose an option for entry validation for the field.
9. Select the **Window** tab to specify how the field and its prompt will appear in your application windows and dialogs.

10. Select the **Report** tab to specify how the field will appear on a printed report.

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

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

12. Repeat steps 7 through 11 for additional fields within this file.

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

13. Press the **Cancel** button in the blank **New Field Properties** dialog that appears after adding the last field, to return to the **Field/Key Definition** dialog.

❑ *Define the Keys in your files*:

1. From the **Keys** tab, 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 made up of one *or more* fields in your file. Thus the **Insert Key Components** dialog allows you to specify which fields will become part of your key.

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

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

5. Repeat steps 2 though 4 for other keys in this file.

6. Press the **Cancel** button to return to the **Field/Key Definition** dialog.

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

Repeat the above sequence defining files, fields, and keys for each additional file in your database.

❑ *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. Press the **OK** button to return to the **Dictionary** dialog.
7. Choose **File ➤ Save As** to save the .DCT file.

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
```

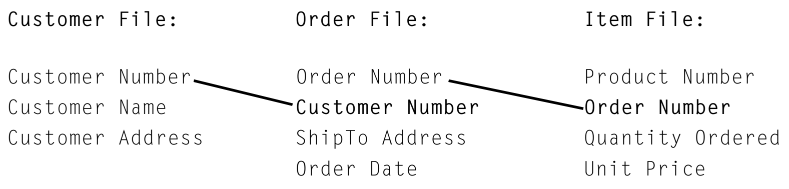
This organizes the data in a logical scheme and eliminates duplication. The process of relating each record to another record in another file requires adding fields to at least two of the files, so that the files can share common values. This will be discussed in a section below.

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:



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 Data 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 allows you to "preconstruct" some of the relational operations specified by database theorists; Clarion language statements handle the remaining operations.

- The Dictionary Editor allows you to easily set up the proper database structure by defining *files, fields, and relations*.
- The Dictionary Editor allows you to easily plan both *primary and foreign keys* for your database, as per the relational model's integrity rules.
- The Dictionary Editor allows you to easily implement *data integrity 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.
- Additionally, the Dictionary Editor supports preconstruction of "Views." The View creates a virtual file, automatically handling any necessary "Joins" and "Projects."

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.

❏ *To open the* **Dictionary Editor** *to create a new dictionary file*:

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

❏ *To open the* **Dictionary Editor** *to edit an existing dictionary file*:

1. Choose **File ➤ Open,** then select the **Dictionary** tab.
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 and press the **Open** button.

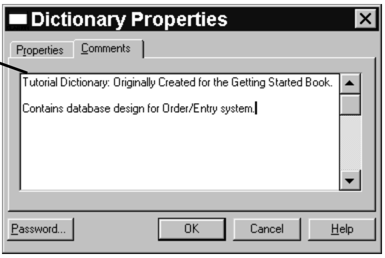
You may use the same data dictionary for more than one application. An application, however, can only have one data dictionary.

Tip: Clarion for Windows will automatically read and convert 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 for Windows, some relationships may not be complete due to stricter error checking.

❏ *To add a text 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.

The Dictionary Properties dialog allows you to store a text description of the data dictionary.



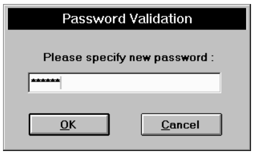
Tutorial Dictionary: Originally Created for the Getting Started Book.

Contains database design for Order/Entry system.

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 a long absence.

❏ *To add a password to the data dictionary*:

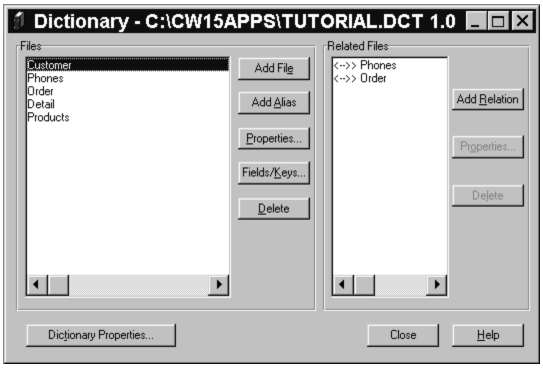
1. Press the **Password** button.
2. When the **Password Validation** dialog appears, type a password in the space provided, and press the **OK** button.
3. When the **Password Verification** dialog appears, type the *same* password, and press the **OK** button.



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.



❏ *To add a file to the files list*, press the **Add File** button. This displays the **New File Properties** dialog.

❏ *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.

❏ *To add a view to the list*, choose **Edit ➤ Add View.** This displays the **New View** dialog. See the *Adding a View* section, below.

A Word About Quick Load

When you press the **Add File** button you will be given the option of using Quick Load to add your file to the data dictionary. Quick Load allows you to specify only the most basic information about your file and its fields, while Quick Load supplies all other required attributes by default. Quick Load is especially useful for quickly producing a working application that can then 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 that you can take advantage of the many file and field attributes supported by Clarion’s data dictionary from the very beginning. For example, the data dictionary supports data entry validation, but validation defaults to none if Quick Load is used. The following section assumes you are not using Quick Load to add the file.

See *Adding a File with Quick Load* in Chapter 3 of *Getting Started*.

New File Properties

Define a new file for the database with the **New File Properties** dialog. This dialog allows you to add a new data file to the list and choose its file driver.

Once the file appears on the list, you may declare fields, keys, set relationships, and other properties for the data file. Using the data from this dialog, the Application Generator will write the FILE structure declaration, plus file I/O routines as required by your application.

General

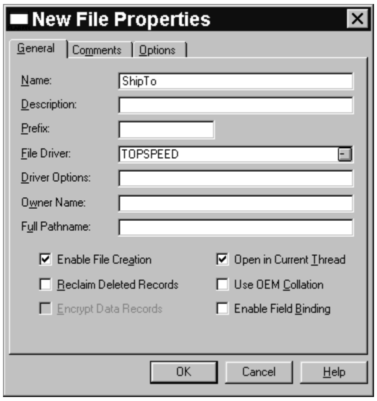
Name Type a data file name, as you wish to refer to it in your code. This serves as the label for the Clarion FILE structure. Specify a valid Clarion label (see the *Language Reference*)—Clarion will automatically truncate the name if necessary. You may also specify a completely different name for the DOS file—see **Full Pathname**, below.

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

1. Prefix the file name with "!Glo:." For example, !Glo:Customer.
2. Then create a variable in your program with the same name, less the exclamation point: Glo:Customer.
3. Embed the following code in the CheckOpen Setup embed point:

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

The file can now be opened without mapping a drive letter to it.



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

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

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 similar variable names occurring in different file 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. Prefix is no longer required in Clarion 1.5. See *Field Qualification Syntax* in the *Language Reference* for details.
File Driver	Specify the file type: TopSpeed, Clarion, Btrieve, ASCII, etc. When using the Application Generator, Clarion for Windows automatically links in the correct database file driver library. See the *Database Driv*ers appendix for a discussion of the relative advantages of each driver. Remember that file systems may vary in their support of some of the attributes which you add to the FILE structure in this dialog box.
Driver Options	Optionally type a string for an additional driver attribute. This conveys additional instructions to the file driver and corresponds to the second parameter for the DRIVER attribute, also known as a “driver string.” The *Database Driv*ers appendix contains additional information.
Owner Name	Optionally type a string containing the password for access to the file. This is dependent on the file system. This adds the OWNER attribute to the FILE statement. For most file systems, you must also check the **Encrypt** box (below).

Encrypting the file means that only your application will be able to read the file. It does not mean that it automatically prompts the end user for a password. The end user, however, may not access the data with any other file viewer.

When using the ODBC driver, type the data source name, user ID, and password, separated by commas, in this field. See the *ODBC append*ix for more information.

Full Pathname

Optionally, type a fully qualified file name 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 currently 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 the *Database Driv*ers appendix for more information.

Note: The first eight characters of the table name must be unique for all the tables stored in this physical file. Example: use \!EmpPayroll and \!EmpMaster, not \!EmployeePayroll and \!EmployeeMaster.

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 the *ODBC append*ix for more information.

Tip: To specify a variable name instead of the actual file name, place the variable name in this field following an exclamation point (!). 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.

Reclaim Deleted Records

This option is dependent upon 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

Optionally turn on file encryption. You must also specify an **Owner Name** (see above). This adds the ENCRYPT attribute to the FILE statement.

Open in Current Thread

Optionally specify that each execution thread that uses this file, allocates memory for its own separate record buffer. This is typically for use in multiple document (MDI) applications, and improves file handling. The Clarion default templates automatically add the THREAD attribute to each FILE structure.

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.

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 variables in the RECORD structure are available for use in dynamic expressions (using BIND and EVALUATE) at run time. The compiler will allocate memory to hold the full Prefix:Name for each variable. It would otherwise use its own internal reference for each variable. Therefore the BINDABLE attribute increases the amount of memory necessary for the application.

Comments

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

Options

Do Not Auto-Populate This File

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

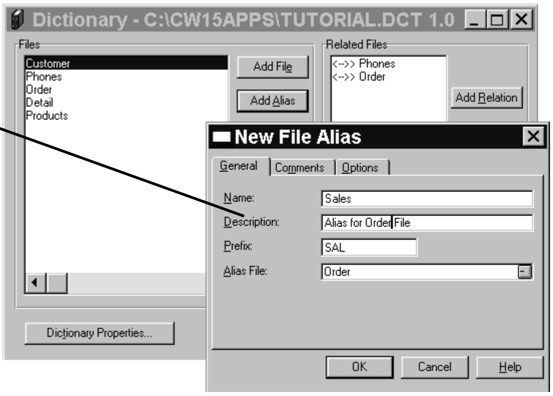
User Options The text typed into this field is made available to any Utility Templates that process this file. The Utility Templates determine the proper syntax for these user options.

To *modify* the file properties at any time, highlight the file name on the **Dictionary** dialog list, then press the **Properties** button.

ADDING A FILE ALIAS TO THE DICTIONARY

An alias creates a second reference for a file without duplicating the file on disk. You can add an alias for a file only if it's already on the Dictionary list. In the Dictionary dialog, press the **Add Alias** button to display the *New File Alias* dialog.

Setting up a file alias, which can then be used like a normal file.



Name: Sales
Description: Alias for Order File
Prefix: SAL
Alias File: Order

A file alias provides several advantages, at the cost of some system overhead:

- *Allows you to set multiple relationships between files.*

  Strict relational database theoreticians state a file may only have a single relational link to another file at a time. Aliases allow you to "legally" work around this limitation.

- *Allows a second file buffer for the same file.*

  You could use this for a second file browse, as well as entry forms and other items for each. This is particularly useful for a Multiple Document Interface (MDI) application.

- *Uses additional memory and resources.*

  Any file driver that uses external key files requires additional file handles for each alias. For example, a file with three external keys and three aliases requires sixteen file handles: one each for the "first" data file and its three keys, and an additional four for each of the aliases. When using aliases, we recommend choosing a file driver that stores keys internally, such as TopSpeed or Btrieve.

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

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

General

Name	Type an alias “name”, as you wish to refer to it in your code. The name must be a valid Clarion label.
Description	Enter a string description for the alias. Clarion displays the descriptions in dialogs such as the **Dictionary** dialog.
Prefix	By default, Clarion will use the first three letters of the Name for the prefix. Optionally specify up to 14 characters of your choice.
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 using the **Add File** command in the **Dictionary** dialog.

Comments

Comments	Select the **Comments** tab to type a separate file 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 procedures or Update procedures for this file alias.
User Options	The text typed into this field is made available to any Utility Templates that process this file. The Utility Templates determine the proper syntax for these user options.

To modify the alias properties at a later time, highlight the alias name on the **Dictionary** dialog list, then press the **Properties** button.

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.

ADDING A VIEW TO THE DICTIONARY

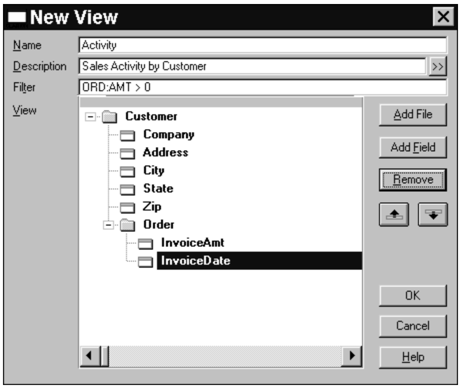
A VIEW is a virtual file constructed from selected fields in multiple files.

Creating a view provides a (potentially) major advantage in a Client-Server environment because the Server has the ability to do much of the work. The Server processes the overhead of the relational “Join” and “Project” operations which would otherwise tie up the local machine.

When working with views, be sure to include all the fields you need to work with in the VIEW, and don’t try to access any fields not in the VIEW. This is necessary because only the data elements specified in the VIEW—not the RECORD structures from the component files—are updated. This means accessing a field in the RECORD structure which is not also defined in the VIEW structure returns an undefined value.

The VIEW structure has no prefix. Access its fields by using the prefixes for the original RECORD structures defining the fields. This is transparent when you use the Application Generator. The **File Schematic Definition** automatically adds the proper prefix so that the generated code is correct.

To add a view to the files list, choose **Edit ➤ Add View**. Fill in the **New View** dialog.



Name: Activity
Description: Sales Activity by Customer
Filter: ORD:AMT > 0

- Customer
  - Company
  - Address
  - City
  - State
  - Zip
  - Order
    - InvoiceAmt
    - InvoiceDate

Name	Type a view name, as you wish to refer to it in your code. The name must be a valid Clarion label.
Description	Type a string description for the view. Clarion displays the descriptions in dialogs such as the **Dictionary** dialog.

	Press the >> button to type a separate description (up to 1000 characters) in a larger text box.
Filter	Type an expression (such as PRE:Field1 > 1) to limit the contents of the view to only those records matching the filter expression. The filter is independent of any keys defined for the files referenced by the VIEW structure. In a Client/Server environment the filter may not adversely affect performance; in any other environment, it may slow down file operations.
Add File	Allows you to add a file to the View.
Add Field	Allows you to add a field from the currently selected file to the view.
Remove	Removes the currently selected file or field from the view.

❏ To add files and fields to the VIEW structure:

1. Press the **Add File** button.
2. In the **Select Primary File** dialog, choose a file and press the **OK** button.

   The file appears in the view list.
3. Back in the **New View** dialog, press the **Add Field** button.
4. In the **Add Field** dialog, CLICK on the fields you wish to include in the view, then press the **OK** button.

   The fields appear directly below the file in a tree diagram.
5. Repeat steps *1* through *4* for any additional files and fields you wish to add to the view.

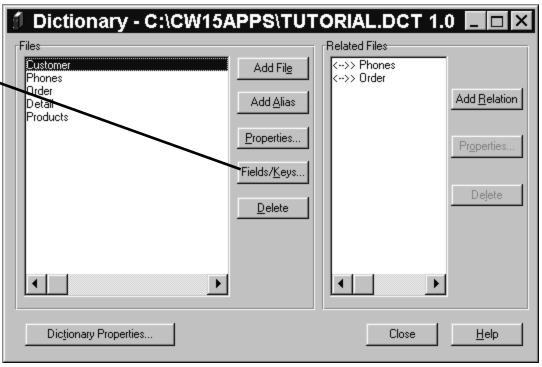
   Only files already related to the current file may be added to the file list below it.
6. Press the **OK** button to close the **New View** dialog.

To modify the view properties at a later time, highlight the view name on the **Dictionary** dialog list, then press the **Properties** button.

ADDING OR MODIFYING FIELDS

Once you define a file, you may define its fields. Highlight the file in the **Dictionary** dialog window and press the **Fields/Keys** button. 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.

To add or edit fields and keys, press the Fields/Keys button.



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 and press the **Properties** button.

❏ *To delete an existing field,* select the field name and press the **Delete** button.

❏ *To move the selected field within the Fields list*, press the ↑ and ↓ buttons. This reorders the field labels within the FILE structure.

When you add or modify a field, the **New Field Properties** or **Edit Field Properties** dialogs appear when you press the **Insert** or **Properties** buttons, respectively.

Defining Field Properties

The **New Field Properties** and **Edit Field Properties** dialogs allow you to set field related options and attributes.

These dialogs are *identical* to the dialogs for defining and editing memory variables. All the Clarion language attributes applicable to a field in a file also apply to memory variables. There are a few additional attributes that are *only* applicable to memory variables.

Clarion's development environment uses the same dialog for defining fields and memory variables, so that you don't have to learn two separate dialogs. Controls which refer to attributes applicable only to memory variables are disabled when defining a field in a file.

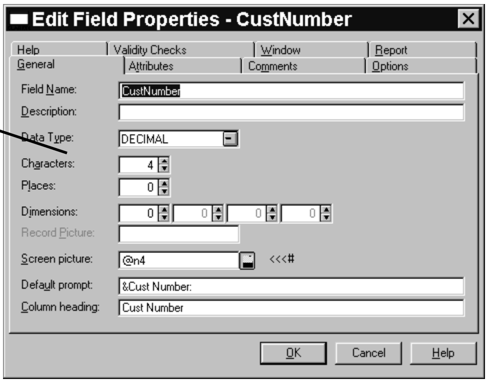
The Dictionary Editor allows you to quickly add the fields one after another. Each time you complete and close the **New Field Properties** dialog for one field, another blank **New Field Properties** dialog appears, ready for the next field. After completing the last field, press **Cancel** when the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

General

❏ *To name the field*, type a valid Clarion label in the **Field Name** field. Valid field names may vary slightly according to the file driver.

❏ *To add a text description*, type it in the **Description** field. The description appears in the list in the **Field Properties** dialog. Also, see **Comments** below.

The Field Properties dialog provides centralized access to a wide range of field attributes and properties.



Field Name: CustNumber
Description:
Data Type: DECIMAL
Characters: 4
Places: 0
Dimensions: 0 0 0 0
Record Picture:
Screen picture: @n4 <<<#
Default prompt: &Cust Number:
Column heading: Cust Number

❏ *To assign a field data type*, choose one from the **Data Type** drop down list. Clarion supports the following field types, which specify how the data will be stored on disk by the file driver, and accessed in memory by the application. The types available vary according to the selected file driver.

STRING	A fixed length character string, usually up to 65,520 characters in length, depending on the file system.
MEMO	A variable length text field, up to 65,520 characters in length. To specify that a memo field may hold binary data, check the **Binary** box. This is dependent on the file driver. Refer

	to the *Database Driv*ers appendix for more information on how each driver stores memos.
PICTURE	Provides a “storage picture” for a String field. Picture is not a separate data type, but declares the field as a STRING of length and format PICTURE. Fill in the Record Picture field with the Storage Picture Token. Refer to 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	Can contain 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.

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 Can contain 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.

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.

Tip: Use the ↑ and ↓ buttons to move fields into and out of the Group.

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.

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

❏ *To assign a field length*, specify a number in the **Characters** field. If the field holds decimal places, specify a number in the **Places** field.

❏ *To declare the field as an array*, and to specify the array dimensions, type them in the **Dimensions** fields. You can specify up to four dimension sizes. Total array size may not exceed 65,520 bytes.

❏ *To specify the record storage picture for a field*, type a picture token in the **Record Picture** field.

❑ *To specify a screen picture*, type a picture token in the **Screen Picture** field. When the Application Generator creates window and report controls for the field, this serves as the default display picture for the control.

❑ *To lock the screen picture*, which specifies that it may not be changed even if the field type is changed, press the “Lock” icon next to the **Screen Picture** field.

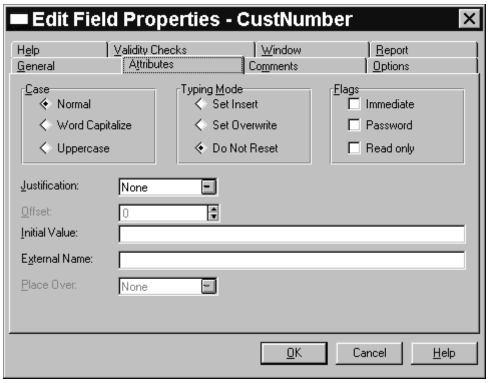
❑ *To specify the default prompt string*, type it in the **Default Prompt** field. The Application Generator places this text in the PROMPT control associated with this field when it is populated onto a window.

❑ *To specify the default column title*, type it in the **Column Heading** field. The Application Generator uses this for reports.

Attributes

❑ *To specify the case attribute for controls referencing the field*, choose from the **Normal**, **Word Capitals** or **Uppercase** radio buttons, in the **Case** group box. The Application Generator adds the CAP or UPR attribute to the field’s entry control.

❑ *To specify the default typing mode attribute for controls referencing the field*, choose from the **Set Insert**, **Set Overwrite** or **Do Not Reset** radio buttons in the **Typing Mode** group box. The Application Generator adds the INS or OVR attribute to the field’s entry control.


Edit Field Properties - CustNumber
Help | Validity Checks | Window | Report
General | Attributes | Comments | Options
OK | Cancel | Help

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

❑ *To specify immediate event notification for controls referencing the field*, check the **Immediate** box in the **Flags** group box. The Application Generator adds the IMM attribute to the field’s entry control.

❑ *To specify the data non-display attribute for controls referencing the field*, check the **Password** box. The Application Generator adds the

PASSWORD attribute to the field's entry control. When an end user types in an entry control referencing this field, the characters typed appear on screen as asterisks.

❏ *To specify the display only attribute for controls referencing the field*, check the **Read only** box. The Application Generator adds the READONLY attribute to the field's entry control.

❏ *To specify justification for controls referencing the field*, select from the **Justification** 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 (which hides any digits to the right of the decimal point).

❏ *To specify an offset to the justification*, specify a number in the **Offset** field. 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.

❏ *To specify a default value for the field*, type it in the **Initial Value** field.

Tip: You can type a function or a variable in the Initial Value field in a file. For example, in a date field, you can add the TODAY() function to make the initial value today's date. Functions and variables are not valid initial values for memory variables, i.e. Global Data, Local Data, and Module Data.

❏ *To specify an external name for the field*, type it in the **External Name** field. This covers cases where the field label within the program is different than the name of the field in the data file; for example, you may be accessing a field through an ODBC connection to a database which contains a field name that is not a valid Clarion label. Place the name of the field as it exists in the data file here. This creates the field's NAME attribute.

❏ *To declare the field as an overlay*, select another field name from the **Place Over** drop down list. This allows the current field to redefine the other field's location in memory. This adds the OVER attribute to the field's declaration.

❏ The **Storage Class** drop down list is enabled only when defining memory variables. This selection sets the EXTERNAL, STATIC, and AUTO attributes for memory variables, which determines memory allocation for those variables. See the *Language Reference* for more information.

Comments

❏ *To add a comment or text description*, select the **Comments** tab to type up to 1000 characters.

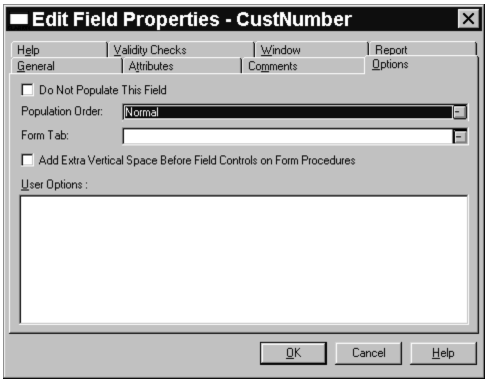
Options

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

❏ *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.

❏ *To tell Clarion's Wizards which property sheet tab will contain this field, use the* **Form Tab** drop down list.

❏ *To cause Clarion's Wizards to add extra vertical space before this field on form procedures,* check the **Add Extra Vertical Space Before Field Controls on Form Procedures** box.


Edit Field Properties - CustNumber

Do Not Populate This Field
Population Order: Normal
Form Tab:
Add Extra Vertical Space Before Field Controls on Form Procedures
User Options :

❏ *To specify User Options*, type free form text into the **User Options** field. The text typed into this field is made available to any Utility Templates that process this file. The individual Utility Templates determine the proper syntax for these user options.

Help

❏ *To specify a Help ID for controls referencing the field*, specify a help topic in the **Help ID** field. The Application Generator adds the HLP attribute to the field's entry control.

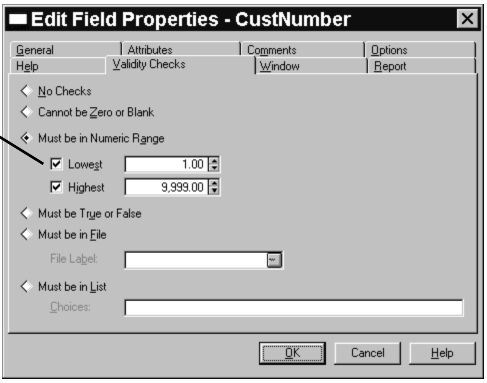
❏ *To specify a status bar message for controls referencing the field,* type the message in the **Message** field. When the control referencing the field has focus, the text appears on the status bar, provided the application in which the control appears has one. The Application Generator adds the MSG attribute to the field's entry control.

❏ *To specify a popup message for controls referencing the field,* type the message in the **Tool Tip** field. When the control referencing the mouse is idle over the field, the text appears immediately below the mouse in a popup box. The Application Generator adds the TIP attribute to the field's entry control.

Validity Checks

To check an entry when the user completes the field, select the **Validity Checks** tab in the **Field Properties** dialog, then choose a validation option by clicking on one of the radio buttons.

Using the Validity Checks tab to specify a limited range of values for a field.



Must be in Numeric Range
☑ Lowest 1.00
☑ Highest 9,999.00

The Application Generator uses this information when creating and maintaining controls. When the user tabs off the field and shifts focus to another control, or presses OK on the data entry dialog, the application will sound a warning beep and set 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:

❏ *To disable validity checking*, choose **No Checks**. This is the default.

❏ *To require a user entry without specifying any other criteria*, choose **Cannot be Zero or Blank**. The Application Generator adds the REQ attribute.

  The REQ attribute behaves differently for tabbed dialogs than for single page dialogs. Because the user has the option of never even 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; add the REQ attribute to the tab and to the required entry fields; or

  Make a (see also)"Wizard"; or

  Embed code at the beginning or end of the procedure that selects all tabs with required fields; add the REQ attribute to the required entry fields and to their parent tabs.

❏ *To specify the entry fall between two numeric values*, choose **Must be in Numeric Range**. Then enter the two values in the **Lowest** and **Highest** fields.

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

❏ *To specify a logical binary entry (yes/no, true/false, off/on)*, choose **Must be True or False**. This feature works best with the BYTE data type and a check box screen control.

❏ *To specify the value match a field in an external file*, choose **Must be in File**. Choices will appear in the **File Label** drop down list *only* if you previously related another file or files. See *Adding or Modifying Relations* later in this chapter.

❏ *To specify the value match an entry in a list,* choose **Must be in List**. Then type the choices in the **Choices** field, in the format “Choice1|Choice2|Choice3.” Separate the choices with a pipe (|) character (usually SHIFT+\).

Tip: If you plan to allow the end user to choose a limited number of choices from a list box, drop down list, or radio buttons, type the choices and separate them with a pipe symbol, or vertical bar character (|).

Window

Use the **Window** tab in the **Field Properties** dialog to specify how a particular field will be presented 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 will be 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*.

Customarily, field data have been presented to the end user with two controls: a prompt and a simple entry box, called an entry control. The *prompt* is simply the label or heading on the computer screen or report that identifies the data item. The *entry control* is the area on the screen or report where the data item is represented (printed or typed). For example, in the following illustration, “Date:” is the prompt, the underscore is the entry control, and “Sep 1, 1995” is the data item.

Date: <u>Sep 1, 1995</u>

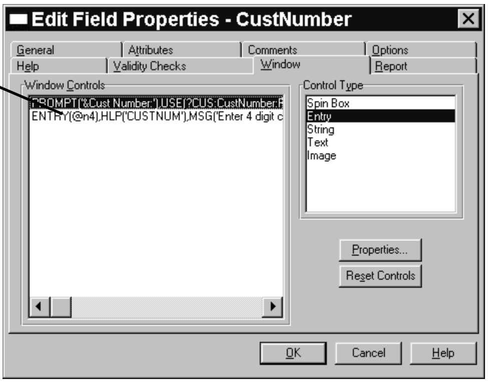
In the windows environment, a data item may be presented to the end user in a variety of ways. Most of these methods still use a simple prompt, but the entry control varies widely. Entry controls include entry boxes, list boxes, drop down lists, check boxes, radio buttons, spin boxes, etc.

To customize the default characteristics for prompts and entry controls for a field, select the **Window** tab, in the **Field Properties** dialog.

Tip: **By choosing the properties for a control at this time, you can save time later. Every application you generate from the dictionary, and every procedure in the application will automatically format 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.**

Select either the prompt or entry field from the **Window Controls** list, then press the **Properties** button. The *prompt* is the label which appears next to the control on your application window or report. The *entry* field is the control which actually accepts user input.

Preformatting an ENTRY control.



```
PROMPT('&Cust Number:'),USE(?CUS:CustNumber:P
ENTRY(@n4),HLP('CUSTNUM'),MSG('Enter 4 digit c
```

❑ *To customize the default prompt*, select the PROMPT item in the **Window Controls** list box, then press the **Properties** button (or simply DOUBLE-CLICK the prompt item) to display the **Prompt Properties** dialog. See the *Setting Control Properties* chapter for details.

❑ *To customize the default entry control*, select the ENTRY, LIST, or other entry control item in the **Window Controls** list box, then press the **Properties** button (or simply DOUBLE-CLICK the item) to display the **Entry Properties** dialog. See the *Setting Control Properties* chapter for details.

❑ *To change the type of entry control*, first select the ENTRY item you want to change in the **Window Controls** list box, then in the **Control Type** list, select a new control type.

❑ *To change prompt and entry controls back to their default values*, press the **Reset Controls** button.

Tip: If you specified Must be True or False on the Validity Checks tab for a numeric value, the Screen Control will default to a check box. You may edit the check box properties.

Report

To customize the default characteristics for report controls for a field, select the **Report** tab in the **Field Properties** dialog.

❑ Highlight the control from the **Report Controls** list by clicking on it, then press the **Properties** button. The **String Properties** dialog appears. See the *Using the Report Formatter* chapter for details.

❑ *To change the default report string control to an option control*, check the **Option Control** box. You must have previously defined the **Must be in List** choices on the **Validity Checks** tab.

Tip: If you specified Must be True or False in the Validity Checks tab for a numeric value, the Report Control will default to a check box. You may edit the check box properties.

ADDING OR MODIFYING KEYS

Add and edit 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 in the dialog boxes.

Keys and indexes specify sort orders for a single data file. A key may reside within the file itself, or as an external file, depending on the file system.

Keys are automatically updated whenever records are added, changed, or deleted. Indexes are not. The *Database Driv*ers appendix provides more information regarding how each file driver supports keys or indexes.

Indexes usually exist as external files. Remember that a separate DOS file handle is necessary for each external key or index file. *Index files are not updated automatically*. The BUILD statement updates an index.

A runtime index allows you to declare an index file without specifying the key component field(s) in the Data Dictionary. The application must define the key component field(s) at run time, as the second parameter of the BUILD statement. The application may rebuild the same index file at a later time, specifying different key component field(s) for the index key.

The basic steps for creating a key are:

1. Select a file from the list on the **Files** side of the **Dictionary** dialog and press the **Field/Keys** button.
2. In the **Field/Keys 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.

   The **New Key Properties** dialog appears.

4. Type a valid Clarion label in the **Key Name** field.
5. Optionally type a **Description**. This displays in various dialog boxes, including the **File Definition** dialog.
6. Select the **Attributes** tab and check all boxes that are appropriate for the key.
7. Optionally type a valid DOS file name in the **External Name** field, if the file system needs one.

   Clarion automatically adds the proper extension.
8. Select the **Fields** tab, then press the **Insert** button.

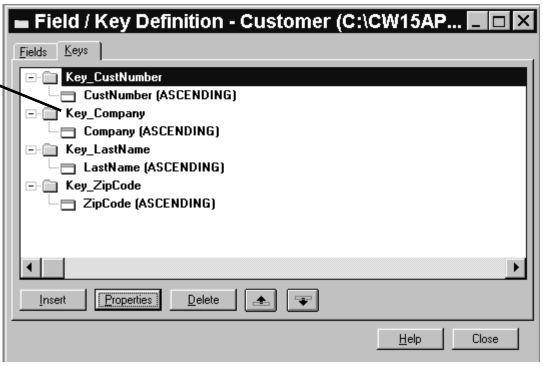
   The **Insert Key Component** list appears.
9. DOUBLE-CLICK a field in the list; this transfers its name to the **Fields** tab, which indicates the field will be part of the new key.
10. Press **OK** to close the **New Key Properties** dialog.

    The **New Key Properties** dialog appears again, ready to accept additional keys.
11. Repeat steps 4 through 10 to create additional keys for this file.
12. When you are finished adding keys, press **Cancel** to close the **New Key Properties** dialog and return to the **Field/Keys Definition** dialog.

At the end of the process, your keys appear on the **Keys** tab, with their field components arranged in order, one above the other in a tree diagram.

To modify a key, select the key and press the **Properties** button in the **Field/Key Definition** dialog. The **Edit Key Properties** dialog appears. 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, below, describes the options in this dialog.

The Field/Keys Definition dialog displays the fields and keys for a single file.



Key_CustNumber
CustNumber (ASCENDING)
Key_Company
Company (ASCENDING)
Key_LastName
LastName (ASCENDING)
Key_ZipCode
ZipCode (ASCENDING)

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

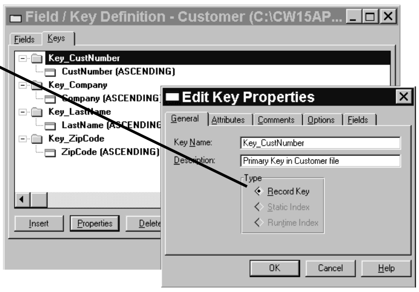
Key Name *To specify a Clarion label for the key*, type a valid Clarion label in this field.

Tip: Remember that 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 *To place a text description for the key* in the Data Dictionary, type it 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.

Type *To specify a record key, a static index, or runtime index*, choose one of the radio buttons in the **Type** group. Remember, record keys are *automatically* updated whenever records are added, changed, or deleted. Indexes are *not* automatically updated, but require a BUILD statement. The **Static Index** and **Run Time Index** options are *disabled* when the **Require Unique Value** check box is marked on the **Attributes** tab, because indexes always allow duplicates.

Defining a primary key.



Key_CustNumber
CustNumber (ASCENDING)
Key_Company
Company (ASCENDING)
Key_LastName
LastName (ASCENDING)
Key_ZipCode
ZipCode (ASCENDING)

Key Name: Key_CustNumber
Description: Primary Key in Customer file
Type
[✓ Record Key]
Static Index
Runtime Index

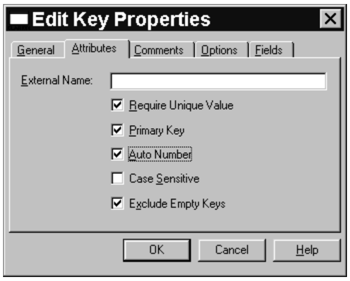
Attributes

External Name *To optionally specify a DOS file name for an external key*, type a valid DOS file name in this field. Clarion automatically adds the proper extension. The Application Generator adds the NAME attribute to the KEY statement. Some file systems require an external name. See the *Database Drivers* appendix for more information.

Require Unique Value
To disallow multiple records with duplicate values in their keys, 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 from the KEY statement. This may be required for certain file drivers. See the *Database Drivers* appendix for more information.

Auto Number *To specify the Application Generator should create code to manage record sequence numbers*, check this box.


Edit Key Properties

General | Attributes | Comments | Options | Fields

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, all capital letters will 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 statement.

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

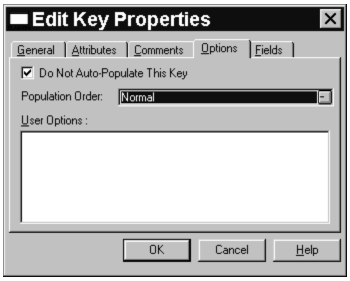
Comments

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.



☑ Do Not Auto-Populate This Key
Population Order: Normal
User Options :

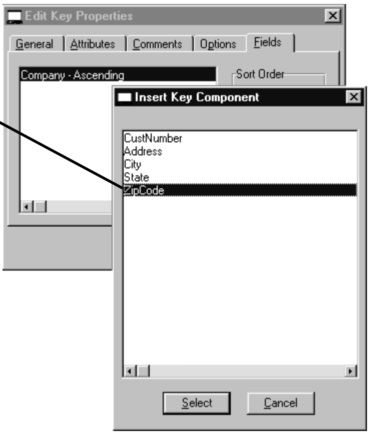
User Options *To pass information to any Utility Templates that process this key,* type the information here. The text typed into this field is made available to any Utility Templates that process this key. The Utility Templates determine the proper syntax for these user options.

Fields

Specify the components of the keys (the sort field or fields) using the **Fields** tab of the **Edit Key Properties** dialog. You may specify more than one field for a key. You may mix data types when defining a key on multiple fields. You may also specify different orders—one field ascending, one field descending—when defining a key on multiple fields, however, mixing sort order is file driver dependent. See the *Database Drivers* appendix 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 place its name in the **Key Fields List**.

Specifying a key component.



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 RELATIONS

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.

The basic steps in setting up a relationship are:

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

   The **New Relationship Properties** dialog appears.
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.

   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 will be 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 should be exactly one (i.e., unique) customer number 629 in our customer file. Otherwise, we could easily confuse information belonging to Katy (customer 629) with information belonging to a different customer with the same customer number. So customer number is the **Primary Key** for the Customer file.

A **Foreign Key** need not be unique, but it should match the primary key in another file. In our example, there is only one customer number 629 in our Customer file. However, because Katy (customer 629) has submitted several orders, customer number 629 appears several times in the Order file. So customer number is also a key to this file relationship, but it is the **Foreign Key.**

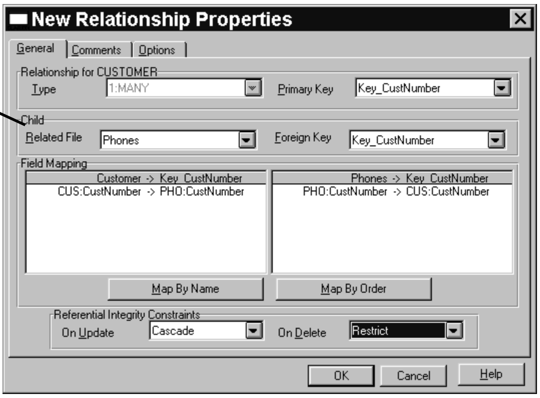
6. Select the **Primary Key** or **Foreign Key** for the related file, if applicable, from the drop down list immediately below the first drop down list.

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 made up of multiple components, or fields, are handled correctly.

Defining a one to many relationship, including key field mapping and referential integrity.



Relationship for CUSTOMER
Type: 1:MANY
Primary Key: Key_CustNumber

Child
Related File: Phones
Foreign Key: Key_CustNumber

Field Mapping

Customer -> Key_CustNumber
CUS:CustNumber -> PHO:CustNumber

Phones -> Key_CustNumber
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 section below for more information on Referential Integrity Constraints.

9. Press the **OK** button.

The new relationship appears in the **Dictionary** dialog.

Setting Referential Integrity Constraints

By setting referential integrity constraints in the data dictionary, you can tell the Application Generator how to set up executable code for handling linked field updates and deletions when working with related data files.

Referential Integrity requires that a foreign key must always have a match in the primary key. This raises potential problems when the end user wishes to change or delete the primary key record.

The **New Relationship Properties** dialog allows you to specify how the generated code will handle the situations where 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.

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

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

MANAGING YOUR DICTIONARY

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

- You can copy and paste file and field definitions from one dictionary file to another.

- The Dictionary Editor offers version management, which allows you to document your changes when you make significant revisions. It also allows for rollback abilities, so that you can “undo” your revisions.
- The Dictionary Editor offers custom setup options which, for example, allow you to define the default file driver.

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 dictionary file.
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/Keys Definition** dialogs open, rather than the **Dictionary** dialog. The limitations are that the target file must support the field type being copied.

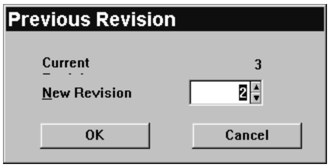
Tip: You can copy a Data Dictionary item, such as a file or a field, into the clipboard, then paste it into the Text Editor (and vice versa)!

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, manually, 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.



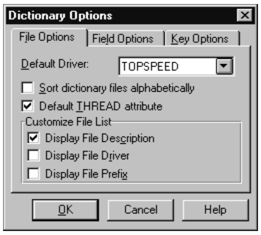
Dictionary Editor Setup Options

You can customize some of the default dictionary settings in the **Dictionary Options** dialog. To access the dialog, choose **Setup ➤ Dictionary Options**.

File Options

❏ *To select the default database driver for new dictionaries,* choose from the **Default Driver** drop down list.

❏ *To see the* **Dictionary** *dialog's* **Files** *list in alphabetical order*, check the **Sort dictionary files alphabetically** box.



Default Driver: TOPSPEED
☐ Sort dictionary files alphabetically
☑ Default THREAD attribute
Customize File List
☑ Display File Description
☐ Display File Driver
☐ Display File Prefix

❏ *To select the* THREAD *attribute (setting aside a separate* RECORD *buffer for each procedure) as the default for new file definitions,* check the **Default THREAD Attribute** box.

❏ *To see file descriptions in the* **Dictionary** *dialog's* **Files** *lis*t, check the **Display File Description** box.

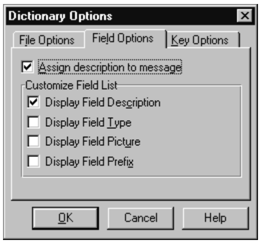
❏ *To see file drivers in the* **Dictionary** *dialog's* **Files** *list*, check the **Display File Driver** box.

❏ *To see the file prefix in the* **Dictionary** *dialog's* **Files** *list*, check the **Display File Prefix** box.

Field Options

❏ *To specify that the field descriptions you type when defining a field should also serve as the text for the* **Message** *field* (status bar message), check the **Assign Description to Message** box.

❏ *To see the field description in the* **Field/Key Definition** *dialog*, check the **Display Field Description** box.



Dictionary Options — Field Options

☑ Assign description to message

Customize Field List
- ☑ Display Field Description
- ☐ Display Field Type
- ☐ Display Field Picture
- ☐ Display Field Prefix

❏ *To see the field type in the* **Field/Key Definition** *dialog*, check the **Display Field Type** box.

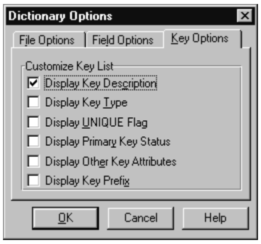
❏ *To see the field picture in the* **Field/Key Definition** *dialog*, check the **Display Field Picture** box.

❏ *To see the field prefix in the* **Field/Key Definition** *dialog*, check the **Display Field Prefix** box.

Key Options

❏ *To see the key description in the* **Field/Key Definition** *dialog*, check the **Display Key Description** box.

❏ *To see the key type in the* **Field/Key Definition** *dialog*, check the **Display Key Type** box.



Dictionary Options — Key Options

Customize Key List
- ☑ Display Key Description
- ☐ Display Key Type
- ☐ Display UNIQUE Flag
- ☐ Display Primary Key Status
- ☐ Display Other Key Attributes
- ☐ Display Key Prefix

❏ *To see the unique flag in the* **Field/Key Definition** *dialog*, check the **Display UNIQUE Flag** box.

❏ *To see the primary key status in the* **Field/Key Definition** *dialog*, check the **Display Primary Key Status** box.

❏ *To see other key attributes in the* **Field/Key Definition** *dialog*, check the **Display Other Key Attributes** box.

❏ *To see the key prefix in the* **Field/Key Definition** *dialog*, check the **Display Key Prefix** box.

USING THE APPLICATION GENERATOR

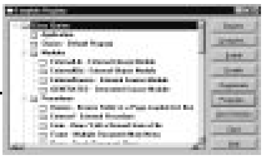
Contents

The Application Generator writes source code for you, based on the procedure templates you pick, and your customization of the procedures. The Application Generator calls other development environment tools as needed.

You start by defining the .APP file, which stores the procedures you select and your customizations.

The Application Tree lists your procedures. Those you have not yet “filled in” are marked as “To Do.”

The Procedure Properties dialog acts as a “hub” from which you call other Development Environment tools to customize the procedure.



You can customize, add or delete templates; you can also regenerate .TPL files using the Template Registry.

When you use the Application Generator, you create procedures for the major tasks you want your application to do, you describe how the application accomplishes the tasks, and how its 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.

This chapter describes how to complete all the dialogs the Application Generator needs to generate source code:

- How to begin a new application by creating an .APP file.
- How to set global application properties.
- How to add procedures to the application.
- How to fully customize your procedures.
- How to set application options.
- How to maintain your templates and the REGISTRY.TRF file.
- How to import and export procedures.

CREATING THE .APP FILE

The first step in creating a new application (after creating a Data Dictionary) is to create an .APP file. The .APP file holds the procedures, data, and other properties you define for your application, that is, 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 .APP file, Clarion for Windows will use the directory in which the .APP 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 and press **OK** (or use the DOS prompt, and the MkDir command).
2. Start Clarion and choose **File ➤ New**.

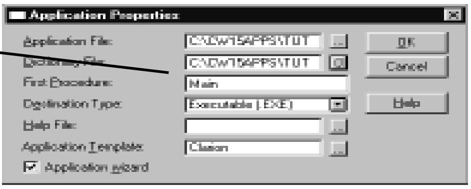
   The **New** dialog appears.
3. On the **Application** tab (CLICK on it) uncheck the **Quick Start Wizard** box by CLICKING on it.

   See Chapter 3 in the *Getting Started* manual for more information on using **Quick Start**.

4. Use the **Drives** drop box and the **Folders** box to navigate to your application directory, then press the **Create** button.

   The **Application Properties** dialog appears. This dialog allows you to define the essential files for the application.

Defining a new .APP file called TUTORIAL.APP, using a data dictionary called TUTORIAL.DCT.



Application File: C:\CW15\APPS\TUT
Dictionary File: C:\CW15\APPS\TUT
First Procedure: Main
Destination Type: Executable (.EXE)
Help File:
Application Template: Clarion
☑ Application wizard

5. Type a name for the .APP file in the **Application File** field.

   Type a legal DOS file name (must also be a valid Clarion label, see the *Language Reference* for more information). Clarion automatically adds the .APP extension. Clarion will use the path you defined on the **New** dialog. Alternatively, you can press the ellipsis (**...**) button to define a path 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. The **Select Dictionary** dialog is a standard **Open File** dialog.

Tip: The Application Generator will *not* require a data dictionary to generate an application if you *uncheck* the Require a dictionary box in the Application Options dialog, described later in this chapter.

7. Optionally rename the first procedure from MAIN to another name of your choice.

   You can do so by typing another procedure name in the **First Procedure** field.

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

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

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.

   The Application Generator does *not* require that the .HLP file exist at this point. You can also name help topics that do not exist at this point.

However, 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 various controls and dialogs. There are many third party products that help you do this.

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

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

11. Uncheck the **Use Application Wizard** box by CLICKING on it.

    Checking this box will cause the Application Generator to use your dictionary to create an entire working application. In this chapter, we will build an application without using the **Application Wizard**.

12. Press the **OK** button.

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

OVERVIEW: CREATING YOUR APPLICATION

Once the .APP file exists, you design your application through a series of dialogs. When you create your application's menus and toolbars, they will call procedures that you name. The Application Generator adds these "To Do" procedures to the application tree. You "fill-in" their functionality by picking from a set of procedure templates. Use the Window and Report Formatters to design 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 Tutorial in the *Getting Started* manual provides a more detailed description.

❏ Define the Main procedure type.

❏ Customize the application's menu, adding menu commands and the procedures they execute.

❏ Add functionality to your procedures.

  ❏ Choose the files the procedures use.

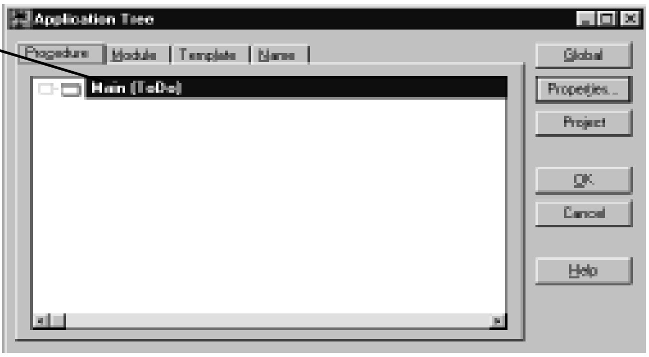
  ❏ Add local variables.

  ❏ Use the **Window Formatter** to design your windows.

❏ Generate the source code and make the application.

❏ Test the application by pressing the **Run** button in the **Compile Results** dialog.

The Application Tree dialog for a new application. It contains one MAIN procedure, whose functionality is still "ToDo."

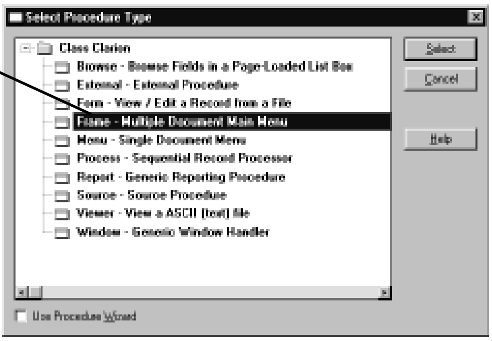


❏ Define the Main procedure type.

Press the **Properties** button to access the **Select Procedure Type** dialog. This lists the procedure types available in the currently registered templates.

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

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


Select Procedure Type
Class Clarion
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
Viewer - View a ASCII (text) file
Window - Generic Window Handler
Select
Cancel
Help
Use Procedure Wizard

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

The **Procedure Main Properties** dialog appears. Normally, each procedure already contains defaults or starting points for such elements as the application window, a basic menu structure, reports and more. These default procedures are designed with real world uses in mind, such as 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.

❏ Customize the application's menu, adding new menu commands and the procedures they execute.

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 the *Creating Menus and Toolbars* chapter for details on editing the menu.

Typically, you add a menu item by pressing the **New 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."

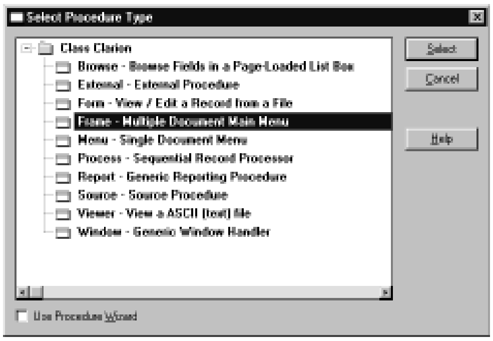
When creating an Multiple Document Interface (MDI) application, check the **Initiate Thread** box when prompted.

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

❏ Add functionality to your 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**.

Select a Procedure type from the **Select Procedure Type** dialog and press **Select**. At this point, you might choose, for example, a Browse template, which displays records in a list box.



Class Clarion
- 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
- Viewer - View a ASCII (text) file
- Window - Generic Window Handler

☐ Use Procedure Wizard

If you check the **Use Procedure Wizard** box, the **Browse Wizard, Form Wizard**, or **Report Wizard** will prompt you for all information necessary to complete your procedure. Otherwise, proceed manually:

❏ 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 will use. See the *Defining 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 *Defining Procedure Data* section, below, describes this process in detail. Essentially, you declare each variable in much the 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; when the **Window Formatter** appears a sample window is diaplayed. Depending on the procedure template you chose, predefined controls may appear in the window.

  Everything that appears in a 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. See the *Using Control, Code, and Extension Templates, Using the Window Formatter,* and *Setting Control Properties* chapters for more information.

  Use control templates (**Populate** menu) to place "prefabricated" controls, that is, fully functional controls with associated source code that is maintained by the Application Generator. For example, a BrowseBox template is a list control with associated code that loads and scrolls the list. Modify the code simply by changing settings in the Window Formatter.

  Use controls (**Control** menu) to place "do it yourself" controls, that is, controls with no associated source code.

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

- ❏ Generate the source code and make the application.

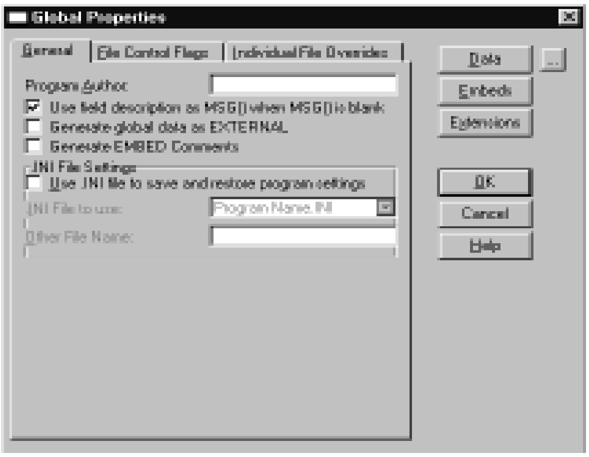
  Press the **Make** (lightning bolt) button on the toolbar to generate source, compile, and link the application. The Application Generator automatically maintains the compile and link information for the application.

- ❏ Test the application by pressing the **Run** button in the **Compile Results** dialog.

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

SPECIFYING GLOBAL PROPERTIES

You can specify a number of defaults that will apply to your entire application. You can also specify memory variables to share throughout your application. These "global" settings are specified with the **Global Properties** dialog.



Program Author:
[✓] Use field description as MSG() when MSG() is blank
[ ] Generate global data as EXTERNAL
[ ] Generate EMBED Comments
[ ] Use INI file to save and restore program settings
INI File to use: Program Name.INI
Other File Name:

Global Data Variables

All application global variables must be declared before the CODE statement in your PROGRAM module (see the *Language Reference* for more information). The easiest way to accomplish this is to set up your global variables using the **Global Properties** dialog.

To access the **Global Properties** dialog, go to the **Application Tree** dialog and press the **Global** button.

To add global data variables, press the **Data** button in the **Global Properties** dialog:

❏ 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 the data type, length, label, etc. in this dialog. See *Adding or Modifying Fields* in the previous chapter.
3. Press the **OK** button to close the **New Field Properties** dialog.

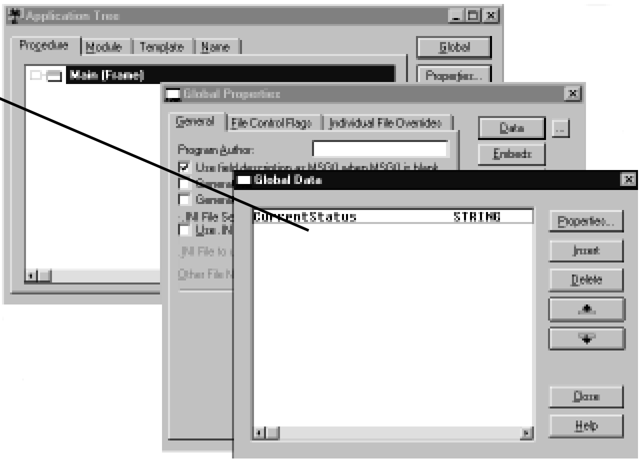
4. Press the **Close** button to close the **Global Data** dialog.

❑ 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 and press **OK**.
4. Press the **Close** button to close the **Global Data** dialog.

❑ To move a variable up in the list, highlight it, then press the ↑ button.

A list of variables. The variable types and other properties are defined in the Field Properties dialog, which you access by pressing the Properties button.

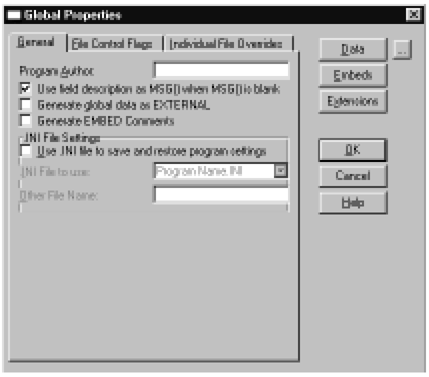


```
CurrentStatus          STRING
```

❑ To move a variable down in the list, highlight it, then press the ↓ button.

General Application Properties

The following specifications are available on the **General** tab of the **Global Properties** dialog:



Program Author:
[✓] Use field description as MSG() when MSG() is blank
[ ] Generate global data as EXTERNAL
[ ] Generate EMBED Comments
[ ] Use .INI file to save and restore program settings
.INI File to use: Program Name.INI
Other File Name:

Program Author Record bragging rights here.

Use field description as MSG() when MSG() is blank
Tells the Application Generator to use the Data Dictionary field description as the MSG attribute parameter, when no other parameter is specified. In other words, the field description becomes the default status bar message for each field in your application.

Generate global data as EXTERNAL
Adds the EXTERNAL attribute to your global variable declarations (see the *Language Reference*). This means your program will rely on an external library to allocate memory for these variables, and to declare them as public variables so your program can access them.

NOTE: If you are creating an application that consists of more than one AppGen created DLL, you MUST check the "Generate Internal Global Data as EXTERNAL" check box for all DLLs except one. Likewise, you MUST check the "Generate Internal Global Data as EXTERNAL" check box for each APP creating an .EXE.

Generate EMBED Comments
Tells the Application Generator to surround your embedded source code with comments that make your embedded source easy to find within the generated source code.

.INI File Support

The Clarion template set supports .INI (standard windows initialization) files. This file is an ASCII file which stores settings for an application between sessions.

One use for the .INI file is to store the user's preferred window positions for the next session. Many of Clarion's default procedures allow you to do this, provided you enable .INI file support.

❑ To enable automatic .INI file support:

1. Select the **General** tab in the **Global Properties** dialog.
2. Check the **Use .INI file to save and restore program settings** box.
3. Specify the .INI file name

To specify the same file name as the .EXE, with an .INI extension, choose **Program Name.INI** from the **.INI file to use** drop down list. To specify a different name, choose ***Other*** in the drop down list, then fill in the **Other File Name** field.

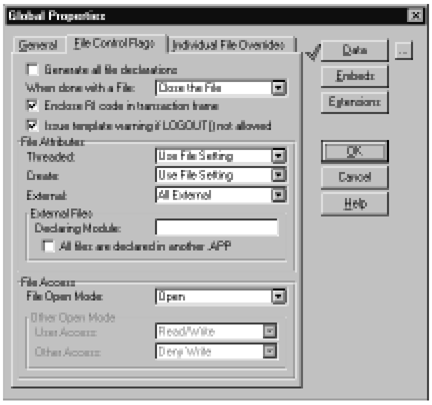
4. Press the **OK** button to close the **Global Properties** dialog.

When your application needs to revise the value in the .INI file, use the PUTINI statement. To read a value from the INI file, use the GETINI statement. See the *Language Reference* for details.

Tip: If your application requires dozens or even hundreds of variables to store from session to session, don't put them in an .INI file. Retrieving a variable from a .INI file is relatively slow. Also, if you need to hide the information from the end user, remember that .INI files are text files, and are easily accessible.

Specifying Other Global Properties - File Control

The **Global Properties** dialog also allows you to *override* some of the settings in your data dictionary. You can also define how procedures will access files. You can specify these file attributes for *all* files, or for *each* file individually. To access these features, select the **File Control Flags** tab.



☐ Generate all file declarations
When done with a File: Close the File
☑ Enclose RI code in transaction frame
☑ Issue template warning if LOGOUT() not allowed

File Attributes
Threaded: Use File Setting
Create: Use File Setting
External: All External

External Files
Declaring Module:
☐ All files are declared in another .APP

File Access
File Open Mode: Open

Other Open Mode
User Access: Read/Write
Other Access: Deny Write

Generate all file declarations

Checking this box tells the Application Generator to declare all files in the Data Dictionary, whether or not they are used within the application. By declaring all files, you can reference the files in any hand coded source you may add to your application.

When done with a File

Specifies whether the application automatically closes each file when a procedure is finished.

Tip: **One way you can design your application to be a well behaved Windows application is not to hog system resources. One limited resource is file handles. You can "give back" file handles not in use by selecting *Close the File* from the drop down list.**

Enclose RI code in transaction frame

Enables rollback of data if an update fails during a Referential Integrity maintenance operation.

Tip: **If all files in a relation chain are using the same file system, and the file system supports transaction framing, and you do not want transaction framing around the RI code, you must clear the check box for each file in Individual File Overrides and in Global Settings.**

Issue template warning if LOGOUT() not allowed

When your data dictionary includes a file driver which does not support the LOGOUT statement (used in the Referential Integrity checking routines), checking this box enables a warning at compile time.

You should be sure that this check box is *un*checked for drivers such as dBase III. See the *Database Drivers* appendix for more information.

Threaded

Specifies whether the application generator will add the THREAD attribute to FILE structures.

THREAD is needed for MDI browse and form procedures, to prevent record buffer conflicts when the end user changes focus from one thread to another.

Use File Setting sets the THREAD attribute according to the data dictionary. ***All Threaded*** and ***None Threaded*** are self explanatory.

Create

Specifies whether your application should allow the creation of a data file should it not exist. Adds the CREATE attribute to the FILE structure.

Use File Setting sets the CREATE attribute according to the data dictionary. ***Create All*** and ***Create None*** are self explanatory.

External Specifies whether the EXTERNAL attribute is added to your application files. EXTERNAL specifies that the file is defined in an external library and therefore may be referenced in the Clarion code. The memory for the FILE's record buffer is allocated by the external library. See the *Language Reference* for more information.

None External omits the EXTERNAL attribute from all file declarations. ***All External*** adds the EXTERNAL attribute to all file declarations *and* allows you to specify the **Declaring Module** and whether **All files are declared in another .APP.**

Note: When using EXTERNAL to declare a FILE shared by multiple libraries (.LIBs, or .DLLs and .EXE), only one library should define the FILE without the EXTERNAL attribute. This ensures that there is only one record buffer allocated for the FILE and all the libraries and the .EXE will reference the same memory when referring to data elements from that FILE.

Declaring Module The filename (without extension) of the MEMBER module containing the FILE definition without the EXTERNAL attribute. If the FILE is defined in a PROGRAM module, leave this field blank.

All files are declared in another .APP
***?

File Open Mode Specifies how your application opens files. ***Open*** opens files as Read/Write(primary user) + Deny Write(all other users). ***Share*** opens files as Read/Write(primary user) + Deny None(all other users). See the *Language Reference* for more information.

Choose ***Other*** to specify a custom combination of primary user + other user access. For the primary **User Access**, choose from ***Read Only***, ***Write Only***, or ***Read and Write***. For **Other Access**, choose from ***Deny None, Deny All, Deny Read, Deny Write***, and ***Any Access*** (FCB compatibility mode).

Individual File Overrides

Select the **Individual File Overrides** tab to specify the above settings for individual files in the data dictionary. Highlight the file whose attributes you want to chnage, and press the **Properties** button.

Use File Setting	sets the attribute according to the data dictionary.
Use Default	sets the attribute according to the **File Control Flags** tab.

Embed Points in the Global Properties Dialog

Through embed points, the **Global Properties** dialog provides access to the application program's global data section, to the section that opens all files, and to the section that processes errors encountered when opening the files.

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.

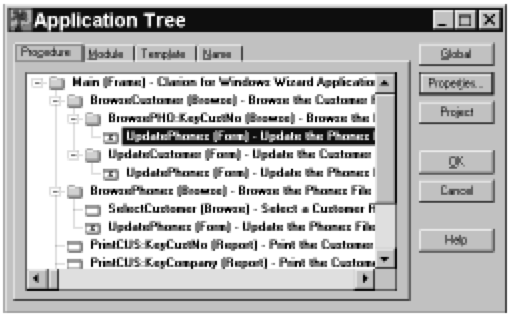
See the *Defining Embedded Source Code* section below, for more information on adding embedded source code to your application.

ADDING A PROCEDURE TO YOUR APPLICATION

A procedure is a collection of instructions—Clarion language statements—which perform a task. When the program *performs* the task, such as printing a report, it is *executing* the procedure. The first procedure your application executes is called "Main" by default. You can name your first procedure anything you want, but when this chapter refers to the first procedure, we will refer to "Main." All other procedures branch from "Main"—one procedure can call another.

The heirarchical 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 (an "S" on the procedure icon means the procedure contains embedded source code).

The Application Tree shows the procedures you create when you add a menu item, a toolbar command, or an embedded source procedure. Each new procedure is marked “To Do.” When you “fill in” its functionality, the Application Tree replaces the “To Do” with your description.



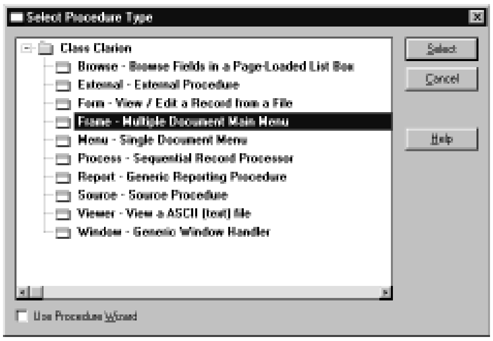
Main (Frame) - Clarion for Windows Wizard Applicatio
BrowseCustomer (Browse) - Browse the Customer F
BrowsePHO:KeyCustNo (Browse) - Browse the
UpdatePhones (Form) - Update the Phones
UpdateCustomer (Form) - Update the Customer
UpdatePhones (Form) - Update the Phones
BrowsePhones (Browse) - Browse the Phones File
SelectCustomer (Browse) - Select a Customer R
UpdatePhones (Form) - Update the Phones File
PrintCUS:KeyCustNo (Report) - Print the Customer
PrintCUS:KeyCompany (Report) - Print the Custom

Defining the Procedure Type

Once the procedure appears on the Application Tree, the first step is to *define its procedure type* from the choices available in the **Select Procedure Type** dialog.

❏ 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 procedure.

❏ To define the procedure type for your application’s procedures, highlight a procedure type from the **Select Procedure Type** dialog list, then press the **Select** button. You can also DOUBLE-CLICK on a procedure type. If you have selected a Browse procedure, a Form procedure, or a Report procedure, *and* you have checked the **Use Procedure Wizard** box, a **Wizard** dialog will appear, guiding you through each step of the procedure properties definition. If you have not checked the **Use Procedure Wizard** box, the **Procedure Properties** dialog appears.



Class Clarion
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
Viewer - View a ASCII (text) file
Window - Generic Window Handler

Use Procedure Wizard

If you must change the procedure *type* later, 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" some other previously defined procedures. Therefore, be cautious when changing procedure type.

Defining the 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 source code
- *parameters* passed to the procedure
- *return values* from functions
- *INI file settings* used by the procedure
- *files* accessed by the procedure
- *window* displayed by the procedure, including its size, shape, appearance and functionality
- *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 source code that *extends* the procedure

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 generated with this template.

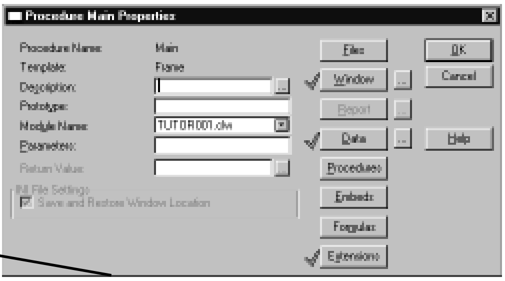
For the properties you do define, use a **Wizard** dialog, or the **Procedure Properties** dialog and its subordinate dialogs and formatters. The **Wizard** dialogs supply defaults for many of the procedure properties, and then prompt you, step-by-step, for the other information necessary to complete the procedure. The **Procedure Properties** dialogs allow for a more flexible definition of procedure properties.

This section will primarily be concerned with the **Procedure Properties** dialogs. The properties discussed in this section are common to *all* **Procedure Properties** dialogs. You define these properties by completing the entry boxes and using the command buttons on the **Procedure Properties** dialogs.

The following chapters discusses additional procedure properties, their prompts, and the various procedure templates that drive them.

❑ To define your procedure properties:

From the Application Tree, select a procedure and press the **Properties** button to access the **Procedure Properties** dialog. Alternatively, *double-click* on the procedure, or *right-click* and choose **Properties** from the popup menu.



The Procedure Properties dialog. Other dialogs may include additional prompts for specifying additional functionality.

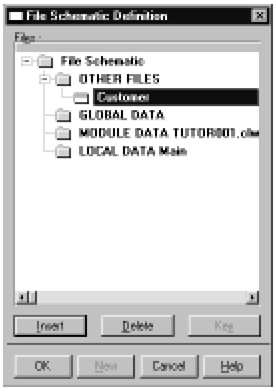
Description	A short text description for the procedure, which appears next to the procedure name in the **Application Tree** dialog. Press the ellipsis (...) button to edit a longer (up to 1000 characters) description.
Prototype	Optionally specify a *custom* procedure prototype (see the *Language Reference* for details) 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.
Module Name	Specifiy 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.
Parameters	Allows you to specify parameter names (an optional list of labels separated by commas) for your procedure, which you pass to it from a calling procedure. You must provide the functionality for the parameters in your source code.

Return Value	Specify the variables receiving return values from procedures prototyped as functions (functions return values, procedures do not).
Save and Restore Window Location	Specify whether this procedure will save and restore its window location using an .INI file.

Defining Procedure Files

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 will be used so it can provide source code for reading the file.

❑ *To specify which files your procedure will use*, press the **Files** button in the **Procedures Properties** dialog.



❑ *To add a file to the file schematic*, select an item in the **Files** list, and press the **Insert** button. Choose a file from the **Insert File** dialog.

The first file you add for a control is always the “primary” file. All others are secondary.

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

❑ *To specify the sort sequence of a file*, select it in the **Files** list, and press the **Key** button. Then choose a key from the **Change Access Key** dialog.

Defining Procedure Windows

The **Window Formatter** allows you to visually design the size, shape, menus, controls and functionality for the window in this procedure. Access the **Window Formatter** by pressing the **Window** button in the **Procedure Properties** dialog, or from the Application Tree, select a procedure, RIGHT-CLICK and choose **Window** from the popup menu. See the *Using the Window Formatter* chapter for details on how to define your window.

Defining Procedure Reports

The **Report Formatter** allows you to visually design the size, shape, content, layout, and functionality for the report in this procedure. Access the **Report Formatter** by pressing the **Report** button in the **Procedure Properties** dialog, or from the Application Tree, select a procedure, RIGHT-CLICK and choose **Report** from the popup menu. See the *Using the Report Formatter* chapter for more information.

Defining Procedure Data

Procedures may access several classes of data. These include GLOBAL data (see *Specifying Global Properties*, this chapter), MODULE data, LOCAL data, and file or field data (see *Defining Procedure Files*, this chpater, *Using the Window Formatter* and *Setting Control Properties*).

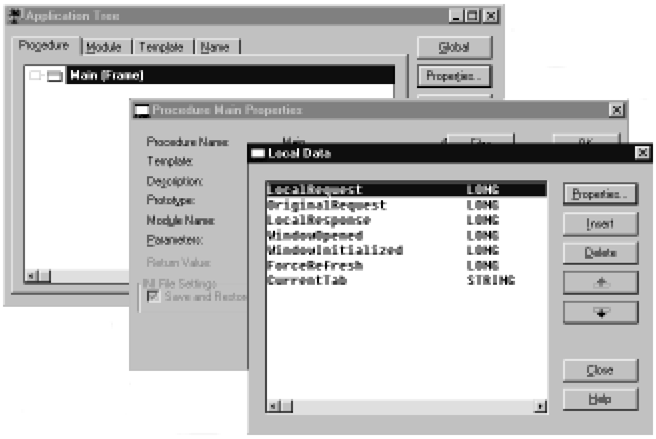
GLOBAL data may be accessed by any procedure in the entire application. MODULE data may only be accessed by the procedures contained in the *module* where the data are defined. LOCAL data may only be accessed within the single *procedure* where the data are defined. See the *Language Reference* section on *Data Declarations and Memory Allocation* for more information.

Defining LOCAL Data and MODULE Data

❑ To define LOCAL data (memory variables local to a single *procedure*) to a procedure:

1. Select a procedure in the **Application Tree** dialog.
2. Press the **Properties** button, choose **Edit ➤ Properties,** or RIGHT-CLICK and choose **Properties** from the popup menu to display the **Procedure Properties** dialog.
3. Press the **Data** button to display the **Local Data** dialog.

   If any local variables already exist, they appear in the list.



```
LocalRequest          LONG
OriginalRequest       LONG
LocalResponse         LONG
WindowOpened          LONG
WindowInitialized     LONG
ForceRefresh          LONG
CurrentTab            STRING
```

4. Press the **Insert** button and define the variable.

   The **New Field Properties** dialog appears. Type in the variable name, choose the variable type, and set any additional attributes, including screen attributes. You can also specify how memory is allocated for the variable.

   The **Allocation** drop down list on the **Attributes** tab allows you to specify that the variable is allocated uninitialized memory (AUTO attribute). It allows you to specify that the variable is EXTERNAL (memory is allocated in a .DLL). It allows you to specify that the variable is STATIC (the value persists from one instance of the procedure to another). It allows you to add the THREAD attribute, which provides a thread specific static variable.

   This is the *same* dialog used to set field properties in the data dictionary. For details, see the *Using the Data Dictionary* chapter.

5. Close the **Field Properties** and the **Local Data** dialogs.

   The data variables are now included in the procedure.

❑ To define MODULE data (memory variables local to the procedures in a single *module*):

1. From the **Application Tree** dialog, select the **Module** tab.
2. Highlight a module (folder) inside the **Application Tree** dialog.
3. Press the **Properties** button to display the **Module Properties** dialog, or RIGHT-CLICK and choose **Data** from the popup menu.
4. Press the **Data** button.

   The **Module Data** dialog appears.

5. Press the **Insert** button.

   The **Field Properties** dialog appears. This is the *same* dialog used to set field properties in the data dictionary. For details, see *Using the Data Dictionary*, and the section above this one.

6. Close the **Field Properties**, the **Module Data**, and the **Module Properties** dialogs.

Defining Calls to Other Procedures

Procedures may call other procedures. Press the **Procedures** button to access the **Called Procedures** dialog. To add a procedure, press the **Insert** button, and type a procedure name in the next dialog. To delete a called procedure, press the **Delete** button. Additional buttons allow you to change the order of the called procedures listed.

Tip: The main purpose of the Procedures button is to add procedures called from embedded source to the Application Tree. Procedures called any other way are automatically added to the tree.

Defining Embedded Source Code

Embedding source code in a procedure allows you to fully customize your procedure. You can specify or create code to execute before, during, and after the procedure. You can write your own code, or use code templates which help write the code for you.

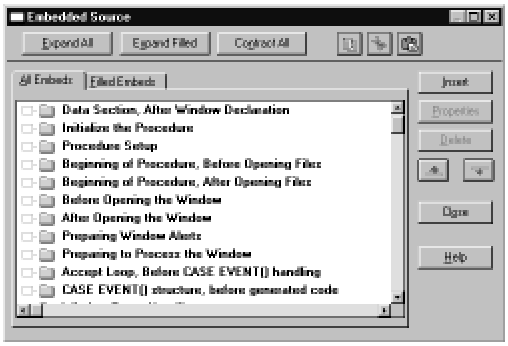
In order to effectively embed code, you should understand the generated code you are embedding into. Study a browse procedure and a form procedure (C:\CW15\EXAMPLES\APPS\TREE017.CLW and TREE019.CLW) to familiarize yourself with the coding conventions typical of Clarion's templates.

The Application Generator adds your embedded code to the code it generates, at *precisely* the point you specify based on predefined embed points. For example, any procedure with a window includes points for embedding code immediately before and immediately after opening the window. However, you can add your own embed points if needed. See the *Template Language Reference* for details on modifying the templates.

Once you embed source code, the procedures containing embedded source are flagged with an "S" on the procedure's icon in the Application Tree.

❑ To embed custom source code in a procedure:

1. In the **Application Tree** dialog, highlight a procedure and press the **Properties** button, OR RIGHT-CLICK and choose **Embeds** from the popup menu.
2. Press the **Embeds** button in the **Procedure Properties** dialog to display the **Embedded Source** dialog.


Embedded Source
Expand All
Expand Filled
Contract All
All Embeds
Filled Embeds
Data Section, After Window Declaration
Initialize the Procedure
Procedure Setup
Beginning of Procedure, Before Opening Files
Beginning of Procedure, After Opening Files
Before Opening the Window
After Opening the Window
Preparing Window Alerts
Preparing to Process the Window
Accept Loop, Before CASE EVENT() handling
CASE EVENT() structure, before generated code
Insert
Properties
Delete
Close
Help

The **Embedded Source** dialog lists points within the procedure where your custom source code may be inserted. This includes the points where the *field specific events* occur within the procedure. For example, if you place an entry box in a window, the embed points you can access include points where the user selects (TABS onto or mouse CLICKS on) the field, and points where the user completes or accepts (TABS off, presses the ENTER key, or presses the **OK** button) the field.

The following code was generated by a generic window template, to illustrate standard embed points. Each embed point is marked by a comment (exclamation points initiate comments) showing the label of the embed point as it appears in the **Embedded Source** dialog. Remember, you can add your own embed points if needed.

```
ShowEmbeds PROCEDURE

LocalRequest         LONG,AUTO
OriginalRequest      LONG,AUTO
LocalResponse        LONG,AUTO
WindowOpened         LONG
WindowInitialized    LONG
ForceRefresh         LONG,AUTO
CurrentTab           STRING(80)
LastName             STRING(20)
!!! Data Section, Before Window Declaration
window               WINDOW('Caption'),AT(,,260,100),GRAY
                       ENTRY(@s20),AT(108,25),USE(LastName)
                       BUTTON('Ok'),AT(130,72),USE(?Ok)
                     END
!!! Data Section, After Window Declaration
  CODE
  !!! Initialize the Procedure
  LocalRequest = GlobalRequest
  OriginalRequest = GlobalRequest
  LocalResponse = RequestCancelled
  ForceRefresh = False
  CLEAR(GlobalRequest)
  CLEAR(GlobalResponse)
  !!! Procedure Setup
  !!! Beginning of Procedure, Before Opening Files
  !!! Beginning of Procedure, After Opening Files
  !!! Before Opening the Window
```

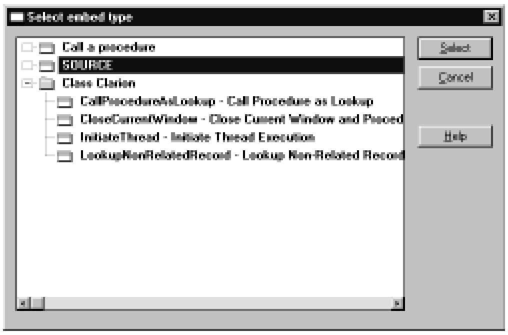
```
OPEN(window)
WindowOpened=True
!!! After Opening the Window
!!! Preparing Window Alerts
!!! Preparing to Process the Window
ACCEPT
  !!! Accept Loop, Before CASE EVENT() handling
  CASE EVENT()
  !!! CASE EVENT() structure, before generated code
  OF EVENT:AlertKey
    !!! Window Event Handling (AlertKey)
  OF EVENT:PreAlertKey
    !!! Window Event Handling (PreAlertKey)
  OF EVENT:CloseWindow
    !!! Window Event Handling (CloseWindow)
  OF EVENT:CloseDown
    !!! Window Event Handling (CloseDown)
  OF EVENT:OpenWindow
    !!! Window Event Handling (OpenWindow)
    IF NOT WindowInitialized
      DO RefreshWindow
      WindowInitialized = True
    END
    SELECT(?LastName)
  OF EVENT:LoseFocus
    !!! Window Event Handling (LoseFocus)
  OF EVENT:GainFocus
    !!! Window Event Handling (GainFocus)
    ForceRefresh = True
    WindowInitialized = True
    DO RefreshWindow
  OF EVENT:Suspend
    !!! Window Event Handling (Suspend)
  OF EVENT:Resume
    !!! Window Event Handling (Resume)
  ELSE
    !!! Other Window Event Handling
  !!! CASE EVENT() structure, after generated code
  END
  !!! Accept Loop, After CASE FIELD() handling
  !!! Accept Loop, Before CASE FIELD() handling
  CASE FIELD()
  !!! CASE FIELD() structure, before generated code
  OF ?LastName
    !!! Control Handling, before event handling (?LastName)
    CASE EVENT()
    OF EVENT:Accepted
      !!! Control Event Handling, before generated code (?LastNAme, Accepted)
      !!! Control Event Handling, after generated code (?LastName, Accepted)
    OF EVENT:Rejected
      !!! Control Event Handling, before generated code (?LastName, Rejected)
      !!! Control Event Handling, after generated code (?LastName, Rejected)
    OF EVENT:Selected
      !!! Control Event Handling, before generated code (?Lastname, Selected)
      !!! Control Event Handling, after generated code (?OLastName, Selected)
    ELSE
      !!! Other Contorl Event Handling (?LastName)
      END
      !!! Control Handling, after event handling (?LastName)
    OF ?Ok
      !!! Control Handling, before event handling (?Ok)
      CASE EVENT()
      OF EVENT:Accepted
        !!! Control Event Handling, before generated code (?Ok, Accepted)
        DO SyncWindow
        !!! Control Event Handling, after generated code (?Ok, Accepted)
      OF EVENT:Selected
        !!! Control Event Handling, before generated code (?Ok, Selected)
        !!! Control Event Handling, after generated code (?Ok, Selected)
      ELSE
        !!! Other Control Event Handling (?Ok)
      END
      !!! Control Handling, after event handling (?Ok)
    !!! CASE FIELD() structure, after generated code
    END
    !!! Accept Loop, After CASE FIELD() handling
  END
  DO ProcedureReturn
!-----------------------------------------------------------------------------
ProcedureReturn ROUTINE
  !!! End of Procedure, Before Closing Files
  !!! End of Procedure, After Closing Files
  !!! Before Closing the Window
  IF WindowOpened
    CLOSE(window)
  END
  !!! After Closing the Window
  !!! End of Procedure
  IF LocalResponse
    GlobalResponse = LocalResponse
  ELSE
    GlobalResponse = RequestCancelled
  END
  RETURN
!-----------------------------------------------------------------------------
InitializeWindow ROUTINE
  !!! Window Initialization Code
  DO RefreshWindow
!-----------------------------------------------------------------------------
RefreshWindow ROUTINE
  IF window{Prop:AcceptAll} THEN EXIT.
  !!! Refresh Window routine, before lookups
  !!! Lookup Related Records
  !!! Refresh Window routine, after lookup
  !!! Refresh Window routine, before DISPLAY()
  DISPLAY()
  ForceRefresh = False
!-----------------------------------------------------------------------------
SyncWindow ROUTINE
  !!! Sync Record routine, before lookups
  !!! Lookup Related Records
  !!! Sync Record routine, after lookups
!-----------------------------------------------------------------------------
!!! Procedure Routines
```

3. Select a point at which to embed the code and press the **Insert** button.

   The **Select Embed Type** dialog appears. There are three ways to create the embedded source code: hand-coding with the text editor, calling another procedure, or embedding a template. You can even combine all three methods.

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

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


Select embed type
Call a procedure
SOURCE
Class Clarion
CallProcedureAsLookup - Call Procedure as Lookup
CloseCurrentWindow - Close Current Window and Proced
InitiateThread - Initiate Thread Execution
LookupNonRelatedRecord - Lookup Non-Related Record
Select
Cancel
Help

2. Press the **Select** button to start the Text Editor with a blank source code window.
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. Press the **Close** button to close the **Embedded Source** dialog.

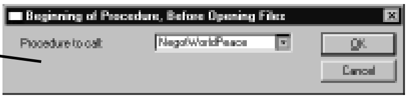
❏ To call another procedure:

1. Select the **Call a Procedure** item in the **Select Embed Type** dialog.
2. Type a name for the procedure or choose an existing procedure from the drop down list which appears in the next dialog. The caption of the dialog box corresponds to the embed point chosen.

   Typing a new name specifies that the application calls another procedure, which automatically appears in the Application Tree as a "To Do." If another procedure with the same name already exists, the Application Generator assumes you meant to call it, and does not add a new "To Do."

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

The Select Embed Type dialog allows you to call a procedure, add your own custom code, or use a code template to help you write your own code.


Beginning of Procedure, Before Opening Files

Procedure to call: NegotWorldPeace

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

❑ To use a code template to help construct the code:

Code templates help you construct source code with minimal effort on your part. They can automatically take care of complex control structures and property assignments. When you select a code template to embed, Clarion displays a dialog box containing an explanation of what the template does as well as prompts for the information required to complete the source code.

The names of the available code templates appear in the **Select Embed Type** dialog under the Class Clarion item. For example, the **Initiate Thread** template makes starting a new execution thread (for creating and opening an MDI window from the Application frame) simple.

See the *Using Control, Code, and Extension Templates* chapter for descriptions of the templates included with Class Clarion.

1. Select a code template and press the **Select** button.

   This displays a **Prompts for**... dialog box (the title includes the name of the code template).

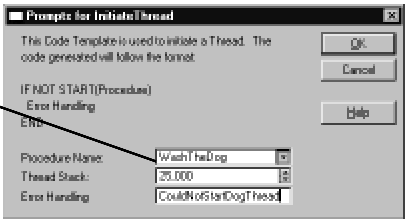
2. Read the instructions and explanations in the dialog.

   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 inside the **Prompts for**... dialog.

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

Filling in the Initiate Thread code template. Here it STARTs a procedure called WashTheDog.


Prompts for InitiateThread

This Code Template is used to initiate a Thread. The code generated will follow the format:

```
IF NOT START(Procedure)
  Error Handling
END
```

Procedure Name: WashTheDog

Thread Stack: 25,000

Error Handling: CouldNotStartDogThread

❑ To use any combination of hand-coding, calling another procedure, or embedding a template:

1. Follow one set of instructions above for creating the first embedded source code item.

After closing the **Select Embed Type** dialog, your embed point appears in the list. If you added source, an item called “Source” appears, followed by the first line of code, to help you identify it when you look at the list. If you added a procedure, an item called “PROCEDURE,” followed by the procedure name appears. If you added a code template, the name of the code template appears.

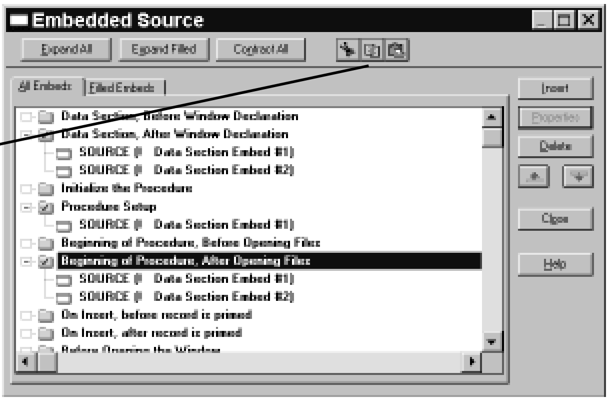
2. Select the item you inserted in the **Embedded Source** dialog.
3. Press the **Insert** button.
4. Add another Procedure, Source, or Code Template, according to the directions in the previous sections.

   The **Embedded Source** dialog also contains ↑ and ↓ buttons to change the order of the multiple embedded source items; execution occurs in the order they are listed. There are also **Delete** and **Properties** buttons for maintenance.
5. Press the **Close** button to close the **Embedded Source** dialog.

❏ 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 **Cut** or **Copy** button.
3. Again, highlight a line in the tree diagram.
4. Press the **Paste** button.



Data Section, Before Window Declaration
Data Section, After Window Declaration
SOURCE (! Data Section Embed #1)
SOURCE (! Data Section Embed #2)
Initialize the Procedure
Procedure Setup
SOURCE (! Data Section Embed #1)
Beginning of Procedure, Before Opening Files
Beginning of Procedure, After Opening Files
SOURCE (! Data Section Embed #1)
SOURCE (! Data Section Embed #2)
On Insert, before record is primed
On Insert, after record is primed
Before Opening the Window

Use the cut, copy, and paste buttons to transfer one or more embedded items from one embed point to another.

Defining Procedure Formulas

You can access the **Formula Editor** by pressing the **Formula** button in the **Procedure Properties** dialog. This allows you to create expressions, and store the results in a memory variable or a field within a file. You can then display the result in a window or report control.

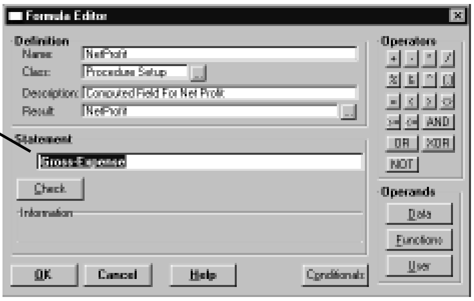
The Formula Editor can create simple assignment statements and complex conditional structures. See the *Using the Formula Editor* chapter for details on how to create the actual expression.

❑ When using the Formula Editor with the Application Generator:

1. Press the **Formulas** button in the **Procedure Properties** dialog to open the **Formula Editor** dialog.
2. Type a name for the expression in the **Name** box.
3. Choose a class for the expression by pressing the ellipsis (...) button.

   The class determines where the formula will be inserted into the procedure: such as at procedure setup, before lookups, after lookups, at procedure exit, or at prime fields. These formula classes should not be confused with template classes.
4. Type the name of a field or variable in the **Result** box, or press the ellipsis (...) button next to it to choose one from the **Select Field** dialog.
5. Fill in the other options in the **Formula Generator** dialog, according to the instructions in the *Using the Formula Generator* chapter.

Creating a Formula.



Name: NetProfit
Class: Procedure Setup
Description: Computed Field For Net Profit
Result: NetProfit
Statement: Gross-Expense

6. Close the Formula Editor.

❑ Usually, you'll display the result of the expression in a string control in a window or report. To display it in a window:

1. Press the **Window** button to open the **Window Formatter**.

   If there is not yet a window defined for the procedure, you can define one.

2. Select the string control tool from the **Controls** toolbox (or choose **Control ➤ String**), and CLICK in the window to place the control.
3. With the new string control selected, choose **Edit ➤ Properties** (or RIGHT-CLICK on the string and choose **Properties** from the popup menu).
4. Check the **Variable String** check box in the **String Properties** dialog.

   The **Select Field** dialog appears.
5. Highlight the variable you want to display and press the **Select** button.

   The **String Properties** dialog returns, note however, that the **Parameter** field has now been replaced by the **Picture** field.
6. Type a valid display picture in the **Picture** field.
7. Press the **OK** button to close the **String Properties** dialog.
8. Choose **Exit!** to close the **Window Formatter** and return to the **Procedure Properties** dialog.

Defining Procedure Extensions

Extension and control templates provide additional functionality to basic procedure templates. *Control templates* give your procedure the ability to display and manage *specific controls*. For example a browse box may be added using a 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. The *Using Control, Code and Extension Templates* chapter provides additional information on using these templates.

From the **Procedure Properties** dialog press the **Extensions** button to display the **Extension and Control Templates** dialog. Or, from the Application Tree, select a procedure, *right-click*, and choose **Extensions** from the popup menu. This dialog displays a list of control and extension templates *and* the prompts associated with each template. Highlighting or selecting a template on the *left* side of the dialog causes the prompts associated with the selected template to be displayed on the *right* side of the dialog.

Add additional *extension* templates by pressing the **Insert** button. Customize existing templates by filling in the prompts on the *right* side of the dialog.

Tip: ***Only Extension templates*** **may be added and deleted using the Extensions button. Control templates may** ***not*** **be added or deleted, but may be modified. Control templates may be added or deleted from the Window Formatter by adding or deleting their associated controls.**

MAINTAINING THE APPLICATION TREE

When the **Application Tree** dialog is active, Clarion provides menus for maintaining the tree, and tabs for providing different views of the tree. The menus used for maintaining the tree are the **Edit** menu, the **Application** menu, and the **Procedure** menu.

Using the Edit Menu

With the **Application Tree** dialog active, you can execute the following commands from the **Edit** menu:

Properties	Calls the currently selected procedure's **Procedure Properties** dialog. Equivalent to the **Properties** button.
Find	Allows you to search 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	Allows you to search for another procedure, using the same search string as the previous search. If you did not search previously, the **Search for Procedure** dialog appears.
Edit by Name	Allows you to type the name of a procedure in the **Edit Procedure by Name** dialog, then opens the **Procedure Properties** dialog of the procedure you typed in. This can be very useful in a large application with many procedures.
Delete	Deletes the currently selected procedure code, properties, etc. The procedure remains as a ToDo item in the Application Tree because the procedure is still called by another procedure. To remove the procedure from the Application Tree, you must remove all references to the procedure.

Using the Application Menu

With the **Application Tree** dialog active, you can execute the following commands from the **Application** menu:

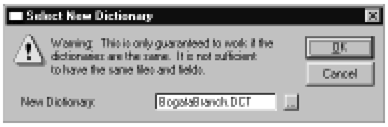
Properties

Displays the **Application Properties** dialog for specifying changes to the .APP file.

Global Properties Displays the **Global Properties** dialog. Equivalent to using the **Global** button in the **Application Tree** dialog.

Change Dictionary Allows you to name 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: BogotaBranch.DCT

Insert Module Specifies a new MODULE for generated source code. You can also specify an external .LIB or .OBJ file to add to the project.

Template Utility Allows you to 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.

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, however, actual source module files will not be 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, however, actual source module files will not be affected until the next time source is generated

Renumber Modules

Renumbers modules. This is useful when empty modules have been deleted. Please note this menu option immediately affects the Application Tree, however, actual source module files will not be 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, however, *module files (.CLW) on your disk drive are not deleted*.

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, however, *library files on your disk drive are not deleted*.

Using the Procedure Menu

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 keydoes the same thing,
Rename	Allows you to change the name of the currently selected procedure. Type a new name in the **Rename** dialog box.

Copy	Allows you to copy the currently selected procedure. Type a new name in the **New Procedure** dialog box.
Change Module	Allows you to 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	Allows you to 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" some other previously defined procedures. Therefore, be cautious when changing procedure type.

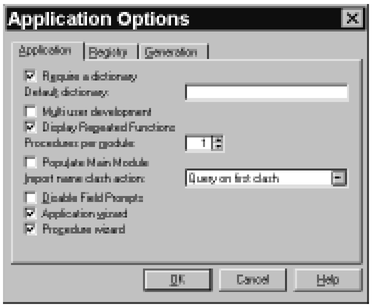
Using the View Tabs

The **Application Tree** dialog allows you to view your procedures in four different arrangements. Change the arrangement by selecting the desired tab.

Procedure	Displays procedures in heirarchical order, nesting each procedure under its calling procedure. This is the default view, and combined with the "To Do" legends for undefined procedures, is the best view for determining what remains to be done to complete the application.
Module	Each procedure appears nested under the name of the module file to which its source code generates.
Template	Lists procedures according to procedure type.
Name	Lists procedures alphabetically by name. This is sometimes convenient for large projects.

SETTING APPLICATION OPTIONS

The **Application Options** dialog allows you to specify default settings for *each new application* you create as well as for the currently active application. To access the dialog, choose **Setup ➤ Application Options**. The dialog is divided into three sections or tabs: **Application Options**, **Registry Options**, and **General Options**.



- [✓] Require a dictionary
- Default dictionary:
- [ ] Multi user development
- [✓] Display Repeated Functions
- Procedures per module: 1
- [ ] Populate Main Module
- Import name clash action: Query on first clash
- [ ] Disable Field Prompts
- [✓] Application wizard
- [✓] Procedure wizard

Application

Require a dictionary Specifies that each new application *must* have a data dictionary.

Default dictionary Specifies a default data dictionary file name that appears in the **Application Properties** dialog whenever a new application is being created. You may change to another dictionary before closing the dialog, however, a single data dictionary may be used to support multiple applications.

Multi user development
Tells the Dictionary Editor and the Application Generator to open the Data Dictionary and REGISTRY.TRF in read only mode, so many developers can work with the same dictionary. See the *Multi-Programmer Development* appendix for more information.

Display Repeated Functions
Specifies whether the Application Generator re-displays procedures called from more than one place within the Application Tree, or simply refers to them as “Expanded Above”.

Procedures per Module

The **Procedures per Module** spin control specifies the number of *procedures* that the Application Generator writes to each source code *module*. This can affect compile time when the Conditional Generation switch is turned on. Specifying one procedure per module, for example, means that each successive compile rebuilds *only* those procedures changed since the last compile. The down side to this is that it requires more disk space. Generally, a smaller number is faster.

Populate Main Module

Specifies that the Application Generator writes procedures to the *main* source code module. When this option is off, the main module contains only the program global data and code. All other procedures reside in other files.

Import name clash action

Specifies how the Application Generator handles procedure names from an imported application file which clash with procedure names already resident. The drop down list choices are self-explanatory.

Disable Field Prompts

Specifies that template-generated field-specific prompts will not display on **Action** tabs. This does not disable prompts created by Control Templates.

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** will build an entire application based on your data dictionary.

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, or Report procedure.

Registry

See *Setting Template Registry Options* below.

Generation

Conditional Generation	Specifies that only source code modules *changed* since the last make should be compiled.
Debug Generation	Specifies a text file for the Application Generator to log events to, and turns logging on and off. 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 will be displayed for your information during source code generation.

MAINTAINING YOUR TEMPLATES

Template files (*.TPL) drive the Application Generator. Each procedure template contains generic or “model” code, as well as prompts for additional information neede 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 your application, Clarion preprocesses the template class and stores the information in the REGISTRY.TRF file. Preprocessing occurs only when the Application Generator detects a new or changed template.

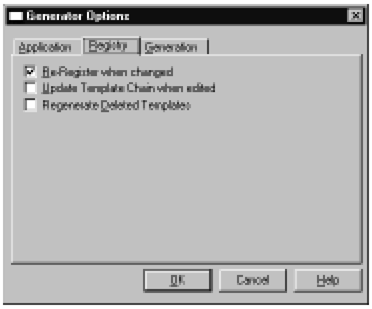
  When it preprocesses the template set, the Application Generator stores a list of all the information you must provide to each procedure. It also determines the points where you can embed your own Clarion source code to customize a procedure.

- 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, and 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.

Setting Template Registry Options

To set registry options, choose **Setup ➤ Application Options**. The **Generator Options** dialog appears. Select the **Registry** tab.



☑ Re-Register when changed
☐ Update Template Chain when edited
☐ Regenerate Deleted Templates

Template language code can be stored among many files, typically .TPW and .TPL files. Clarion uses these files to produce one logical template class (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 allows you to specify how the template language files and the one logical template class are managed for your applications. These options are mainly for programmers who produce their own template files or make modifications to the default templates.

❏ *To automatically re-register your templates* when the Application Generator detects a change, that is, when a .TPL or .TPW file changes, if you want the change copied, or registered, to the REGISTRY.TRF file, check the **Re-register when changed** box.

❏ *To automatically update the template files* when making a change in the **Template Registry**, that is, if you want changes in the REGISTRY.TRF file to be reflected in the .TPL and .TPW files, check the **Update Template Chain when edited** box.

❏ *To specify the Application Generator should re-generate the .TPL and .TPW files* from REGISTRY.TRF, should the files be deleted, check the **Regenerate Deleted Templates** box.

Opening the Registry

The Template Registry stores a list of all the templates (and their procedures) available to you when building an application. To create an application, you must have at least one template class registered.

Though the default template Class Clarion (CW.TPL) is pre-registered, if you need to re-register it, or if you wish to register a third party template class, this is how to do it:

1. Choose **Setup ➤ Template Registry**.

   The **Template Registry** dialog appears. This provides a heirarchical list of your templates and their procedures, plus a group of command buttons to maintain the registry.

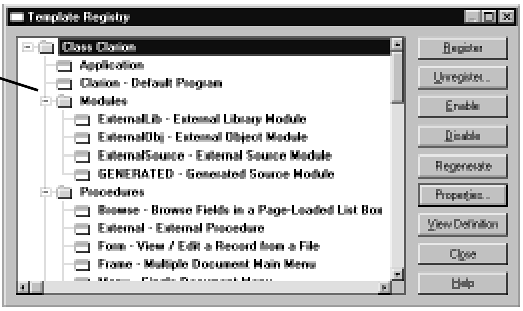
2. Press the **Register** button.

   The **Template File** dialog appears. This dialog lets you choose which templates will be registered. The **List Files of Type** specification is *.TPL. The default template subdirectory or folder is C:\CW15\TEMPLATE.

3. Select the CW.TPL file (or third party .TPL file) and press the **OK** button.

   This registers (preprocesses) the template set (class), making it available for use.

Registry Maintenance



- Class Clarion
  - Application
  - Clarion - Default Program
  - Modules
    - ExternalLib - External Library Module
    - 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
    - Form - View / Edit a Record from a File
    - Frame - Multiple Document Main Menu

The Template Registry.

The **Template Registry** dialog also provides other command buttons for other file maintenance options for the registry:

Unregister	This button deletes the currently highlighted template class from REGISTRY.TRF.

Enable	This button enables the currently highlighted template class or procedure (if you had previously disabled it).
Disable	This button disables the currently highlighted template class or procedure, which makes it unavailable to your application.
Regenerate	This button regenerates the .TPL file from the REGISTRY.TRF file for the currently highlighted template class. The Template Registry not only allows you to *add* templates and procedures to choose from, it also allows you *customize* your templates. See **Properties** immediately below, for instructions on how to customize your template set. When you change the properties for your template, the changes are stored in your REGISTRY.TRF file, effectively establishing *new default* values for your application. The **Regenerate** button reads the REGISTRY.TRF file, then rewrites the .TPL file which contained the original template language code. This allows you to customize a template, then give a copy of the changed .TPL file to another programmer to register!
Properties	This button accesses the **Template Procedure Properties** dialog. Press the **Global Data** button to edit default global data contained in this template. Press the **Defaults** button to edit default structures (windows, list boxes, etc.) contained in this template.
View Definition	This button displays the Template code (the .TPL) in a text window. You may not edit the code in this window. If the currently highlighted item in the Template Registry tree is a module, the text window opens to the first line of template language code for the #MODULE. If the currently highlighted item in the Template Registry tree is a procedure, it opens to the first line of template language code for the #PROCEDURE. See the *Template Language Reference* for more information on template language format and syntax.

APPLICATION IMPORT/EXPORT COMMANDS

When the Application Generator is active, the **File** menu contains commands for incorporating procedures from other applications. The export process is managed through the use of an export file format (.TXA) designed to help incorporate procedures written with other versions of Clarion. The import process can directly process the .APP file format in which the Application Generator stores the application in progress.

The following commands appear on the **File** menu:

Import from Application
Allows you to 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.

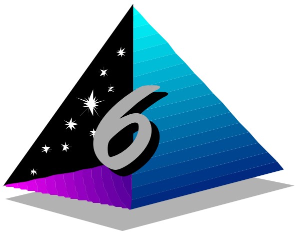
Warning: Be sure to back up *both* source and target before importing. Before importing procedures or files from earlier versions of CW .APP or .DCT files, CW 1.5 *converts* the *earlier* versions to CW 1.5 format. Once converted to CW 1.5 format, *the source files will no longer be usable by earlier versions of CW.*

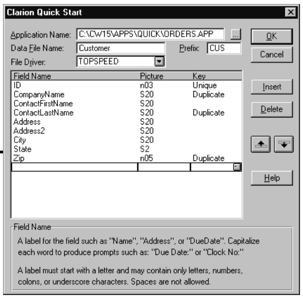
Import Text	Incorporates the procedures defined in a .TXA file (an ASCII version of an .APP or application file), created with the **Export Text** (see below) command. You will be prompted to rename or replace procedures with name conflicts.
Export Text	Allows you to create a .TXA file (an ASCII version of your .APP file) from the current application.
Selective Export	Allows you to create a .TXA file containing only the selected procedure.

USING THE PROCEDURE TEMPLATES

Contents

In this chapter, you'll find guides to the Procedure Wizards and the Procedure Templates.





Clarion Quick Start

Application Name: C:\CW15\APPS\QUICK\ORDERS.APP
Data File Name: Customer Prefix: CUS
File Driver: TOPSPEED

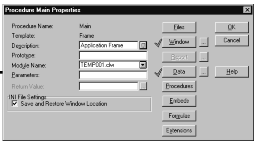
Field Name	Picture	Key
ID	n03	Unique
CompanyName	S20	Duplicate
ContactFirstName	S20	
ContactLastName	S20	Duplicate
Address	S20	
Address2	S20	
City	S20	
State	S2	
Zip	n05	Duplicate

Field Name

A label for the field such as "Name", "Address", or "DueDate". Capitalize each word to produce prompts such as: "Due Date:" or "Clock No:"

A label must start with a letter and may contain only letters, numbers, colons, or underscore characters. Spaces are not allowed.

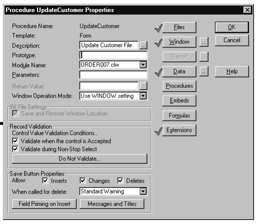
The Quick Start Wizard is one of the tools that makes Rapid Application Development truly Rapid.



Procedure Main Properties

Procedure Name: Main
Template: Frame
Description: Application Frame
Prototype:
Module Name: TEMP001.clw
Parameters:
Return Value:
INI File Settings
☑ Save and Restore Window Location

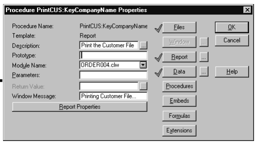
The Frame Template provides an application frame for your Multiple Document Interface applications.



Procedure UpdateCustomer Properties

Procedure Name: UpdateCustomer
Template: Form
Description: Update Customer File
Prototype:
Module Name: ORDER007.clw
Parameters:
Return Value:
Window Operation Mode: Use WINDOW setting
INI File Settings
☑ Save and Restore Window Location
Record Validation
Control Value Validation Conditions..
☑ Validate when the control is Accepted
☑ Validate during Non-Stop Select
Do Not Validate...
Save Button Properties
Allow: ☑ Inserts ☑ Changes ☑ Deletes
When called for delete: Standard Warning
Field Priming on Insert
Messages and Titles

The Browse Template provides a predefined window with a scrolling list and update buttons to call a Form (update) procedure.



Procedure PrintCUS:KeyCompanyName Properties

Procedure Name: PrintCUS:KeyCompanyName
Template: Report
Description: Print the Customer File
Prototype:
Module Name: ORDER004.clw
Parameters:
Return Value:
Window Message: Printing Customer File...
Report Properties

The Report Template allows you to create a report using the Report Formatter, and plug it into your procedure tree.

WIZARDS

Clarion for Windows provides *WIZARDS*—powerful utility templates that enable you to create a Browse, Form, or Report procedure by merely answering a few quick questions. You can even use a wizard to create an entire Application from an existing dictionary!

Options you specify in advance in the Data Dictionary provide additional control over the procedures the wizards create. See *Using Wizard Options* for more information.

Quick Start Wizard

Using the Quick Start Wizard, you can create a data dictionary and a working application with no coding required.

Simply define a data file, and the Quick Start Wizard creates a complete Windows application. The entire process takes less than five minutes; if you're a fast typist! Your application has a form procedure for updating the file, a multi-keyed browse procedure, and as many reports as the data file has keys.

Just define the fields for a single file. For each field, you provide a name, display format picture, and key information. This creates a data dictionary. The Quick Start Wizard creates the application based on this dictionary. Once you've specified all options, the OK button generates the .APP file, and loads the procedures into the Application Tree dialog.

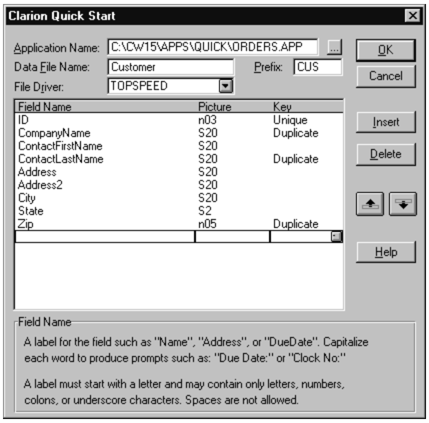
To use the Quick Start Wizard:

1. Optionally, in File Manager, choose **File ➤ Create Directory** or in Windows 95 Explorer, choose **File ➤ New ➤ Folder**, type a name and press **OK**.
2. In Clarion for Windows, choose **File ➤ New.**

   The **New** file dialog appears.
3. Select the **Application** tab.
4. Type a name for the .APP file in the **Application File** field.

   Type a legal DOS file name. Clarion automatically adds the .APP extension.
5. Check the **Use Quick Start Wizard** box below the file list, then press the **Create** button.

   The **Quick Start Wizard** dialog appears. This dialog allows you to define the file for the application.



Clarion Quick Start

Application Name: C:\CW15\APPS\QUICK\ORDERS.APP
Data File Name: Customer
Prefix: CUS
File Driver: TOPSPEED

Field Name	Picture	Key
ID	n03	Unique
CompanyName	S20	Duplicate
ContactFirstName	S20	
ContactLastName	S20	Duplicate
Address	S20	
Address2	S20	
City	S20	
State	S2	
Zip	n05	Duplicate

Field Name

A label for the field such as "Name", "Address", or "DueDate". Capitalize each word to produce prompts such as: "Due Date:" or "Clock No:"

A label must start with a letter and may contain only letters, numbers, colons, or underscore characters. Spaces are not allowed.

Application Name The DOS file name for the .APP file. The Quick Start Wizard will use the same file name (with the .DCT extension) for the data dictionary file.

Optionally press the ellipsis button (...) to change the directory, and type a file name in the Open File dialog box. The working directory, in which all source code files will be generated, depends on where the .APP file resides.

Data File Name Type a legal DOS file name (no extension necessary) for the data file.

Prefix This box automatically fills in with the first three letters of the name of the data file when you TAB from the Data File Name box. Optionally specify up to 14 letters of your choice in this field.

The prefix allows your application to distinguish between similar variable names occurring in different file structures. A field called Invoice may exist in one data file called Orders and another called Sales. By establishing a unique prefix for Orders (ORD) and Sales (SAL), the application may distinguish the two fields as *ORD:Invoice* and *SAL:Invoice*.

File Driver Specify the data file type. When using the Application Generator, Clarion for Windows automatically links in the correct database file driver library. See the *Database Drivers* appendix for a discussion of the relative advantages of each driver.

	Remember that individual file drivers may vary in their support of some of the attributes which you add to the FILE structure in this dialog box.
Field Name	To name each field, type a valid Clarion label in the Name field. Valid field names may vary slightly according to the file driver.
Picture	Specify a default picture token by typing it in the **Picture** field. The picture token, together with the selected File Driver, determine the data type which the Quick Start Wizard uses for the field. When the Application Generator creates window and report controls for the field, this also serves as the default picture for the control.
Key	This specifies whether to create a key using this field as a component, and if so, the type of key. By specifying *Unique*, your application will ensure that each record has a distinct value. *Duplicate* specifies a key that allows more than one record with the same value in the key component. The Quick Start Wizard creates a multi-keyed browse procedure and reports for every key you specify.

The Quick Start Wizard allows you to name each field, one by one, by pressing the DOWN-ARROW to add a new item to the list.

Press the DOWN-ARROW, or TAB, to define the next field.

Insert	This button allows you to insert a new, blank field, above the currently selected field.
Delete	This button allows you to delete the currently selected field.
Move Up	This button allows you to move the currently selected field up one position in the fields list.
Move Down	This button allows you to move the currently selected field down one position in the fields list.

5. When you have defined all the desired fields, press the **OK** button.

The Quick Start Wizard creates your application and displays the Application Tree.

Application Wizard

This wizard creates a complete application from an existing dictionary. It creates a Frame containing a menu with options calling all procedures it creates. It also creates Browse and Report procedures each specified file, with associated Form (Update) procedures.

To use the Application Wizard:

1. Optionally, in File Manager, choose **File ➤ Create Directory** or in Windows 95 Explorer, choose **File ➤ New ➤ Folder**, type a name and press **OK**.
2. In Clarion for Windows, choose **File ➤ New.**

   The **New** file dialog appears.
3. Select the **Application** tab.
4. Type a name for the .APP file in the **Application File** field. Don't use the **Quick Start Wizard**, uncheck the box below the file list (see *Using the Quick Start Wizard*).

   Type a legal DOS file name. Clarion automatically adds the .APP extension.

   The **Application Properties** dialog appears. This dialog allows you to define the essential files for the application.
5. Name the .DCT file the application will use in the **Dictionary File** field, or press the ellipsis (...) button to select the file in the **Select Dictionary** dialog.
6. Optionally, rename the **First** procedure or accept the default—*Main*.

   The **Application Wizard** will use the name you provide to create the starting procedure.
7. Choose the **Destination Type** from the drop down list.

   This defines the type of target file for your application. Choose from *Executable (.EXE), Library (.LIB),* or *Dynamic Link Library (.DLL)*.
8. Optionally, type a name for the application's .HLP file in the **Help File** field, or use the ellipsis (...) button to select the file in the **Open File** dialog.

   The Application Generator does not require that the .HLP file exist at this point. You can leave the field blank for now, then fill in the field later.

The Application Generator allows you to name the help topics in your application without determining that the help file exists. You are responsible for creating a .HLP file that contains the context strings and keywords that you optionally enter as HLP attributes for the various controls and dialogs.

9. Accept the default *Clarion* template in the **Application Template** field.

   The selected application template controls code generation.

10. Check the **Use Application Wizard** box to use the wizard to create a complete application based on the selected dictionary and a few answers you specify.

11. Press the **OK** button.

    The **Application Wizard** dialogs appear.

12. Answer the question(s) in each dialog, then press the **Next** button. On the last dialog, the **Finish** button is enabled. If you are satisfied with your answers, press the **Finish** button.

    The Application Wizard creates the .APP file based on the dictionary and the answers you provided, then displays the **Application Tree** dialog for your new application.

    You can control some of the wizard options in the Data Dictionary by specifying Options for Files, Fields, Keys, or relations (see *Using Wizard Options).*

Browse Procedure Wizard

This wizard creates a multi-keyed Browse Procedure from an existing dictionary file definition. The Browse Box is sorted by each key you specify. The sort order is controlled by TABs. It also creates associated Form (Update) procedures, if you specify that updates are allowed.

To use the Browse Procedure Wizard:

1. Highlight a ToDo Procedure in the **Procedure Tree** and press ENTER.

   The **Select Procedure** dialog appears.

2. Select ***Browse*** from the list of Procedure templates.

3. Check the **Use Procedure Wizard** box to use the wizard to create the procedure based on the selected dictionary file and a few answers you specify.

4. Press the **Select** button.

   The **Browse Procedure Wizard** dialogs appear.

5. Answer the question(s) in each dialog, then press the **Next** button. On the last dialog, the **Finish** button is enabled. If you are satisfied with your answers, press the **Finish** button.

   The Browse Procedure Wizard creates the procedure(s) based on the dictionary file and the answers you provided, then displays the **Procedure Properties** dialog for your new procedure.

You can control some of the wizard options in the Data Dictionary by specifying Options for Files, Fields, Keys, or relations (see *Using Wizard Options*).

Form Wizard

This wizard creates an update Form Procedure from an existing dictionary file definition.

To use the Form Procedure Wizard:

1. Highlight a ToDo Procedure in the Procedure Tree and press ENTER.

   The **Select Procedure** dialog appears.
2. Select ***Form*** from the list of Procedure templates.
3. Check the **Use Procedure Wizard** box to use the wizard to create the procedure based on the selected dictionary file and a few answers you specify.
4. Press the **Select** button.

   The **Form Procedure Wizard** dialogs appear.
5. Answer the question(s) in each dialog, then press the **Next** button. On the last dialog, the **Finish** button is enabled. If you are satisfied with your answers, press the **Finish** button.

The Form Procedure Wizard creates the procedure based on the dictionary file and the answers you provided, then displays the **Procedure Properties** dialog for your new procedure.

You can control some of the wizard options in the Data Dictionary by specifying Options for Files, Fields, Keys, or relations (see *Using Wizard Options*).

Report Wizard

This wizard creates a Report Procedure from an existing dictionary file definition.

To use the Report Procedure Wizard:

1. Highlight a ToDo Procedure in the Procedure Tree and press ENTER.

   The **Select Procedure** dialog appears.

2. Select ***Report*** from the list of Procedure templates.
3. Check the **Use Procedure Wizard** box to use the wizard to create the procedure based on the selected dictionary file and a few answers you specify.
4. Press the **Select** button.

   The **Report Procedure Wizard** dialogs appear.

5. Answer the question(s) in each dialog, then press the **Next** button. On the last dialog, the **Finish** button is enabled. If you are satisfied with your answers, press the **Finish** button.

The Report Procedure Wizard creates the procedure based on the dictionary file and the answers you provided, then displays the **Procedure Properties** dialog for your new procedure.

You can control some of the wizard options in the Data Dictionary by specifying Options for Files, Fields, Keys, or relations (see *Using Wizard Options*).

Using Wizard Options

Wizard Options in the Data Dictionary Editor provide further control of the wizard's functionality. Wizards use the Options specified for a file, field, key, or alias when creating procedures.

File Options

Do Not Auto-Populate This File
Directs the wizards to skip this file when creating primary Browse procedures or Report procedures.

User Options User Options are provided to enable you to provide information to be used by a third-party template set. User Options are comma delimited, that is, each entry is separated by a comma.

Follow the instructions provided with your add-on template set.

Alias Options

Do Not Auto-Populate This Aliased File
Directs the wizards to skip the Aliased File when creating primary Browse procedures or Report procedures.

User Options User Options are provided to enable you to provide information to be used by a third-party template set. User Options are comma delimited, that is, each entry is separated by a comma.

Follow the instructions provided with your add-on template set.

Field Options

Do Not Auto-Populate This Field
Directs the wizards to skip this field when creating Form, Browse or Report procedures.

Population Order Specifies the order in which the wizards populate fields. Choose Normal, First, or Last from the drop down list. Wizards populate in this order: all Fields specified as First, then all Fields specified as Normal, and finally all Fields specified as Last.

Form Tab Specifies the TAB onto which the wizards populate the field. Type the Caption for the TAB or select one you have previously created from the drop down list. This allows you to direct the wizard to group fields in the manner you want.

Add Extra Vertical Space Before Field Controls on Forms
Check this box to direct the wizards to add vertical space between this field's control and the one populated above it.

User Options User Options are provided to enable you to provide information to be used by a third-party template set. User Options are comma delimited, that is, each entry is separated by a comma.

Follow the instructions provided with your add-on template set.

Key Options

Do Not Auto-Populate This Key
Directs the wizards to skip this Key when creating primary Browse procedures or Report procedures.

Population Order Specifies the order in which the wizards populate keys. Choose Normal, First, or Last from the drop down list. Wizards populate in this order: all Keys specified as First, then all Keys specified as Normal, and finally all Keys specified as Last.

User Options User Options are provided to enable you to provide information to be used by a third-party template set. User Options are comma delimited, that is, each entry is separated by a comma.

Follow the instructions provided with your add-on template set.

Relation Options

User Options User Options are provided to enable you to provide information to be used by a third-party template set. User Options are comma delimited, that is, each entry is separated by a comma.

Follow the instructions provided with your add-on template set.

PROCEDURE TEMPLATES

The Clarion Template Language provides support for a component-oriented approach to development. When you create a procedure, you start with a Procedure template. This generates the source code for the procedure initialization, execution, and close down.

The Procedure template may already contain Control templates. Control templates contain the control and the executable code for maintaining it. For example, the *Browse* Procedure template is actually a generic *Window* Procedure template which contains the *BrowseBox* and *BrowseUpdateButtons* Control templates.

To duplicate the *Browse* Procedure template, you *could* start with a generic *Window* Procedure template. Then you could add a *BrowseBox* Control template, which you place inside the window just as you would a control—but this control contains the executable code necessary for supporting the browse list. Because you place it as if it were an ordinary control, you can place *multiple* browse lists in the same window. You could also add the *BrowseUpdateButtons* Control template (which contains the Insert, Change and Delete buttons). You don't have to add the Control templates one by one, the *Browse* Procedure template groups them all together.

This chapter describes the Clarion for Windows Procedure templates. It also mentions several Control templates, which are the subject of the next chapter.

Component Oriented Templates

The templates are "granularized," giving you precise control over which components you use to build your procedures.

You can use the generic *Window* Procedure template to define a window or dialog box. Most of the functionality of a Procedure template actually comes from the Control templates in it. Control templates provide both controls and the executable code to handle them.

To illustrate, consider a browse window. If you examine the *Browse* Procedure template, you find its core is simply a collection of Control templates. The following overview provides a conceptual walk-through of the steps for putting the browse window together from its components. It does not provide a precise step by step.

❏ Create a new procedure of the Window procedure type.

This Procedure template predefines a few variables which monitor that the window is actually opened and that it processes events.

The executable code within the Procedure template is limited to defining a small number of variables to keep track of window management. It generates the ACCEPT loop for the window, and then sets up the CASE statements to handle field-specific and window-specific events:

```
  ACCEPT
    CASE EVENT()
      ... (standard window handling code goes here)
    END
    CASE FIELD()
      ... (standard field event handling code goes here)
    END
 END
```

The Procedure templates also contain standard actions for any button, entry or check box and additional prompts for “following up” on the actions of the control. You access these prompts through the Actions dialogs for any of these controls.

An entry box control, for example, comes with additional prompts to help you quickly use it as a lookup field. Another example, is a check box which provides additional prompts so that you can update variables or hide/unhide controls when the end user checks or unchecks the box.

Thus, a Procedure template acts as a container which automatically provides support for additional layers of template functionality.

❑ Press the **Window** button to define a new window.

❑ Within the Window Formatter, choose **Populate ➤ Control Template**, or choose the new Control template tool from the Controls toolbox.

❑ Choose a Control template from the **Select Control Template** dialog.

This is the point where the *BrowseBox* Control template, for example, is available. You can add multiple Control templates to the same window.

This Control template generates the executable code for handling a list box control. The Control template also provides the additional prompts where you can specify its behavior. This includes access to the file schematic. The Control template then goes on to generate the executable code for operating the browse box. It creates a VIEW to get the field values from disk and a QUEUE to store the fields used for a displayable page.

❑ Select fields for the browse from the file schematic.

Note that you reference the file schematic for the control, *not* for the procedure. This is so that you can, if desired, place more than one browse box control in the same window, referencing a different file displaying from another VIEW and QUEUE. Each control, when appropriate, has its own section in the file schematic tree.

❑ Add **Insert**, **Change**, and **Delete** buttons by placing the *BrowseUpdateButtons* Control template in the window.

This Control template places three buttons in your window, one by one, and handles code generation for processing the buttons.

❑ Add a **Close** button.

Another new Control template takes care of this task.

❑ Close the Window Formatter and save the procedure.

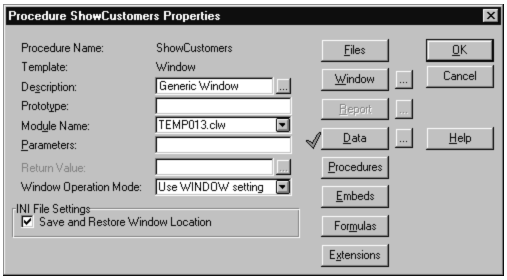
Depending on which templates—procedure and control—you add to your application, embed points appear in the **Embedded Source** dialogs, and new sets of Control templates become available as you place others. The remainder of this chapter describes the procedure and Code templates which ship with Clarion for Windows, and provides a guide to filling out the options and prompts that implement them.

The Window Template

This template functions as a blank slate, upon which you can create your own window procedure of any kind. Press the **Window** button in the **Procedure Properties** dialog to create your window.

For the controls and Control templates you place, there are field templates (referenced in the Window Procedure template) that add embed points to handle the events they generate. The **Embeds** button allows you to attach appropriate code, after you place the controls.

The only “predefined” elements of the template, which you can access through the **Procedure Properties** dialog, are local variables produced by the template which the executable code uses to pass data to and from a calling procedure. These “manage” the window and procedure, keeping track of whether the window is open, and whether the procedure needs to respond to a global event.


Procedure ShowCustomers Properties

Procedure Name: ShowCustomers
Template: Window
Description: Generic Window
Prototype:
Module Name: TEMP013.clw
Parameters:
Return Value:
Window Operation Mode: Use WINDOW setting
INI File Settings
☑ Save and Restore Window Location

The code generated by this template processes the window that you create. It contains an ACCEPT loop for the window, and a CASE structure for handling any field or window events.

The **Procedure Properties** dialog contains the following:

Files	Accesses the **File Schematic Definition** dialog.
Window	Accesses the Window Formatter. No windows are predefined. The ellipsis (...) button next to the Window button allows you to edit the WINDOW structure at the source code level.

Tip: To duplicate a window created for another application or procedure, without copying the entire procedure, copy the WINDOW declaration from the other source code document, then paste it in the editor window available for the procedure under development. Caution: do NOT do this to windows whose controls were placed by Control templates!

Data	Accesses the local data declarations for the procedure under development. Some variables are predefined in the template to handle inter-procedure communication. The ellipsis button next to the **Data** button allows you to edit the data declarations for the procedure at the source code level.

Tip: If you have a long list of local data variables, and you're a reasonably fast typist, it may be faster to enter them as text in the editor window, rather than creating them one by one in a series of Field Properties dialog boxes.

Procedures	Names a new procedure to add to the **Application Tree** dialog.

Embeds Accesses the locations in the generated source code at which you can add a Code template or your custom code. The Window Procedure template includes the standard embed points. Of particular importance to this template are the Window Event Handling and Control Event Handling embed points.

Formulas Accesses the Formula Editor. The formula's result field is then available from the field list in the **File Schematic Definition** dialog.

Extensions Accesses extension and Control templates. Extensions, if applicable to the procedure, can be added or modified from this dialog. If any Control templates were placed in the window, this accesses the prompts for those Control templates. Optionally, you can specify whether or not the prompts for an extension or Control template should display on the procedure properties window.

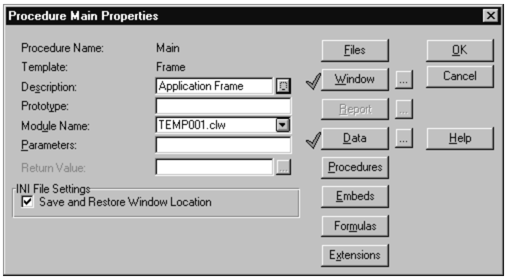
In addition to the standard buttons, the *Window* Procedure template contains two custom prompts:

Window Operation Mode
Allows you to override the window properties specified in the formatter. You can specify **Normal**, **MDI**, which adds the MDI attribute (Multiple Document Interface), or **Modal**, which adds the MODAL attribute to the WINDOW structure. You can also select **Use Window Setting**, which is the default. This specifies you *don't* want to override the **Window Properties.**

INI File Settings Specifies you want to save the window position (location) in the application's .INI file when the end user closes it. You must enable the use of the .INI file in the **Global Properties** dialog to support this.

The Frame Template

This template provides an MDI (Multiple Document Interface) parent frame, containing a predefined shell menu. The menu provides useful items such as an Exit command, plus the standard editing and window management commands.



Procedure Name: Main
Template: Frame
Description: Application Frame
Prototype:
Module Name: TEMP001.clw
Parameters:
Return Value:
INI File Settings
☑ Save and Restore Window Location

When creating an MDI application, the Frame should be the main procedure. You start new execution threads for each MDI child window which you want to appear inside the frame. The Actions buttons for a control or Menu Item provide a check box to specify the start of a new execution thread or you can use the InitiateThread Code template.

The **Procedure Properties** dialog contains the following:

Files	Accesses the **File Schematic Definition** dialog.
Window	Accesses the Window Formatter. One window is predefined. This is an MDI Parent window. Controls can be placed only in a toolbar in this window. The ellipsis (...) button next to the Window button allows you to edit the WINDOW structure at the source code level. The predefined window contains a shell menu, containing the following commands: File/Print Setup and Exit; Edit/Cut, Copy and Paste; Window/Tile/Cascade and Arrange Icons; Help/ Contents, How to Use Help, and Search for Help on. Each of the predefined menu commands implement standard (STD) behaviors. The runtime libraries handle all the functionality automatically, You don’t have to code anything for these menu commands.
Data	Access the local data declarations for the procedure under development. Some variables are predefined in the template to handle inter-procedure communication.

	The ellipsis button next to the **Data** button allows you to edit the data declarations for the procedure at the source code level.
Procedures	Names a new procedure to add to the **Application Tree** dialog.
Embeds	Accesses the locations in the generated source code at which you can add a Code template or your own custom code. The template includes the standard embed points, plus additional embeds for you to hook code in at the selection of any of the menu commands, or if the menu bar “drops” or “highlights” any of the menu commands. If you add a Toolbar, embed points are added for any control populated on the Toolbar.
Formulas	Accesses the Formula Editor. The result field of the formula you create is then available from the field list in the **File Schematic Definition** dialog.
Extensions	Accesses extension and Control templates. Extensions, if applicable to the procedure, can be added or modified from this dialog. If any Control templates were placed in the Toolbar, this accesses the prompts for those Control templates. Optionally, you can specify whether or not the prompts for an extension or Control template should display on the procedure properties window.

In addition to the normal buttons, the Frame procedure contains this additional prompt:

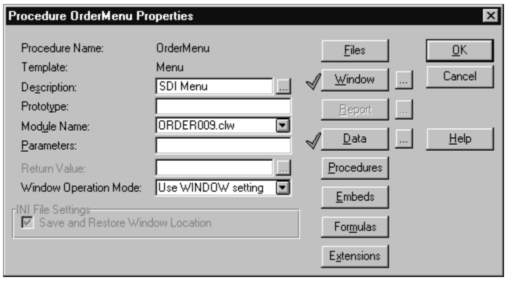
INI File Settings	Specifies you want to save the window position (location) in the application’s .INI file when the end user closes it. You must enable the use of the .INI file in the **Global Properties** dialog to support this.

The Menu Template

This template provides an SDI (Single Document Interface) window.

All prompts are identical to the MDI Frame template. The predefined window contains only a single menu (File), containing a single command (Exit).

Files	Accesses the **File Schematic Definition** dialog.
Window	Accesses the Window Formatter. One window is predefined. Unlike the Application Frame template, the window is a normal window, in which you can place controls. The ellipsis (...) button next to the Window button allows you to edit the WINDOW structure at the source code level. The predefined window contains only a single menu (File), containing a single command (Exit).



Procedure Name:	OrderMenu
Template:	Menu
Description:	SDI Menu
Prototype:	
Module Name:	ORDER009.clw
Parameters:	
Return Value:	
Window Operation Mode:	Use WINDOW setting

INI File Settings
☑ Save and Restore Window Location

Data	Access the local data declarations for the procedure under development. Some variables are predefined in the template to handle inter-procedure communication. The ellipsis (...) button next to the **Data** button allows you to edit the data declarations for the procedure at the source code level.
Procedures	Names a new procedure to add to the **Application Tree** dialog.

Embeds	Accesses the locations in the generated source code at which you can add your custom code. The template includes the standard embed points, plus additional embeds for you to hook code in at the selection of any of the menu commands, or if the menu bar “drops” or “highlights” any of the menu commands. If you place controls in the window, additional embed points are added for each control.
Formulas	Accesses the Formula Editor. The result field of the formula you create is then available from the field list in the **File Schematic Definition** dialog.
Extensions	Accesses extension and Control templates. Extensions, if applicable to the procedure, can be added or modified from this dialog. If any Control templates were placed in the window, this accesses the prompts for those Control templates. Optionally, you can specify whether or not the prompts for an extension or Control template should display on the procedure properties window.
INI File Settings	Specifies you want to save the window position (location) in the application’s .INI file when the end user closes it. You must enable the use of the .INI file in the **Global Properties** dialog to support this.

The Source Template

The Source Procedure template provides an elegant and simple way to add hand code to your application. It provides two points at which to embed your code: the data section, and the code section. The template also makes available additional tools—the Formula Editor and **File Schematic Definition** dialog—for your convenience.

The template simply declares the procedure, handles any optional parameters, places the embedded data declarations in the data section, begins the CODE section, then places any embedded executable code in the CODE section:

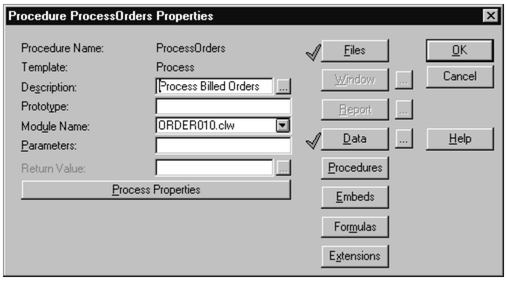
```
... (local data)
CODE
... (your embedded code)
```

The **Procedure Properties** dialog contains the following:

Files	Accesses the **File Schematic Definition** dialog.
Data	Accesses the local data declarations for the procedure under development. The ellipsis (...) button next to the **Data** button allows you to edit the data declarations for the procedure at the source code level.
Procedures	Names a new procedure to add to the Application Tree.
Embeds	Calls the **Embedded Source** dialog. The template defines two embed points: Data Section, and Processed Code. When you select the Data Section, you can select SOURCE in the **Select Embed Type** dialog. You may then add variables or data structures with the editor. When you select **Processed Code**, you may also specify that the embed point call a procedure, or choose a Code template or write source code to add.
Formulas	Accesses the **Formula Editor**. The result field of the formula you create is then available from the field list in the **File Schematic Definition** dialog.
Extensions	Accesses extension and Control templates. Extensions, if applicable to the procedure, can be added or modified from this dialog. Optionally, you can specify whether or not the prompts for an Extension template should display on the Procedure Properties window.

The Process Template

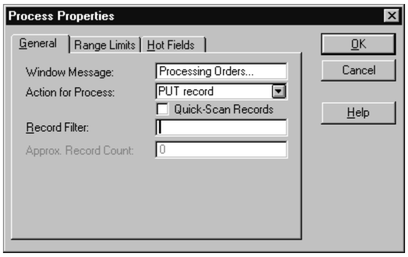
The Process Procedure template reads through a data file and performs an operation on each record. You can specify a filter or range of records to perform the operation on. A predefined window contains a progress indicator to show the end user what percentage of the operation is complete.



Procedure Name: ProcessOrders
Template: Process
Description: Process Billed Orders
Prototype:
Module Name: ORDER010.clw
Parameters:
Return Value:

Process Properties

This button provides access to the properties of the Process.



Window Message: Processing Orders...
Action for Process: PUT record
☐ Quick-Scan Records
Record Filter:
Approx. Record Count: 0

General

Window Message Text to display on the Process Status Window.

Action for Process

This prompt allows you to specify that the process operation changes (PUTs) or deletes the records that it processes. You can attach code to the ***Activity for each Record*** embed point which the Control template adds.

Quick-Scan Records	Specifies buffered access behavior for ODBC, ASCII, DOS, or BASIC files. These file drivers read a buffer at a time (not a record), allowing for fast access. In a multi-user environment these buffers are not 100% trustworthy for subsequent access, because another user may change the file between accesses. As a safeguard, the driver rereads the buffers before each record access. To disable the reread, enable QUICKSCAN.
Record Filter	Type an expression to limit the process to only those records which match the filter expression. This filters all displayable records. When a Record filter is used in conjunction with a Range Limit, only those records within the specified range are considered.
Approx Record Count	This number is used by the progress dialog which appears during the process.

Range Limits

This tab is only available if you specify a Key for the File in the File Schematic.

Range Limit Field	Type in the field name or press the ellipsis (...) button to select the field from the Component list. The Range Limit Field must be a component of the Access Key specified in the File Schematic dialog. The range limit is key-dependent; the generated source code uses the SET statement to find the first valid record.
Range Limit Type	When a field is selected for the Range Limit Field, this specifies the method of determining the records for inclusion in the list box. **Current Value**—Signifies the value contained in the key field at the beginning of the ACCEPT loop. This is the value used for the range for the duration of the procedure.

Single Value—Specifies a variable containing the limiting value. Only records matching the variable are included. Enter a variable in the Range Limit Value box which appears, or press the ellipses (...) button to select the variable from the File Schematic.

Range of Values—Allows you to specify upper and lower limits. Enter a variable in the Low Limit and High Limit Value boxes which appears, or press the ellipsis (...) button to select the variables from the File Schematic.

Hot Fields

When you select the Hot Fields tab, you can select a field (or fields) to add to the VIEW. When scrolling through the file, the generated source code reads the data for these fields from the VIEW, rather than from the disk. This optimizes performance. Elements of the Primary Key and the current key are always included in the VIEW, so they do not need to be inserted in the Hot Field list. Any field used in a computation or filter must be in the VIEW.

In addition, you can BIND fields through this dialog. You must BIND any field used in a filter.

The External Template

The External Procedure template declares a procedure contained in an external library (*.LIB only) or object file. The Application Generator writes no source code. The project system links in the external file as a module.

After selecting the External template type from the **Select Procedure Type** dialog, choose OBJ or LIB from the **Select Module Type** dialog.

In the **Module Name** field, select the file name of the external library or object file from the drop down list . Only those external modules already included in the project appear, so if your module does not appear, add the new module first. To add the module in the Application Generator, choose **Application ➤ Insert Module**.

Optionally type parameter declarations in the **Prototype** field.

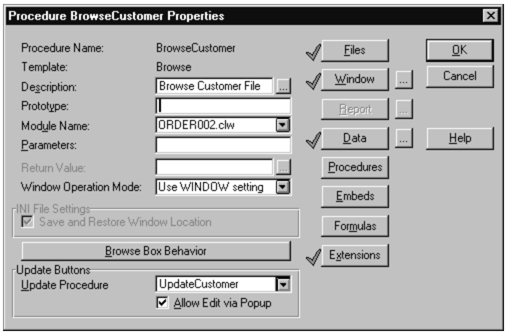
The Browse Template

The Browse Template, as noted in the introduction to the chapter, consists of several Control templates which add a list box, update buttons, a Select button and a Close button to the default window. The Control templates also add the Browse Box and Update Buttons prompts to the **Procedure Properties** dialog.

Additionally, the generated code includes a ROUTINE called RefreshWindow which implements the lookups of related records, updating list boxes where necessary, and gets the current data to the controls in the window. Additional code actually updates the controls, i.e., refreshing it for the “screen.”

The **Procedure Properties** dialog contains the following options:

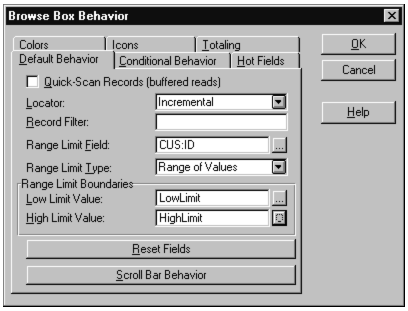
Files	Accesses the **File Schematic Definition** dialog. You can name the FILE structure(s) within the **Procedure Properties** dialog. **The File Schematic Definition** dialog automatically attaches your choices to the list box browse control.
Window	Accesses the Window Formatter. One window is predefined. The ellipsis (...) button next to the **Window** button allows you to edit the WINDOW structure at the source code level.
Data	Accesses the local data declarations for the procedure under development. Some variables are predefined in the template to handle inter-procedure communication. The ellipsis (...) button next to the **Data** button allows you to edit the data declarations for the procedure at the source code level.


Procedure BrowseCustomer Properties

Procedure Name: BrowseCustomer
Template: Browse
Description: Browse Customer File
Prototype:
Module Name: ORDER002.clw
Parameters:
Return Value:
Window Operation Mode: Use WINDOW setting
INI File Settings
☑ Save and Restore Window Location
Browse Box Behavior
Update Buttons
Update Procedure: UpdateCustomer
☑ Allow Edit via Popup

Procedures	Names a new procedure to add to the **Application Tree**.
Embeds	Accesses the locations in the generated source code at which you can add a Code template or your own custom code.
Formulas	Accesses the **Formula Editor**. The result field of the formula you create is then available from the field list in the **File Schematic Definition** dialog.
Extensions	Accesses extension and Control templates. Extensions, if applicable to the procedure, can be added or modified from this dialog. If any Control templates were placed in the window, this accesses the prompts for those Control templates. Optionally, you can specify whether or not the prompts for an extension or Control template should display on the procedure properties window.

The Browse Box Control template adds a button—**Browse Box Behavior**— to the **Procedure Properties** dialog. This provides access to a tabbed dialog where you can specify options for the Browse Box Control.



Browse Box Behavior

Colors | Icons | Totaling | Default Behavior | Conditional Behavior | Hot Fields

Quick-Scan Records (buffered reads) [ ]

Locator: Incremental

Record Filter:

Range Limit Field: CUS:ID

Range Limit Type: Range of Values

Range Limit Boundaries

Low Limit Value: LowLimit

High Limit Value: HighLimit

Default Behavior

This tab contains the prompts that control the default behavior of the Browse Box Control.

Quick-Scan Records
Specifies buffered access behavior for ODBC, ASCII, DOS, or BASIC files. These file drivers read a buffer at a time (not a record), allowing for fast access. In a multi-user environment these buffers are not 100% trustworthy for subsequent access, because another user may change the file between accesses. As a safeguard, the driver rereads the buffers before each record access. To disable the reread, enable QUICKSCAN.

Locator A locator is a screen entry field that updates a component of the primary file access key. When the end user types a character(s) in the entry box, then presses TAB, the list box updates to show the closest matching record. This is disabled when browsing a file in Record Order (without specifying a KEY in the File Schematic)

Choose **Step** for a list box which, when the user types in a character, advances the selection to the nearest match in the key field. Retyping the same character advances to the next occurrence of a field beginning with that character.

Choose **Entry** for an entry box to hold the value for the locator. When the end user places a value in the entry box, TAB or reselecting the list box will move the selection to the nearest matching record.

	Choose **Incremental** for a locator which accepts multiple characters and moves the selection to the nearest matching record.
	Choose **None** for no locator.
Record Filter	Type an expression to limit the contents of the browse list to only those records matching the filter expression. The filter is loops through all displayable records to select those that meet the filter. You must BIND any file field that is used in a filter expression. The Hot Fields TAB enables you to BIND fields.
Range Limit Field	In conjunction with the **Range Limit Type**, specifies a record or group of records for inclusion in the list box. Choose a field by pressing the ellipsis (...) button. The range limit is key-dependent; the generated source code uses the SET statement to find the first valid record. This is only available if you specify a Key for the File in the File Schematic.
Range Limit Type	When a field is selected for **Range Limit Field**, specifies a record or group of records for inclusion in the list box.
	Current Value signifies the current value of **Range Limit Field**.
	Single Value allows you to limit the list to a single value. Specify the variable containing that value in the **Range Limit Value** box which appears.
	Range of Values allows you to specify upper and lower limits. Specify the variable containing the values in the **Low Limit** and **High Limit Value** boxes.
	File Relationship allows you to choose a range limiting file from a 1:MANY relationship. This limits the browse to display only those child records matching the current record in the Parent file. For example, if your browse was a list of Orders, you could limit the display to only those orders for the current Customer (in the Customer file).

Reset Fields button

Pressing this button displays a list box allowing you to add Reset Fields. If the value of any field in the Reset Fields list changes the Browse Box is refreshed.

Scroll Bar Behavior button

Pressing this button displays a dialog where you can define the way a scroll bar works.

Scroll Bar Behavior

Specifies the manner the scroll bar works. Choose ***Fixed Thumb*** or ***Movable Thumb*** from the drop down list.

Key Distribution

This specifies the distribution of the points of the scroll bar. Choose one of the two predefined distributions (Alpha or Last Names), or Custom, or Runtime from the drop down list.

Alpha defines 100 evenly distributed points alphabetically.

Last Names defines 100 points distributed as last names are commonly found in the United States. If the access key is sorted on numeric data, you should a custom or runtime distribution.

Custom allows you to define your own points.

Runtime reads the first and last record and computes the values for 100 evenly distributed break points in between.

Custom Key Distribution

Allows you to specify the break points for distribution along the scroll bar (useful when you have data with a skewed distribution). Insert the values for each point in the list. String constants should be in single quotes (' ').

Runtime Distribution Parameters	Allows you to specify the type of characters considered when determining the distribution points. This is only appropriate when the Free Key Element is a STRING or CSTRING. Check the boxes for the types of characters you wish to include for consideration.

Conditional Behavior

This tab contains a list box that allows you to define specific behavior based on conditions. Add conditions to the list by pressing the Insert button. This displays a dialog where you define the Condition and the desired behavior when that condition is true.

At runtime these conditions are evaluated, and the behavior for the first true condition in the list is used.

In this dialog you can specify:

Condition	Any valid expression.
Key to Use	Optionally, the Key to use to determine the sort order of the browse box when this condition is true.

The remaining fields and buttons are the same as the Default behavior tab.

Hot Fields

When you select the Hot Fields tab, you can select a field (or fields) to keep “live” in the QUEUE. When scrolling through the file, the generated source code reads the data for these fields from the QUEUE, rather than from the disk. This speeds up list box updates.

Specifying "Hot" fields also allows you to place file field controls outside of the Browse Box that are updated whenever a different record is selected in the list box. Elements of the Primary Key and the current key are always included in the QUEUE, so they do not need to be inserted in the Hot Field list.

This dialog also enables you to BIND a field. You must BIND any file field that is used in a filter expression or as a field to total.

Totaling

This TAB contains a list box that allows you to define total fields for a browse box. Press the Insert button to add total fields.

This displays a dialog where you can define total fields for a *BrowseBox* Control.

Total Target Field	The variable to store the total. This can be a local, module, or global variable. You may also use a file field; however, you must write the code to update the data file.
Total Type	Choose ***Count, Sum,*** or ***Average*** from the drop down list. **Count** tallies the number of records. **Sum** adds the values of the Field to Total. **Average** determines the arithmetic mean of the Field to Total.
Field to Total	The field to be summed or averaged. This box is disabled when the total field is a **Count** Type.
Total Based On	Choose ***Each Record Read*** or ***Specified Condition*** from the drop down list. This specifies whether to consider every record or only those that meet a certain filter criteria.
Total Condition	The condition to meet when using a Total based on a specified condition. You can use any valid expression.

Colors

This tab is only available if you check the Color Cells box in the List Box Formatter. It displays a list of the fields which have been specified to allow colorization.

To specify colors, highlight the desired field and press the **Properties** button.

Customize Colors

This dialog allows you to specify the default colors for Normal Foreground and Background; and for the Foreground and Background colors to display when the row is selected.

Below the default colors section, is the Conditional Color Assignments list. To add a condition and specify special colors to display for the field when the condition is true, press the Insert Button.

At runtime these conditions are evaluated, and the colors for the first true condition in the list are used.

Icons

This TAB is only available if you check the Icons box in the List Box Formatter. It displays a list of the fields which have been specified to allow Icon display. To specify Icons, highlight the desired field and press the Properties button.

Customize BrowseBox Icons

This dialog allows you to specify the default Icon for the field.

Default Icon	The default icon to display. You may specify a standard Icon or an Icon (.ICO) file on disk.
Conditional Icon Usage	Below the default Icon section, is the Conditional Icon Usage list. To add a condition and specify special Icons to display when the condition is true, press the Insert Button. At runtime these conditions are evaluated, and the Icon for the first true condition in the list is used.

Update Buttons Control template

The Update Buttons Control template adds a prompt for naming or choosing the update procedure.

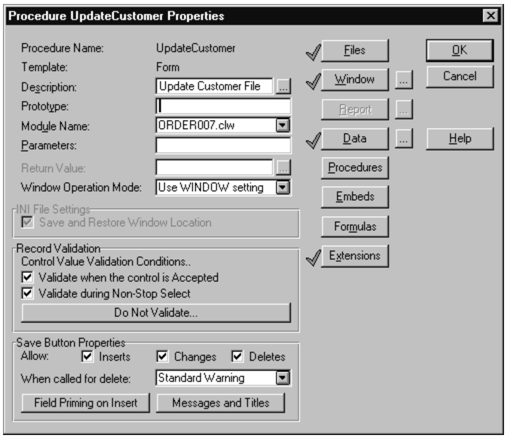
Update Procedure	Type the name of a new procedure, or choose an existing procedure from the drop-down list. If you name a new procedure, the Application Generator automatically adds it to the **Application Tree**.
Allow Edit via Popup	Check this box to enable a popup menu when the user right-clicks on the List Box. The popup menu calls the update procedure to Insert, Change or Delete a record.

The Form Template

The Form Template provides a predefined window, with update buttons, plus an action message text control. See the section explaining the Update Buttons Control template for details on setting options for the Actions dialogs for the included Control templates.

The Form Template **Procedure Properties** dialog contains the following items:

Files Accesses the **File Schematic Definition** dialog. You can name the FILE structure(s) within the **Procedure Properties** dialog. **The File Schematic Definition** dialog automatically attaches your choices to the file I/O controls.



Procedure Name: UpdateCustomer
Template: Form
Description: Update Customer File
Prototype:
Module Name: ORDER007.clw
Parameters:
Return Value:
Window Operation Mode: Use WINDOW setting

INI File Settings
☑ Save and Restore Window Location

Record Validation
Control Value Validation Conditions..
☑ Validate when the control is Accepted
☑ Validate during Non-Stop Select
Do Not Validate...

Save Button Properties
Allow: ☑ Inserts ☑ Changes ☑ Deletes
When called for delete: Standard Warning
Field Priming on Insert
Messages and Titles

Window Accesses the Window Formatter. One window is predefined, containing an Action Message control, and the Update Buttons Control template.

The ellipsis (...) button next to the Window button allows you to edit the WINDOW structure at the source code level.

Data Accesses the local data declarations for the procedure under development. Some variables are predefined in the template to handle inter-procedure communication.

The ellipsis (...) button next to the **Data** button allows you to edit the data declarations for the procedure at the source code level.

Procedures Names a new procedure to add to the **Application Tree** dialog.

Embeds Accesses the locations in the generated source code at which you can add a Code template or your own custom code.

Formulas Accesses the Formula Editor. The result field of the formula you create is then available from the field list in the **File Schematic Definition** dialog.

Extensions Accesses extension and Control templates. Extensions, if applicable to the procedure, can be added or modified from this dialog. If any Control templates were placed in the window, this accesses the prompts for those Control templates. Optionally, you can specify whether or not the prompts for an extension or Control template should display on the procedure properties window.

In addition to the normal buttons, the Form procedure contains additional prompts:

INI Settings Specifies you want to save the window position (location) in the application's .INI file when the end user closes it. You must enable the use of the .INI file in the **Global Properties** dialog to support this.

The SaveButton Control template adds these prompts:

Allow: Check the appropriate boxes for permitted operations: **Inserts, Changes, or Deletes.**

When called for Delete

Allows you to specify what displays when this procedure is called to delete a record. **Standard Warning** displays a message box prompting for confirmation of the delete. **Show Form** displays the form. **Automatic Delete** allows records to be deleted without a display or prompt for confirmation.

Access a dialog to specify display messages and their locations by pressing the **Messages and Titles** button.

Insert Message Specifies the text for the action message when the procedure is called to add a record. The default text is “Record will be added.”

Change Message Specifies the text for the action message when the procedure is called to change a record. The default text is “Record will be Changed.”

Delete Message Specifies the text for the action message when the procedure is called to delete a record. The default text is “Record will be deleted.”

On Aborted add/change
Specifies the desired behavior when an update is aborted. The choices are *Offer to save changes, Confirm cancel, or Cancel without confirming.*

Location of Message
Specifies where the message displays. Choose **None/Window Control** to display the message in a control. Choose **Title Bar** or **Status Bar** to display the message in on of those areas. Optionally specify which area of the status bar in the **Status Bar Section** box.

Display Record Identifier on the Title Bar
Allows you to append a string to the caption on the Title bar.

Record Identifier Specifies the string to append to the Title bar caption, which you can use to identify the record. Type a string in the Record Identifier box. To use a variable name, precede it with an exclamation point (!).

Access a dialog to specify fields to initialize by pressing the **Field Priming** button. "Field Priming" allows you to provide a default data value for fields in a new record. This value supersedes any initial value specified in the data dictionary. You can select a field and set an initial value in the Field Priming dialog.

The ValidateRecord extension template adds prompts for **Control Value Validation Conditions.**

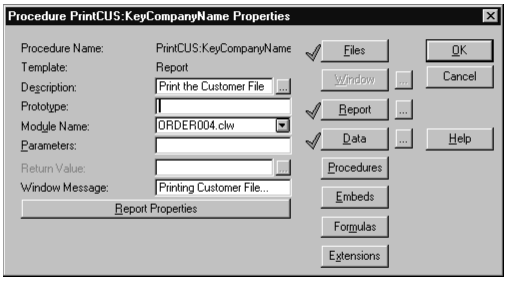
Validate when the control is accepted
Specifies that validity checking occurs when the control generates an EVENT:Accepted, which occurs when the user completes or moves focus from the field.

Validate during NonStop Select
Specifies that validity checking occurs when any control value changes if the window is in AcceptAll (Non-Stop) mode and has focus.

Do Not Validate Opens the **Do Not Validate...** dialog, which allows you to insert fields into a list; these will be excluded from validity checks.

The Report Template

Press the **Report** button in the **Procedure Properties** dialog to create your report. The Procedure template includes a window to show the progress of the report processing. The **Procedure Properties** dialog also includes a checkbox to specify whether you wish to generate a print preview function for your report.


Procedure PrintCUS:KeyCompanyName Properties

Procedure Name:	PrintCUS:KeyCompanyName
Template:	Report
Description:	Print the Customer File
Prototype:	
Module Name:	ORDER004.clw
Parameters:	
Return Value:	
Window Message:	Printing Customer File...

The **Procedure Properties** dialog contains the following:

Files Accesses the **File Schematic Definition** dialog.

Report Accesses the Report Formatter. No report is predefined.

The ellipsis (...) button next to the Report button allows you to edit the REPORT structure at the source code level.

Data Accesses the local data declarations for the procedure under development. Some variables are predefined in the template to handle inter-procedure communication.

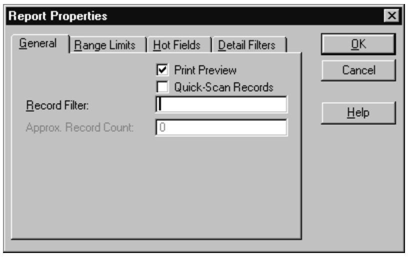
The ellipsis (...) button next to the **Data** button allows you to edit the data declarations for the procedure at the source code level.

Procedures	Names a new procedure to add to the **Application Tree** dialog.
Embeds	Accesses the locations in the generated source code at which you can add a Code template or your custom code.
Formulas	Accesses the Formula Editor. The result field of the formula you create is then available from the field list in the **File Schematic Definition** dialog.
Extensions	Accesses extension and Control templates. Extensions, if applicable to the procedure, can be added or modified from this dialog. If any Control templates were placed on the report, this accesses the prompts for those Control templates. Optionally, you can specify whether or not the prompts for an extension or Control template should display on the procedure properties window.

The Report Procedure template reads through a data file and updates the controls in the report DETAIL for each record. You can specify a filter or range of records to perform the operation on. The predefined window contains a progress indicator to show the end user what percentage of the operation is complete.

Window Message	Specifies the message to display in the progress window.

In addition to the normal buttons, the procedure contains a button for Report properties. This allows access to a dialog where you specify options for the report.


Report Properties

[✓] Print Preview
[ ] Quick-Scan Records
Record Filter:
Approx. Record Count: 0

General

Print Preview	When checked, the **Print Preview** checkbox specifies that the end user sees the report in preview mode before printing. The end user can then print the report, or cancel it.
Quick-Scan Records	Specifies buffered access behavior for ODBC, ASCII, DOS, or BASIC files. These file drivers read a buffer at a time (not a record), allowing for fast access. In a multi-user environment these buffers are not 100% trustworthy for subsequent access, because another user may change the file between accesses. As a safeguard, the driver rereads the buffers before each record access. To disable the reread, enable QUICKSCAN.
Record Filter	Type an expression to limit the report to only those records which match the filter expression. This filters all displayable records. When a Record filter is used in conjunction with a Range Limit, only those records within the specified range are considered.
Approx. Record Count	Type an approximate number of records you expect the report to process. The progress indicator uses this value.

Range Limits

This tab is only available if you specify a Key for the File in the File Schematic.

Range Limit Field	Type in the field name or press the ellipsis (...) button to select the field from the Component list. The Range Limit Field must be a component of the Access Key specified in the File Schematic dialog. The range limit is key-dependent; the generated source code uses the SET statement to find the first valid record.
Range Limit Type	When a field is selected for Range Limit Field, this specifies the method of determining the records for inclusion in the report.

Current Value—Signifies the value contained in the key field at the beginning of the ACCEPT loop. This is the value used for the range for the duration of the procedure.

Single Value—Specifies a variable containing the limiting value. Only records matching the variable are included. Enter a variable in the Range Limit Value box which appears, or press the ellipses (...) button to select the variable from the File Schematic.

Range of Values—Allows you to specify upper and lower limits. Enter a variable in the Low Limit and High Limit Value boxes which appears, or press the ellipses (...) button to select the variables from the File Schematic.

Hot Fields

When you select the Hot Fields tab, you can select a field (or fields) to add to the VIEW. When scrolling through the file, the generated source code reads the data for these fields from the VIEW, rather than from the disk. This optimizes performance. Elements of the Primary Key and the current key are always included in the VIEW, so they do not need to be inserted in the Hot Field list. Any field used in a computation or filter must be in the VIEW.

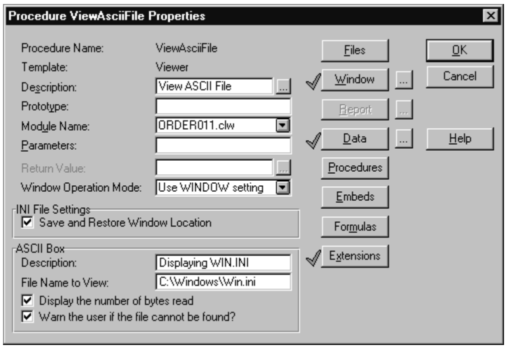
In addition, you can BIND fields through this dialog. You must BIND any field used in a filter.

Detail Filters

Select this tab to specify conditional print filters for detail bands. This enables you to suppress printing of a Detail band unless the filter expression is true.

The Viewer Template

The Viewer Template provides a predefined window, with a list box for viewing the contents of an ASCII text file, and a Close button. If you wish to use the template to view the same ASCII file all the time, you can use it as is; to add the ability to let the end user choose to view any ASCII file selected from a standard file dialog, you'll probably add an entry box to hold a variable naming the file, and the DOS File Lookup Control template.



Procedure Name: ViewAsciiFile
Template: Viewer
Description: View ASCII File
Prototype:
Module Name: ORDER011.clw
Parameters:
Return Value:
Window Operation Mode: Use WINDOW setting
INI File Settings
☑ Save and Restore Window Location
ASCII Box
Description: Displaying WIN.INI
File Name to View: C:\Windows\Win.ini
☑ Display the number of bytes read
☑ Warn the user if the file cannot be found?

The Viewer Template **Procedure Properties** dialog contains the following items:

Files	Accesses the **File Schematic Definition** dialog.
Window	Accesses the Window Formatter. One window is predefined, containing a list box control, an ASCIISearch button, an ASCIIPrint button, and a Close button.

The ellipsis (...) button next to the **Window** button allows you to edit the WINDOW structure at the source code level.

The **List Properties** dialog shows that a QUEUE called Queue:ASCII fills the list box. The Procedure template generates the source code for filling the QUEUE.

To quickly add an entry box to hold a file name for viewing, choose **Populate ➤ Field**, or use the Populate tool in the Controls toolbox. In the **Select Field** dialog, choose the ASCIIFileName variable from the Global Data section. This variable is added by the Procedure template.

To add an ellipsis (...) button, which then allows the end user to pick a file from a standard file dialog, choose the Control template tool from the Controls toolbox, click in the window, then choose the DOS File Lookup Control template. See the following chapter for further information.

To set up the viewer so that the end user has no choice of file, do not add additional controls or Control templates, but instead assign the desired file name to the ASCIIFileName variable, which is a global variable added by the Procedure template.

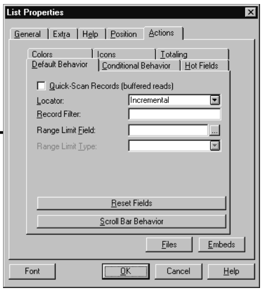
Data	Accesses the local data declarations for the procedure under development. Some variables are predefined in the template to handle inter-procedure communication. The ellipsis (...) button next to the **Data** button allows you to edit the data declarations for the procedure at the source code level.
Procedures	Names a new procedure to add to the **Application Tree** dialog.
Embeds	Accesses the locations in the generated source code at which you can add your custom code. The template includes only the standard embed points.
Formulas	Accesses the Formula Editor. The result field of the formula you create is then available from the field list in the **File Schematic Definition** dialog.
Extensions	Accesses extension and Control templates. Extensions, if applicable to the procedure, can be added or modified from this dialog. If any Control templates were placed in the window, this accesses the prompts for those Control templates. Optionally, you can specify whether or not the prompts for an extension or Control template should display on the procedure properties window.

INI Settings	Specifies you want to save the window position (location) in the application’s .INI file when the end user closes it. You must enable the use of the .INI file in the **Global Properties** dialog to support this.
ASCII Box Control	Allows you to add a description and file name (for the text file to view). Be sure to place a variable name, preceded by an exclamation point (!) when you wish to allow the end user to choose a file to view. Optionally check the **Warn if No File** box to display a message at runtime if no file is selected for viewing. Optionally check the **Display Number of Bytes read** to display the file’s size.

USING CONTROL, CODE, AND EXTENSION TEMPLATES

Contents

In this chapter, you will learn how to use the control templates, whichconsist of predefined window controls, plus the code for creating, maintaining, and integrating them with each other.



☐ Quick-Scan Records (buffered reads)

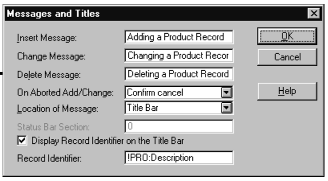
Locator: Incremental

Record Filter:

Range Limit Field:

Range Limit Type:

The BrowseBox Control template allows you to quickly add a browse list box to a window. You can easily add additional BrowseBox controls and synchronize them



Insert Message: Adding a Product Record

Change Message: Changing a Product Recor

Delete Message: Deleting a Product Record

On Aborted Add/Change: Confirm cancel

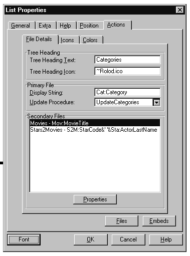
Location of Message: Title Bar

Status Bar Section: 0

☑ Display Record Identifier on the Title Bar

Record Identifier: !PRO:Description

The SaveButton Control template controls file I/O. It allows you to specify the behavior of an update procedure and the messages the user sees.



Tree Heading

Tree Heading Text: Categories

Tree Heading Icon: ~Rolod.ico

Primary File

Display String: Cat:Category

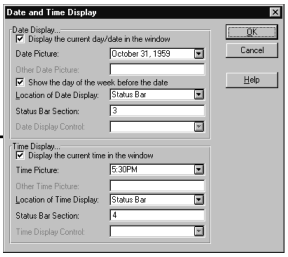
Update Procedure: UpdateCategories

Secondary Files

Movies - Mov:MovieTitle

Stars2Movies - S2M:StarCode&' '&Sta:ActorLastName

The RelationTree Control template enables access to multiple levels of a file relationship. A single *RelationTree* control can replace several *Browse-Form* pairs.



Date Display...

☑ Display the current day/date in the window

Date Picture: October 31, 1959

Other Date Picture:

☑ Show the day of the week before the date

Location of Date Display: Status Bar

Status Bar Section: 3

Date Display Control:

Time Display...

☑ Display the current time in the window

Time Picture: 5:30PM

Other Time Picture:

Location of Time Display: Status Bar

Status Bar Section: 4

Time Display Control:

The DateTimeDisplay Extension template enables you to display the time and/or date in the status bar, or a control.

The Control templates add functionality to the Procedure templates. The Control templates consist of predefined window controls, plus the code for creating, maintaining, and integrating them with each other.

For example, besides the list boxes that support a browse, mentioned in the previous chapter, other Control templates control file I/O. The generated source code can automatically prompt the user with a warning that there were changes made to the file, should the end user try to close the window before saving the changes to disk.

This chapter describes the Control templates included in Clarion for Windows, and incorporates descriptions of the prompts generated from the field templates where necessary. It also briefly describes the code and Extension templates.

ADDING CONTROL TEMPLATES

When starting with a new procedure, to add a Control template:

- ❏ Define a window, by pressing the **Window** button in the **Procedure Properties** dialog.
- ❏ In the Window Formatter, add a Control template to a window by clicking on the Control template tool in the Controls tool box.
- ❏ Choose a Control template from the **Select Control template** dialog, then place the control on the window or report by clicking on the desired location.
- ❏ A control (the type of control depends on the Control template) appears in the window.
- ❏ RIGHT-CLICK on the control, then choose **Actions** from the popup menu to access the prompts specific to the control; these define its functionality.
- ❏ Select the other tabs on the **Properties** dialog to modify a control's properties; these define its appearance, location, and functionality.

Once a Control template is added to a procedure, a check box appears next to the **Extensions** button in the **Procedure Properties** dialog. You can access the prompts added by the Control template through this button.

When starting with a Procedure template which contains a predefined windows and Control templates, you can edit the functionality and appearance of the Control templates either through the right-click popup menu in the Window Formatter or the **Extensions** buttons in the **Procedure Properties** dialog.

Control templates can also be placed in a report, depending on whether its functionality can logically be extended to "paper."

CONTROL TEMPLATES

BrowseBox

This Control template places a LIST control in a window. The LIST control's popup menu takes you to the **List Box Formatter**, so that you can choose which fields or variables populate the list, and define how they appear in the list box (including enabling colorization and Icon display). The **Actions** tab on the List Properties provides the template prompts which allows you to define the browse box's functionality, including record filters, range limits, totaling, scroll bar behavior, and locator behavior.

You can place the *BrowseBox* Control template in a window by clicking on the template control tool, then selecting ***BrowseBox - Browse List Box*** in the **Select Control template** dialog. Then CLICK in the window to place the actual list box control.

Properties

After placing the LIST control, RIGHT-CLICK on the LIST control and choose ***Properties*** from the popup menu to view the **List Properties** dialog. See the *Setting Control Properties* chapter for full information about the options available in this dialog. This section describes only the options directly affected by the BrowseBox Control template.

The template automatically defines the FROM attribute for the LIST control, which names the source (a QUEUE) for the data in the list. The standard templates name the QUEUE as Queue:Browse. The template contains a group (a template routine) that checks to see if you've applied range limits, or are using the list as a lookup. If so, it locates the correct record. The template then loads as many records into the QUEUE as will fit in the list. The QUEUE is filled from a VIEW which gets the values from fields in data files on disk.

RIGHT-CLICK on the LIST control and choose **List Box Format** from the popup menu to access the List Box Formatter to choose the fields and variables to fill the list box, and define their appearance.

The **Populate** button allows you to add a field or variable to the list box, one field or variable at a time. The **Select Field** dialog presents the file schematic. Within the schematic, each browse control appears, with a tree control marked “To Do” beneath it. To add a field from a data file defined in the dictionary:

❏ Select the “To Do” item.

❏ Press the **Insert** button

❏ Select the file from the Insert File dialog. The Browse Control item displays the name of the file.

❏ If you want to use a Key, press the **Key** button to select the key from the **Key Access** dialog. If you do not select a Key, the list is displayed in record order, which also disables the ability to set Range Limits.

❏ Select a field from the **Fields** list, which appears in the right side of the **Select Field** dialog.

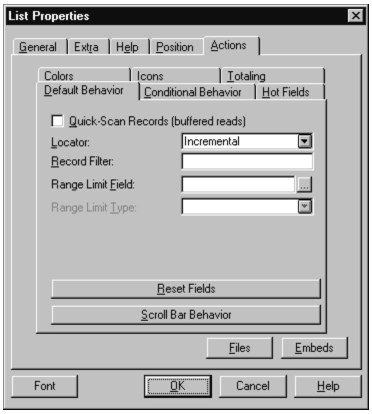
Repeat this for each field you want to add to the list box.

To add a variable to the list box, select **Global Data** or **Local Data** from the **Select Field** dialog, select the desired variable from the **Fields** list, then press the **Select** button.

After you select the file, key and field (or variable) the **List Field Properties** dialog appears. This allows you to precisely define its appearance. The *Using the List Box Formatter* chapter fully describes the options available in this dialog.

Actions

The **Actions** tab of the **List Properties** dialog (accessed by the Actions... command on the popup menu you see when you right-click the control) displays the template prompts which allows you to specify numerous template options, as well as add custom embedded source code for standard list box events, such as when the end user moves the selection bar. The dialog contains the following options:



Quick-Scan Records (buffered reads)
Locator: Incremental
Record Filter:
Range Limit Field:
Range Limit Type:

Default Behavior

This tab contains the prompts that control the default behavior of the Browse Box Control.

Quick-Scan Records
Specifies buffered access behavior for ODBC, ASCII, DOS, or BASIC files. These file drivers read a buffer at a time (not a record), allowing for fast access. In a multi-user environment these buffers are not 100% trustworthy for subsequent access, because another user may change the file between accesses. As a safeguard, the driver rereads the buffers before each record access. To disable the reread, enable QUICKSCAN.

Locator
A locator is a screen entry field that updates a component of the primary file access key. When the end user types a character(s) in the entry box, then presses TAB, the list box updates to show the closest matching record. This is disabled when browsing a file in Record Order (without specifying a KEY in the File Schematic).

Choose **Step** for a list box which, when the user types in a character, advances the selection to the nearest match in the key field. Retyping the same character advances to the next occurrence of a field beginning with that character.

Choose **Entry** for an entry box to hold the value for the locator. When the end user places a value in the entry box, TAB or reselecting the list box will move the selection to the nearest matching record.

Choose **Incremental** for a locator which accepts multiple characters and moves the selection to the nearest matching record.

Choose **None** for no locator.

Record Filter Type an expression to limit the contents of the browse list to only those records matching the filter expression. The filter is loops through all displayable records to select those that meet the filter.

You must BIND any file field that is used in a filter expression. The **Hot Fields** tab enables you to BIND fields.

Range Limit Field In conjunction with the **Range Limit Type**, specifies a record or group of records for inclusion in the list box. Choose a field by pressing the ellipsis (...) button. The range limit is key-dependent; the generated source code uses the SET statement to find the first valid record. This is enabled *only* after you specify a Key for the file associated with this control.

Range Limit Type When a field is selected for **Range Limit Field**, specifies a record or group of records for inclusion in the list box.

Current Value signifies the current value of **Range Limit Field**.

Single Value allows you to limit the list to a single value. Specify the variable containing that value in the **Range Limit Value** box which appears.

Range of Values allows you to specify upper and lower limits. Specify the variable containing the values in the **Low Limit** and **High Limit Value** boxes.

File Relationship allows you to choose a range limiting file from a 1:MANY relationship. This limits the browse to display only those child records matching the current record in the Parent file. For example, if your browse was a list of Orders, you could limit the display to only those orders for the current Customer (in the Customer file).

Reset Fields button

Pressing this button displays a list box allowing you to add Reset Fields. If the value of any field in the Reset Fields list changes the Browse Box is refreshed.

Scroll Bar Behavior button

Pressing this button displays a dialog where you can define the way a scroll bar works.

Scroll Bar Behavior

Specifies the manner the scroll bar works. Choose ***Fixed Thumb*** or ***Movable Thumb*** from the drop down list.

Key Distribution This specifies the distribution of the points of the scroll bar. Choose one of the two predefined distributions (Alpha or Last Names), or Custom, or Runtime from the drop down list.

Alpha defines 100 evenly distributed points alphabetically.

Last Names defines 100 points distributed as last names are commonly found in the United States. If the access key is sorted on numeric data, you should a custom or runtime distribution.

Custom allows you to define your own points.

Runtime reads the first and last record and computes the values for 100 evenly distributed break points in between.

Custom Key Distribution
Allows you to specify the break points for distribution along the scroll bar (useful when you have data with a skewed distribution). Insert the values for each point in the list. String constants should be in single quotes (' ').

Runtime Distribution Parameters
Allows you to specify the type of characters considered when determining the distribution points. This is only appropriate when the Free Key Element is a STRING or CSTRING. Check the boxes for the types of characters you wish to include for consideration.

Conditional Behavior

This tab contains a list box that allows you to define specific behavior based on conditions. Add conditions to the list by pressing the Insert button. This displays a dialog where you define the Condition and the desired behavior when that condition is true.

At runtime these conditions are evaluated, and the behavior for the first true condition in the list is used.

In this dialog you can specify:

Condition	Any valid expression.
Key to Use	Optionally, the Key to use to determine the sort order of the browse box when this condition is true.

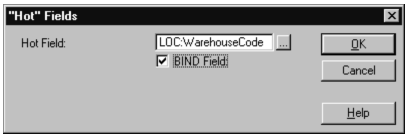
The remaining fields and buttons are the same as the Default behavior tab.

Hot Fields

When you select the Hot Fields tab, you can select a field (or fields) to keep “live” in the QUEUE. When scrolling through the file, the generated source code reads the data for these fields from the QUEUE, rather than from the disk. This speeds up list box updates.

Specifying "Hot" fields also allows you to place file field controls outside of the Browse Box that are updated whenever a different record is selected in the list box. Elements of the Primary Key and the current key are always included in the QUEUE, so they do not need to be inserted in the Hot Field list.

This dialog also enables you to BIND a field. You must BIND any file field that is used in a filter expression or as a field to total.



Hot Field: LOC:WarehouseCode
☑ BIND Field

Totaling

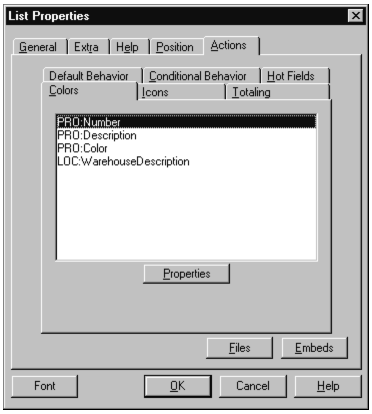
This tab contains a list box that allows you to define total fields for a browse box. Press the Insert button to add total fields.

This displays a dialog where you can define total fields for a Browse Box Control.

Total Target Field	The variable to store the total. This can be a local, module, or global variable. You may also use a file field; however, you must write the code to update the data file.
Total Type	Choose ***Count, Sum,*** or ***Average*** from the drop down list. **Count** tallies the number of records. **Sum** adds the values of the Field to Total. **Average** determines the arithmetic mean of the Field to Total.
Field to Total	The field to be summed or averaged. This box is disabled when the total field is a **Count** Type.
Total Based On	Choose ***Each Record Read*** or ***Specified Condition*** from the drop down list. This specifies whether to consider every record or only those that meet a certain filter criteria.
Total Condition	The condition to meet when using a Total based on a specified condition. You can use any valid expression.

Colors

This tab is only available if you check the **Color Cells** box in the List Box Formatter. It displays a list of the fields which have been specified to allow colorization.



PRO:Number
PRO:Description
PRO:Color
LOC:WarehouseDescription

To specify colors, highlight the desired field and press the **Properties** button.

Customize Colors

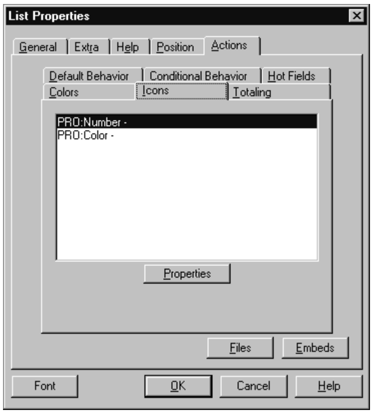
This dialog allows you to specify the default colors for Normal Foreground and Background; and for the Foreground and Background colors to display when the row is selected.

Below the default colors section, is the **Conditional Color Assignments** list. To add a condition and specify special colors to display for the field when the condition is true, press the BButton.

At runtime these conditions are evaluated, and the colors for the first true condition in the list are used.

Icons

This tab is only available if you check the **Icons** box in the List Box Formatter. It displays a list of the fields which have been specified to allow Icon display.



PRO:Number -
PRO:Color -

To specify Icons, highlight the desired field and press the Properties button.

Customize BrowseBox Icons

This dialog allows you to specify the default Icon for the field.

Default Icon	The default icon to display. You may specify a standard Icon or an Icon (.ICO) file on disk.
Conditional Icon Usage	Below the default Icon section, is the Conditional Icon Usage list. To add a condition and specify special Icons to display when the condition is true, press the Insert Button. At runtime these conditions are evaluated, and the Icon for the first true condition in the list is used.

BrowseUpdateButtons

This Control template provides a quick way to add standard functionality for managing the records in a browse list box.

The BrowseUpdateButtons Control template adds three button controls for acting upon records inside a browse box. When pressed, the buttons retrieve the appropriate record and call the procedure specified as the update procedure. Pressing the **Change, Insert,** or **Delete** button sets the variable GlobalRequest to ‘ChangeRecord,’ ‘InsertRecord,’ or ‘DeleteRecord’ respectively.

The Properties dialog for each button control is identical to the normal Button Properties dialog. See the *Setting Control Properties* chapter for a complete description.

The **Actions** tab allows you to name the update procedure and specify special keys for implementing the button actions.

Update Procedure Type a name or select from the drop down list. The Application Generator automatically adds the update procedure to the Procedure tree.

Allow Edit via Popup
Check this box to create a popup menu to call the update procedure when the end user RIGHT-CLICKS on the list box. The menu displays the text specified to display on the buttons.

When Pressed The standard set of prompts for buttons (see *Setting Control Properties)*. Normally, when using a Control template, these prompts are not used.

BrowseSelectButton

This Control template provides a quick way to choose a record from a list box when the procedure is called to select a record.

The generated source code gets the currently selected record from the list, and closes down the browse. For the end user, pressing the Select button is equivalent to double-clicking an item in the list.

The Properties dialog for the button is identical to the normal Button Properties dialog. See the *Setting Control Properties* chapter for a complete description.

The **Actions** tab contains the following:

Hide the Select button when not applicable
Specifies that the control should be hidden when the procedure is not called to select a record.

When Pressed The standard set of prompts for buttons (see *Setting Control Properties)*. Normally, when using a Control template, these prompts are not used.

SaveButton

This Control template provides an **OK** button for your window, plus the ability to display an action message for the end user. The SaveButton also handles the method of deleting records.

The Properties dialog for the Save button is the normal **Button Properties** dialog. See the *Setting Control Properties* chapter for a complete description.

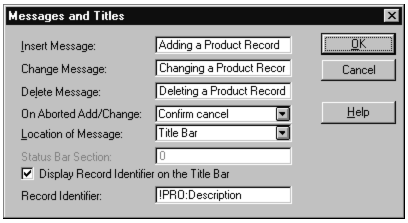
The **Actions** tab contains the following options:

Allow: Check the appropriate boxes for permitted operations: **Inserts, Changes, or Deletes.**

When called for Delete
Allows you to specify what displays when this procedure is called to delete a record. **Standard Warning** displays a message box prompting for confirmation of the delete. **Show Form** displays the form. **Automatic Delete** allows records to be deleted without a display or prompt for confirmation.

Messages and Titles

Access a dialog to specify display messages and their locations by pressing the **Messages and Titles** button.


Messages and Titles

Insert Message:	Adding a Product Record
Change Message:	Changing a Product Recor
Delete Message:	Deleting a Product Record
On Aborted Add/Change:	Confirm cancel
Location of Message:	Title Bar
Status Bar Section:	0
☑ Display Record Identifier on the Title Bar	
Record Identifier:	!PRO:Description

Insert Message Specifies the text for the action message when the procedure is called to add a record. The default text is “Record will be added.”

Change Message Specifies the text for the action message when the procedure is called to change a record. The default text is “Record will be Changed.”

Delete Message Specifies the text for the action message when the procedure is called to delete a record. The default text is “Record will be deleted.”

On Aborted Add/Change
Specifies the action to take when the user presses the Cancel button while adding or modifying a record. Choose ***Offer to save changes, Confirm Cancel,*** or ***Cancel without Confirming*** from the drop down list.

Location of Message
Specifies where the message displays. Choose ***None/Window Control*** to display the message in a control. Choose ***Title Bar*** or ***Status Bar*** to display the message in on of those areas. Optionally specify which area of the status bar in the **Status Bar Section** box.

Display Record Identifier on the Title Bar
Allows you to append a string to the caption on the Title bar.

Record Identifier
Specifies the string to append to the Title bar caption, which you can use to identify the record. Type a string in the Record Identifier box. To use a variable name, precede it with an exclamation point (!).

Field Priming

Access a dialog to specify fields to initialize by pressing the **Field Priming** button. "Field Priming" allows you to provide a default data value for fields in a new record. This value supersedes any initial value specified in the data dictionary. You can select a field and set an initial value in the Field Priming dialog.

CancelButton

This Control template primarily provides a convenient control to allow the user to close a window, and for the developer to add code to “undo” while closing down the procedure.

The generated source code posts a close window event. If differs from the CloseButton Control template (below), in that it sets the LocalResponse variable to ‘RequestCancelled’.

You can insert the executable code you need to “clean up” at an embed point.

The **Actions** tab contains the following:

When Pressed	The standard set of prompts for buttons (see *Setting Control Properties)*. Normally, when using a Control template, these prompts are not used.
When Pressed	The standard set of prompts for buttons (see *Setting Control Properties)*. Normally, when using a Control template, these prompts are not used.

CloseButton

This Control template adds a single button control marked **Close**. The generated source code closes down the current window. You specify, via the Action button, precisely what happens when the end user presses the button.

The properties dialog for CloseButton is identical to the normal Button Properties dialog. See the *Setting Control Properties* chapter for a complete description.

The **Actions** tab contains the following:

When Pressed	The standard set of prompts for buttons (see *Setting Control Properties)*. Normally, when using a Control template, these prompts are not used.

ASCIIBox

This Control template adds a list box in which you can display an ASCII (text) file. If you wish to view the same ASCII file all the time, you can specify a file name in the **Prompts** dialog.

The **Actions** tab contains the following:

Description	Allows you to add a description and to display in the progress window which displays when opening the file.
File Name to View	Specifies the path and name of the file to view, or a variable preceded by an exclamation point (!).

Display Number of Bytes Read
Check this box if you want to display the file's size in the progress dialog.

Warn the user if the file cannot be found?
Check this box if you want to display a message at runtime if the specified file cannot be found.

ASCIIPrintButton

This Control template adds a button named Print, and the underlying code necessary for printing an ASCII (text) file. Use this Control template together with the ASCII Box Control template.

Edit the **Actions** only if you wish to add another, separate action to take place *after* printing. All the code necessary for managing the print job itself is handled automatically.

The **Actions** tab contains the following:

When Pressed	The standard set of prompts for buttons (see *Setting Control Properties)*. Normally, when using a Control template, these prompts are not used.

ASCIISearchButton

This Control template adds two buttons named Find and Find Next, and the underlying code necessary for a modal search dialog, allowing the end user to find text in an ASCII (text) file. Use this Control template together with the ASCII Box Control template.

Edit the **Actions** only if you wish to add another, separate action to take place *after* the search. All the code necessary for managing the search itself is handled automatically.

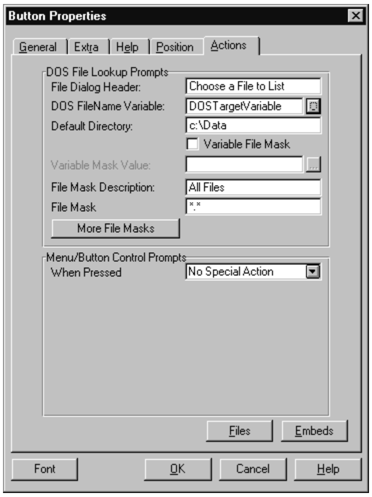
The **Actions** tab contains the following:

When Pressed	The standard set of prompts for buttons (see *Setting Control Properties)*. Normally, when using a Control template, these prompts are not used.

DOSFileLookup

This Control template adds an ellipsis (...) button which leads the end user to a standard Open File dialog. You can specify a file mask, and a return variable to hold the end user's choice.

The properties dialog for DOSFileLookup is identical to the normal Button Properties dialog. See the *Setting Control Properties* chapter for a complete description.



DOS File Lookup Prompts
File Dialog Header: Choose a File to List
DOS FileName Variable: DOSTargetVariable
Default Directory: c:\Data
☐ Variable File Mask
Variable Mask Value:
File Mask Description: All Files
File Mask *.*

Menu/Button Control Prompts
When Pressed No Special Action

The **Actions** tab contains the following:

File Dialog Header
Type the text for the caption of the **Open File** dialog.

DOS FileName Variable
Press the ellipsis (...) button to view the **File Schematic** dialog, and choose a variable to receive the end user's choice. You can also type the variable name directly into the entry box.

Default Directory Allows you to specify a directory name where the **Open File** starts.

Mask Description Type a file type description. The string appears in the drop down list in the **Open File** dialog. You can add additional masks by pressing the **More File Masks** button

File Mask	Type a file specification, such as “*.TXT” or use multiple patterns for this mask separating each with a semi-colon, such as "*.BMP;*.GIF".
More File Masks	Optionally, press this button to add additional file masks. These masks are then available to the user through the drop down list in the File Open dialog.
When Pressed	The standard set of prompts for buttons (see *Setting Control Properties)*. Normally, when using a Control template, these prompts are not used.

FileDrop

This Control template scrolls through a data file and assigns the value of the selected field to the ?Use variable. It does not allow adding records. If you want the ability to add records “on-the-fly,” use the FileDropCombo Control template.

There are two different scenarios for which you can use this Control template:

- Storing and Displaying the same data
- Displaying text data and storing a code.

Storing and Displaying the same data

In this scenario you want to select a value from the lookup file and store it in the Primary file. For example, A Product File with a field storing a color, with a lookup file of colors.

In this case, complete the prompts as follows:

Properties:

?Use	The field to which the value is assigned from the field in the lookup file.
Field to Fill From	The field from the lookup file. This value is assigned to the Target Field.
Remove Duplicates	Check this box to remove duplicates from the list displayed.

Target Field	The field to which the value is assigned from the field in the lookup file. In this case this is the same as the ?USE variable.
Record Filter	Optionally, type an expression to limit the contents of the drop down list to only those records which match the filter expression.
Default to First entry if Use Variable empty	Automatically assign the value of the first field in the list to the ?USE variable. The fields in the list are sorted alphabetically (unless you specify **Sort Fields**).

Displaying text data and storing a code

In this scenario you want to select a value from a textual field in the lookup file and store its associated code in the Primary file. For example, A Product File with a field storing a Location Code, with a lookup file of Locations. You want the user to select the Location from a list of descriptions, but store the Location Number in the Product file.

In this case, complete the prompts as follows:

?Use	Create a local variable that matches the text field.

Using the List Box Formatter, populate the list with the text field from the lookup file. It is automatically assigned to the ?Use variable.

Field to Fill From	The code field from the lookup file. This value is assigned to the Target Field.
Remove Duplicates	Check this box to remove duplicates from the list displayed.
Target Field	The field to which the value is assigned from the field in the lookup file.
Record Filter	Optionally, type an expression to limit the contents of the drop down list to only those records which match the filter expression.

Default to First entry if Use Variable empty
Automatically assign the value of the first field in the list to the ?USE variable. The fields in the list are sorted alphabetically (unless you specify **Sort Fields**).

Properties:

The List Properties for this control are the same as a list ; however, the **From** entry requires some explanation.

From: When placing a File Drop Control, this field is filled in with Queue:FileDrop. You should not modify this.

Note: You can use list box formatter to populate this control, but only the first field populated is valid for assignment.

Sort Fields

This tab allows you to add fields by which the list is sorted. The sort order is independent of Keys. Press the **Insert** button to add fields to the list. This sorts the list dynamically at runtime.

Range Limits

This tab appears *only* after you specify a Key for the file associated with this control.

Range Limit Field In conjunction with the **Range Limit Type**, specifies a record or group of records for inclusion in the list box. Choose a field by pressing the ellipsis (...) button. The range limit is key-dependent; the generated source code uses the SET statement to find the first valid record.

Range Limit Type When a field is selected for **Range Limit Field**, specifies a record or group of records for inclusion in the list box.

Current Value signifies the current value of **Range Limit Field**.

Single Value allows you to limit the list to a single value. Specify the variable containing that value in the **Range Limit Value** box which appears.

Range of Values allows you to specify upper and lower limits. Specify the variable containing the values in the **Low Limit** and **High Limit Value** boxes.

File Relationship allows you to choose a range limiting file from a 1:MANY relationship. This limits the browse to display only those child records matching the current record in the Parent file.

Colors

This tab is only available if you check the Color Cells box in the List Field Properties in the List Box Formatter. It displays a list of the fields which have been specified to allow colorization.

To specify colors, highlight the desired field and press the **Properties** button.

Customize Colors

This dialog allows you to specify the default colors for Normal Foreground and Background; and for the Foreground and Background colors to display when the row is selected.

Below the default colors section, is the Conditional Color Assignments list. To add a condition and specify special colors to display for the field when the condition is true, press the Insert Button.

At runtime these conditions are evaluated, and the colors for the first true condition in the list are used.

Icons

This tab is only available if you check the Icons box in the List Box Formatter. It displays a list of the fields which have been specified to allow Icon display. To specify Icons, highlight the desired field and press the Properties button.

Customize BrowseBox Icons

This dialog allows you to specify the default Icon for the field.

Default Icon	The default icon to display. You may specify a standard Icon or an Icon (.ICO) file on disk.

Conditional Icon Usage

Below the default Icon section, is the Conditional Icon Usage list. To add a condition and specify special Icons to display when the condition is true, press the Insert Button. At runtime these conditions are evaluated, and the Icon for the first true condition in the list is used.

FileDropCombo

This Control template scrolls through a data file and assigns the value of the selected field to the ?Use variable. It also allows adding records by typing a new value in the entry portion of the combo box.

There are two different scenarios for which you can use this Control template:

- Storing and Displaying the same data
- Displaying text data and storing a code.

Storing and Displaying the same data

In this scenario you want to select a value from the lookup file and store it in the Primary file. For example, A Product File with a field storing a color, with a lookup file of colors.

In this case, complete the prompts as follows:

Properties:

?Use	The field to which the value is assigned from the field in the lookup file.
Field to Fill From	The field from the lookup file. This value is assigned to the Target Field.
Remove Duplicates	Check this box to remove duplicates from the list.
Target Field	The field to which the value is assigned from the field in the lookup file. In this case this is the same as the ?USE variable.

Record Filter	Optionally, type an expression to limit the contents of the drop down list to only those records which match the filter expression.
Default to First entry if Use Variable empty	Automatically assign the value of the first field in the list to the ?USE variable. The fields in the list are sorted alphabetically (unless you specify Sort Fields).

Update Behavior

In this scenario, a form is NOT needed to update the lookup file. Checking the Allow Updates box enables updates directly from this control.

Displaying text data and storing a code

In this scenario, you want to select a value from a text field in the lookup file and store its associated code in the Primary file. For example, A Product File with a field storing a Location Code, with a lookup file of Locations. You want the user to select the Location from a list of descriptions, but store the Location Number in the Product file.

In this case, complete the prompts as follows:

?Use	Create a local variable that matches the textual field.

Using the List Box Formatter, populate the list with the textual field from the lookup file. It is automatically be assigned to the ?Use variable.

Field to Fill From	The code field from the lookup file. This value is assigned to the Target Field.
Remove Duplicates	Check this box to remove duplicates from the list displayed.
Target Field	The field to which the value is assigned from the field in the lookup file.
Record Filter	Optionally, type an expression to limit the contents of the drop down list to only those records which match the filter expression.

Default to First entry if Use Variable empty
Automatically assign the value of the first field in the list to the ?USE variable. The fields in the list are sorted alphabetically (unless you specify Sort Fields).

Update Behavior

In this scenario, a form is needed to update the lookup file, if you want to allow updates, specify a form procedure.

Properties:

The List Properties for this control are the same as a list; however, the **From** entry requires some explanation.

From:	When placing a File Drop Combo Control, this field is filled in with Queue:FileDropCombo. You should not modify this.

Note: You can use list box formatter to populate this control, but only the first populated is valid for the entry portion of the control.

Range Limits

This tab appears *only* after you specify a Key for the file associated with this control.

Range Limit Field	In conjunction with the **Range Limit Type**, specifies a record or group of records for inclusion in the list box. Choose a field by pressing the ellipsis (...) button. The range limit is key-dependent; the generated source code uses the SET statement to find the first valid record.
Range Limit Type	When a field is selected for **Range Limit Field**, specifies a record or group of records for inclusion in the list box. **Current Value** signifies the current value of **Range Limit Field**. **Single Value** allows you to limit the list to a single value. Specify the variable containing that value in the **Range Limit Value** box which appears.

Range of Values allows you to specify upper and lower limits. Specify the variable containing the values in the **Low Limit** and **High Limit Value** boxes.

File Relationship allows you to choose a range limiting file from a 1:MANY relationship. This limits the browse to display only those child records matching the current record in the Parent file.

Colors

This tab is only available if you check the Color Cells box in the List Field Properties in the List Box Formatter. It displays a list of the fields which have been specified to allow colorization.

To specify colors, highlight the desired field and press the **Properties** button.

Customize Colors

This dialog allows you to specify the default colors for Normal Foreground and Background; and for the Foreground and Background colors to display when the row is selected.

Below the default colors section, is the Conditional Color Assignments list. To add a condition and specify special colors to display for the field when the condition is true, press the Insert Button.

At runtime these conditions are evaluated, and the colors for the first true condition in the list are used.

Icons

This TAB is only available if you check the Icons box in the List Box Formatter. It displays a list of the fields which have been specified to allow Icon display. To specify Icons, highlight the desired field and press the Properties button.

Customize BrowseBox Icons

This dialog allows you to specify the default Icon for the field.

Default Icon	The default icon to display. You may specify a standard Icon or an Icon (.ICO) file on disk.

Conditional Icon Usage

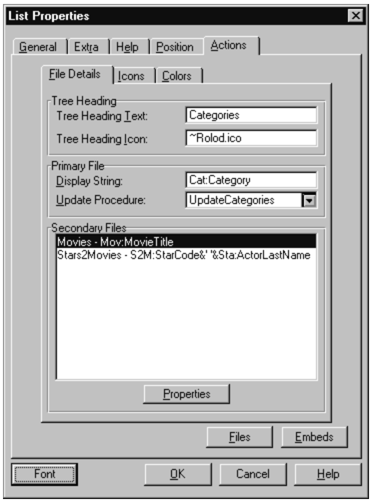
Below the default Icon section, is the Conditional Icon Usage list. To add a condition and specify special Icons to display when the condition is true, press the Insert Button. At runtime these conditions are evaluated, and the Icon for the first true condition in the list is used.

RelationTree

The tree control is actually a list box formatted to display as a tree.

Using the *RelationTree* Control template, you can specify multiple file levels to display on multiple levels of a tree control. The Relation Tree control can display an unlimited number of related files—with an associated update procedure for each level. The provides an alternative for the *Browse-Form* paradigm. A single *RelationTree* control can replace several *Browse-Form* pairs.

The *RelationTree* template employs a fully-loaded QUEUE for the root level. The child levels are demand-loaded when a branch is expanded. This template is not appropriate for databases with a very large primary file. You should use the *BrowseBox* Control template, which is page-loaded, instead.



Tree Heading
Tree Heading Text: Categories
Tree Heading Icon: ~Rolod.ico

Primary File
Display String: Cat:Category
Update Procedure: UpdateCategories

Secondary Files
Movies - Mov:MovieTitle
Stars2Movies - S2M:StarCode&' '&Sta:ActorLastName

To create a tree using the Relation Tree Control template:

1. Place a *RelationTree* Control template on a window.

   The **List Box Formatter** appears. **Do *Not* use the formatter to populate your tree.**

2. If you want to enable colorization or icon display in your tree control, press the **Properties** button on the **List Box Formatter** and check the appropriate boxes.

3. Press the **OK** button on the **List Box Formatter**.

4. RIGHT-CLICK on the Relation Tree Control template and choose **Actions** from the popup menu.

5. Press the **Files** button to specify the file schematic for the control.

6. Specify the File details:

Tree Heading Text	An optional text heading at the top of the tree. Tree Heading Text is required to enable the user to add a record at the root level.
Tree heading Icon	An optional Icon at the top of the tree. Icons must be enabled in the List Box Formatter for this prompt to be enabled.
Primary File	
Display String	The field name or text to display for the primary file level.
Update Procedure	The Update Procedure to call for this level.
Secondary Files	
	Optionally, specify display strings and Update Procedures for any secondary files by highlighting the secondary file and pressing the **Properties** button below the **Secondary Files** list box.

Calling Update Procedures

One of the most powerful features of the Relation Tree Control template is the ability to call the update procedure for the selected level of the tree (if an Update Procedure is specified for that level).

The Update Procedure is called to change a record when the user DOUBLE-CLICKS on a record.

A RIGHT-CLICK calls a popup menu to insert, change, or delete records. The menu displays the text displayed on the associated *RelationTreeUpdateButtons*.

A third method to call a update procedures is to place a *RelationTreeUpdateButtons* Control template on the window.

RelationTreeUpdateButtons

This Control template adds three buttons (**Insert**, **Change**, and **Delete**) which allow the user to call the associated update procedure for the selected level of a Relation Tree (if an update procedure has been specified) . There are no prompts for this control. The Update Procedure is specified for each level of the Relation Tree Control template.

The Change and Delete buttons correspond to the currently highlighted record. The Insert button adds a child record (the next level down the tree structure).

CODE TEMPLATES

Code templates generate executable code. The purpose is to make customization—adding embedded source code fragments that do exactly what you want it to—easier. Each Code template has one well-defined task. For example, the Initiate Thread Code template simply starts a new execution thread, and no more. Typically, the Code template provides a dialog box with options and instructions. Clarion for Windows contains the following Code templates:

InitiateThread

When opening an MDI window from an Application Frame, you must initiate an execution thread. This Code template provides an easy way to initiate a thread.

In the **Prompts for Initiate Thread** dialog, simply name the procedure that opens the MDI window.

You can optionally add a line of code to execute if the application was unable to open the thread. Type in the edit box labelled **Error Handling**. For example,

```
    BEEP; MESSAGE('Could not Start Thread','Error',ICON:HAND)
```

would beep and display a message box with the halt (hand) icon, if the thread failed to start.

You can add a procedure name to call upon an error by typing the name of the procedure in the **Error Handling** box. You would then add the procedure to the **Application Tree** with the **Insert Procedure** command.

CallProcedureAsLookup

This Code template calls a procedure to select a record. It sets a variable called RequestCompleted to advise whether the lookup was successful.

Lookup Procedure	Specifies the procedure to call.
Code before	Type in any executable code to execute before performing the lookup. Multiple statements can be used if separated by a semicolon.

Code After, Completed

Type in any executable code to execute after completing a lookup. Multiple statements can be used if separated by a semicolon.

Code After, Canceled

Type in any executable code to execute if the lookup is canceled. Multiple statements can be used if separated by a semicolon.

ControlValueValidation

This Code template gets the value of a control and matches it against the value in the key. You can add this Code template to an ENTRY, SPIN, LIST, or COMBO control; at the Accepted or Selected embed point. The code generated by this Code template gets the value in the control, then matches it against the value in the key.

It can also call a lookup procedure, to let the end user select a value. You can check whether the end user has successfully completed the lookup procedure by checking the value of the LocalResponse variable.

LookupNonRelatedRecord

This Code template is used to perform a lookup of a value based on a relationship, whether it is or is not defined in the data dictionary (Ad hoc relation). You can add this Code template to the Lookup Up Related Records embed point.

Lookup Key Type in the key name or press the ellipsis (...) button to select the key from the File Schematic.

The lookup key is used to perform the lookup into the lookup file. This *must* be a unique key. If the key is a multicomponent key, the other key elements must be primed before executing this Code template.

Lookup Field Type in the field name or press the ellipsis (...) button to select the field from the Component list.

The Lookup Field must be a component of the Lookup Key. This is the unique value within the lookup file.

Related Field	Type in the related field or press the ellipsis (...) button to select it from the File Schematic.
	The Related Field provides the unique value used to perform the lookup.

This code template generates the following code:

```
LookUpField = RelatedField       ! Move value for lookup
GET(LookUpFile,LookUpKey)        ! Get value from file
IF ERRORCODE()                   ! IF record not found
  CLEAR(LookupfileRecord)        ! Clear the record buffer
END                              ! END (IF record not found)
```

CloseCurrentWindow

This Code template simply posts an EVENT:CloseWindow, which tells the currently active window to close. There are no prompts to fill in.

EXTENSION TEMPLATES

Extension templates add functionality to procedures, but are not bound to a control or embed point. Each Extension template has one well-defined task. For example, the Date Time Display enables you to display the date and a running clock.

From a **Procedure Properties** dialog, add an Extension template by pressing the **Extensions** button.

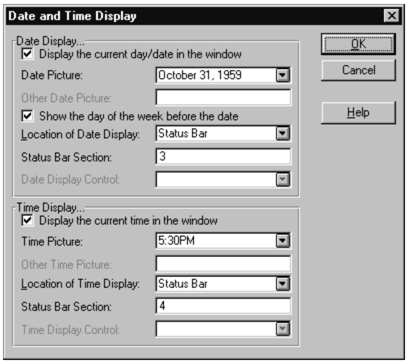
Clarion for Windows contains the following Extension templates:

DateTimeDisplay

This Extension template adds to the functionality of a Procedure template, allowing you to display the time and/or date in the status bar, or a control.

The options which appear in the Date and Time Display dialog are divided into two group boxes— Date Display and Time Display—

Display in Window	Check the box or boxes to add the display to your window.
Picture	Choose a date and/or time display picture from the drop down list. The list displays examples, such as "October 31, 1959," and "5:30P.M."
Other Picture	Type in a picture of your choice, if the picture type you wish does not appear in the list. See also: *Date Picture Tokens* or *Time Picture Tokens* in the *Language Reference.*
Day of Week	(Date only) Optionally displays the day of week.
Location	Choose between displaying the date and/or time on the status bar, or in a control.
Status Bar Section	When the Date or Time should appear on the status bar, specify the status bar section number.
Display Control	When the Date or Time should appear in a control, choose the control from a drop down list of field equate labels for the window.


Date and Time Display

Date Display...
- [✓] Display the current day/date in the window
- Date Picture: October 31, 1959
- Other Date Picture:
- [✓] Show the day of the week before the date
- Location of Date Display: Status Bar
- Status Bar Section: 3
- Date Display Control:

Time Display...
- [✓] Display the current time in the window
- Time Picture: 5:30PM
- Other Time Picture:
- Location of Time Display: Status Bar
- Status Bar Section: 4
- Time Display Control:

RecordValidation

This Extension template adds functionality to a Procedure template by enforcing data dictionary-defined control value validation. It also allows you to specify controls to exclude from validation.

Validate when the control is Accepted
Specifies that validity checking occurs when the control generates an EVENT:Accepted, which occurs when the end user completes or moves the focus from the field.

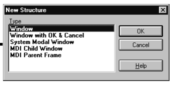
Validate during NonStop Select
Specifies that validity checking occurs when any control value changes if the window is in AcceptAll (Non-Stop) mode and has focus.

Do Not Validate Opens the Do Not Validate dialog, which allows you to select fields from a drop down list. The fields you choose will be excluded from validity checks.

USING THE WINDOW FORMATTER

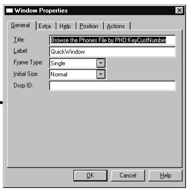
Contents

Using the Window Formatter, you visually design your application's windows and dialogs. The Window Formatter generates Clarion Language statements from the windows you design on screen; *and,* if you hand-edit the generated source code, the changes will appear on screen when you reload the Window Formatter.



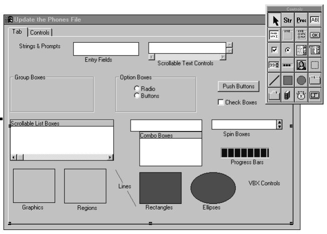
Type: Window, Window with OK & Cancel, System Modal Window, MDI Child Window, MDI Parent Frame

When you call the Window Formatter from the Application Generator or the Text Editor, you first choose the type of window to create.

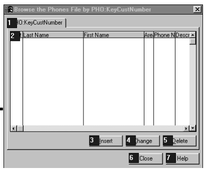


Title: Browse the Phones File by PHO:KeyCustNumber
Label: QuickWindow
Frame Type: Single
Initial Size: Normal
Drop ID:

The Window Properties dialog allows you to set important options for your window, such as whether it should have a system menu.



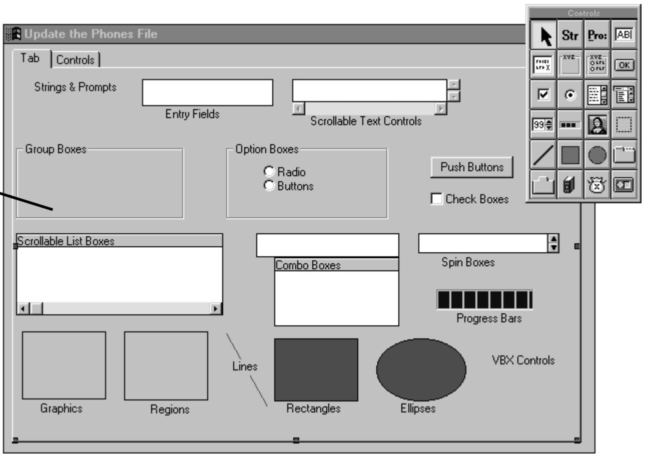
To place a control, choose a control type from the Toolbox, then click in the window under construction.



Preview your window, align its controls, set the tab order, and Snap to Grid options.

Use the **Window Formatter** to visually design Window elements—windows and controls—on screen. The **Window Formatter** automatically generates the Clarion language source code that describes these elements and the Application Generator places the generated source code at the appropriate point in your application.

The Window Formatter displays a sample window showing the controls you place in it. You can resize any control by dragging its handles.



This chapter will:

- Tell you how to use the **Window Formatter** to create a new structure or edit an existing one.
- List the types of window properties and describe how to add them to a window.
- Explain the **Preview** mode, which allows you to see the windows you create exactly as they will appear to the user.
- Detail the menus and commands available in the **Window Formatter**.
- Introduce the various window control properties dialogs you will use with the **Window Formatter**. The *Setting Control Properties* chapter discusses these in more detail.

OVERVIEW: CREATING YOUR APPLICATION'S WINDOWS

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 will display the window.

See the *Using the Application Generator* chapter for more information.

2. From the **Application Tree** dialog, display the **Procedure Properties** dialog for the procedure

   DOUBLE-CLICK the procedure name, OR RIGHT-CLICK the procedure name and select **Properties** from the popup menu, or highlight the procedure name and press the **Properties** button.

3. Optionally specify data, files, or both, that the procedure will use.

   Press the **Data** and **Files** buttons and make your choices in the dialog boxes which appear, then return to the **Procedure Properties** dialog. See the *Using the Application Generator* chapter for more information.

4. Press the **Window** button to design your window. Or, from the **Application Tree** dialog, RIGHT-CLICK the procedure name and select **Window** from the popup menu.

   If no default window is defined, the **New Structure** dialog appears, and you will have to decide what *type* of window to use. See *Choosing a Window Type* below. If a default window is already defined, the **Window Formatter** appears.

Tip: You can also 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. See WINDOW and APPLICATION in the *Language Reference* for more information.

5. From the **New Structure** dialog, select the *type* of window the procedure should display .

   Specify your choice by DOUBLE-CLICKING it, or by highlighting a choice from the **Type** list, then pressing the **OK** button. The **Window Formatter** appears.

6. Customize the window by setting its *size* and *properties*.

   See *Customizing Your Application's Windows* below.

7. Optionally, place a menu in the window using the **Menu Editor**.

   The *Creating Menus and Toolbars* chapter explains this procedure.

8. Place *controls* in the window—these might include *entry boxes* for editing fields from the database, *command buttons* for initiating or cancelling an action, *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* below, and see the *Setting Control Properties* chapter.

9. Return to the **Procedure Properties** dialog.

CHOOSING A WINDOW TYPE

Overview: Windows

In most Windows programs there are three types of screen windows used: application windows, document windows, and dialog boxes. An application window is the first window opened in a Windows program, and it usually contains the main menu as the entry point to the rest of the program. All other windows in the program are document windows or dialog boxes.

Along with these three screen window types, there are two user interface design conventions that are used in Windows programs: the Single Document Interface (SDI), and the Multiple Document Interface (MDI).

An SDI program usually only contains linear logic that allows the user to take only one execution path (thread) at a time; it does not open separate execution threads which the user may move between. This is the same type of program logic used in most DOS programs. An SDI program would not contain a Clarion APPLICATION structure as its application window. The Clarion WINDOW structure (without an MDI attribute) is used to define an SDI program's application window, and the subsequent document windows or dialog boxes opened on top of it.

An MDI program allows the user to choose multiple execution paths (threads) and change from one to another at any time. This is a very common Windows program user interface. It is used by applications as a way of organizing and grouping windows which present several execution paths for the user to take.

A Clarion APPLICATION structure defines the MDI application window. The MDI application window acts as a parent for all the MDI child windows (document windows and dialog boxes), in that the child windows are clipped to its frame and automatically moved when the application frame is moved. They can also be concealed en masse by minimizing the parent. There may be only one APPLICATION open at any time in a Clarion Windows program.

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 usually display data. By convention they are movable and resizable. They usually have a title, a system menu, and maximize button. 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 usually request information from the user or alert the user to some condition, usually prior to performing some action requested by the user. They may or may not be movable, and so, may or may not have a system menu and title. By convention, they are not resizable, although they can have a maximize button which gives the dialog two alternate sizes. A dialog box may be system modal (the user must respond before doing anything else in Windows), application modal (the user must respond before doing anything in the application), or modeless. For example, in the Clarion environment, the window that appears from the File menu's Open selection is an application modal dialog box that requests the name of the file to open.

Default Window Structures

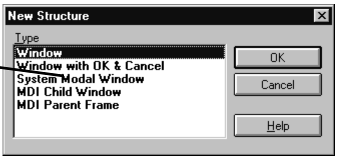
Some of the types of windows you can create appear in the **New Structure** dialog. The items in the **New Structure** dialog represent Clarion structures. You may see window structures, or report structures, or both, depending on how you access the dialog. A window structure is a group of Clarion Language statements that defines all the attributes of a window. You may want to think of a window structure as the definition of the window.

This section discusses only the default window structures supplied with this release; however, once you choose a default window structure, you may modify it. You may even add your own default window structures by editing the C:\CW15\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 should appear in the **New Structure** dialog.

The New Structure dialog lists the types of windows and report structures you can create.



Following is a description of the types of window structures provided with this release.

Window

To create a general purpose document window or dialog box, choose **Window** from the **New Structure** dialog. The **Window Formatter** generates a non-Multiple Document Interface (non-MDI) WINDOW structure with no controls. That is, a bare or empty window. See the *Windows Design Issues* Appendix for more information on MDI versus non-MDI windows and their uses.

This window will accept any controls (list boxes, entry boxes, buttons, etc.) you want to add. See *Placing Controls in a Window* below. Also see the *Setting Control Properties* chapter.

Window with OK & Cancel

To create a general purpose dialog box with **OK** *and* **Cancel** *buttons*, choose **Window with OK & Cancel** from the **New Structure** dialog. Again, the **Window Formatter** generates a non-Multiple Document Interface (non-MDI) WINDOW structure, however, this window already contains **OK** and **Cancel** buttons. The window will accept any other controls (list boxes, entry boxes, buttons, etc.) you want to add. You may also delete the **OK** and **Cancel** buttons if you want to.

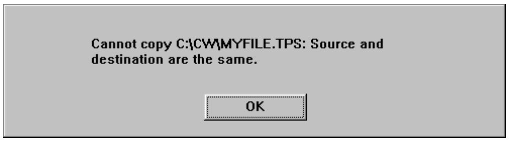
At the source code level, the only difference between this choice and the previous one is that the window structure starts out with two additional statements creating the **OK** and **Cancel** buttons—there is no difference in the window type, and there are no procedure calls associated with the two buttons. Clarion provides this additional choice simply for convenience, because these two buttons are such common elements in many application windows.

Tip: For most *data entry* windows, you can use one of these two choices (dialog boxes). The window might appear with a single line frame, entry fields for the user to type in the data for a new record, plus the command buttons. When the user presses the OK button, the application might add the record; if the user presses the Cancel button, the operation should be abandoned.

System Modal Window

To create a system modal window, choose **System Modal Window** from the **New Structure** dialog. System modal means the user will not be able to do anything else—not even switch to another application—until the window closes. The **Window Formatter** generates a WINDOW structure with the MODAL attribute.

Tip: ***To signal a critical error, use a system modal window.*** **Windows is a cooperative multi-tasking environment, in which the user should be in control as much as possible. A system modal window temporarily limits the user's freedom to run another application. Unless your application has a compelling reason to halt *all* system activity — for example, a severe file error which might result in lost data unless corrected at once — limit your application's use of this type of window.**


Cannot copy C:\CW\MYFILE.TPS: Source and destination are the same.

MDI Child Window

To create a document window which will appear only inside an application frame, choose **MDI Child Window**. The **Window Formatter** generates a WINDOW structure with the MDI attribute.

In general, MDI—the Multiple Document Interface—provides a method for presenting and editing more than one document, or more than one view of a document, within a single application. That is, the same MDI window may be opened many times from the same application. Each *document* appears in a *child* window inside the main application window.

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. A typical use of an MDI window might be to present a different, or second, or third view of your application's database.

All MDI windows must reside in separate procedures and execution threads from the APPLICATION window (see *MDI Parent Frame* below). This means the Application Generator must use the START function to begin a new execution thread for the MDI window called from the APPLICATION frame. You may execute more than one MDI window in the thread.

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.

Tip: Typically, the APPLICATION window should have a resizeable frame, plus a system menu, maximize and minimize buttons, and a menu. Usually, the File menu should provide a command to open or create MDI child windows, and a Window menu should provide commands for managing the separate child windows.

The APPLICATION window cannot have controls inside the window—in fact, Clarion will not allow you to place any there. MDI child windows contain all controls in an MDI application. The APPLICATION window should hold only the child windows, and optionally, a toolbar (which may contain controls).

Place any global menu commands or toolbar controls in the APPLICATION window. *Each child window inherits these commands and toolbars*, and may add additional items, or enable and disable global commands and toolbar controls as necessary.

The APPLICATION window and its MDI children must *not* reside in the same procedure. This requires the use of the START function, so that the MDI child runs in a separate thread. Multiple MDI windows may run in the same thread, but not the same thread as the APPLICATION window.

Tip: When the OPEN(AppFrame) statement executes, Clarion hides the window until it encounters the first DISPLAY statement or ACCEPT loop. This allows your application to make any cosmetic changes to the APPLICATION window before the user sees it.

CUSTOMIZING YOUR APPLICATION'S WINDOWS

Once you specify the type of window to create, the **Window Formatter** appears. It contains five major components to help you create or edit your window.

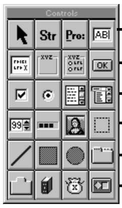
Window Formatter Tools

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 action bar.

Controls Toolbox

The **Window Formatter** contains a floating **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.



No Tool	**String**	**Prompt**	**Entry Box**
Text Box	**Group Box**	**Option Box**	**Button**
Check Box	**Radio Button**	**List Box**	**Combo Box**
Spin Box	**Progress Bar**	**Image**	**Region**
Line	**Box**	**Ellipse**	**Property Sheet**
Tab	**Dictionary Field**	**Custom Control**	**Control Template**

String	Allows you to place a STRING control on the window under construction. See *Setting String Control Properties*.
Prompt	Allows you to place a PROMPT control on the window under construction. See *Setting Prompt Control Properties*.
Entry Box	Allows you to place an ENTRY control on the window under construction. See *Setting Entry Box Properties.*
Text Box	Allows you to place a TEXT control on the window under construction. See *Setting Text Control Properties*.

Group Box	Allows you to place a GROUP control (group box) on the window under construction. See *Setting Group Box Control Properties*.
Option Box	Allows you to place an OPTION control (OPTION structure, which appears as a group box with radio buttons) on the window under construction. See *Setting Option Box Control Properties*.
Button	Allows you to place a BUTTON control on the window under construction. See *Setting Push Button Properties*.
Check Box	Allows you to place a CHECKBOX control on the window under construction. See *Setting Check Box Properties*.
Radio Button	Allows you to place a RADIO control on the window under construction. See *Setting Radio Button Properties*.
List Box	Allows you to place a LIST control (list box, or drop down list box) on the window under construction. See *Creating List Boxes* in the *Setting Control Properties* chapter.
Combo Box	Allows you to place a COMBO control (combo box, or drop combo box) on the window under construction. See *Setting Combo Box Properties*.
Spin Box	Allows you to place a SPIN control on the window under construction. See *Setting Spin Box Properties*.
Progress Bar	Allows you to place a PROGRESS control on the window under construction. See *Setting Progress Bar Properties*.
Image	Allows you to place an IMAGE control (graphic image) on the window under construction. See *Setting Image Control Properties*.
Region	Allows you to place a REGION control on the window under construction. See *Setting Region Control Properties*.

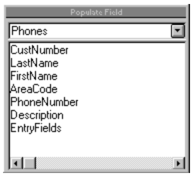
Line	Allows you to place a LINE control on the window under construction. See *Setting Line Control Properties*.
Box	Allows you to place a BOX control on the window under construction. See *Setting Box Control Properties*.
Ellipse	Allows you to place an ELLIPSE control on the window under construction. See *Setting Ellipse Control Properties*.
Sheet	Allows you to place a SHEET control on the window under construction. Sheet controls contain Tab controls. See *Setting Sheet Control Properties*.
Tab	Allows you to place a TAB control on the window under construction. Tab controls may contain any other control types. See *Setting Tab Control Properties*.
Dictionary Field	Allows you to select a field defined in the Data Dictionary, and place the control specified in the data dictionary, plus an associated PROMPT control, on the window under construction.
Custom Control	Allows you to place a CUSTOM control (Visual Basic custom control) on the window under construction. See *Setting Custom Control Properties*.
Control Template	Allows you to place a Control Template on the window under construction. See the *Using Control, Code, and Extension Templates* chapter.

Display or hide the **Controls** toolbox by choosing **Options** ä **Toolbox.** All the controls in the toolbox are also available from the **Controls** menu. See *Placing Controls in a Window* below. Also see the *Setting Control Properties* chapter.

Tip: Position the cursor over any tool and wait for half a second. A tool tip appears telling you the type of control that will be created by this tool.

Fields Toolbox

The **Window Formatter** contains a floating **Populate Field** toolbox. This toolbox allows you to quickly "populate" a window with entry controls for fields in your data files. First, choose a *file* from the drop down list, then DOUBLE-CLICK the *field* you want to appear on your window to place *both* a prompt, and an entry 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.



Display or hide the **Populate Field** toolbox by choosing **Options ➤ 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.

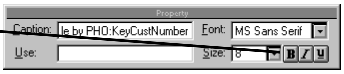
You may also populate a window with entry controls for fields in your data files by using the **Populate** menu, or by using the Dictionary Fields tool in the **Controls** toolbox.

Property Toolbox

The **Window Formatter's Property** toolbox allows you to 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.

Display or hide the **Property** toolbox by choosing **Options ➤ 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.

Text Formatting buttons—Bold, Italic and Underline.

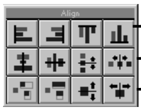


Align Toolbox

The **Window Formatter's Align** toolbox allows you to quickly, professionally, and precisely align the controls in your window. Select the controls to align (CTRL+CLICK allows you to select multiple controls, or you can "lasso" multiple controls with CTRL+DRAG), then click on the appropriate alignment tool. All the alignment actions are also available from the **Align** menu.

Display or hide the **Align** toolbox by choosing **Options ➤ 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.

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 Vertical	Along a vertical axis, aligns the centers of the selected controls with the center of the last control selected (red handles).

Align Horizontal	Along a horizontal axis, aligns the centers of the selected controls with the center of the last control selected (red handles).
Spread Vertical	Equalizes the vertical spaces between the selected controls.
Spread Horizontal	Equalizes the horizontal spaces 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).
Center Vertical	As a group (relative positions of selected controls don't change), centers the selected controls horizontally within the window.
Center Horizontal	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 appears telling you the type of alignment this tool will accomplish.

Window Formatter Procedures

The **Window Formatter** lets you directly manipulate the window and the controls inside it. The initial sample window, for example, contains 'handles' — the tiny boxes located at the corners and sides of the window. By selecting a corner 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 creating the window.

Similarly, the **Window Formatter** supplies the other attributes by presenting you with options, check boxes and fields in which you specify your preferences.

Here is the typical process for customizing a new window with the **Window Formatter**:

1. Set the *size* of the window by dragging the 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 and choose **Properties** from the popup menu, or select the window and choose **Edit ➤ Properties.**

   Other attributes include the window caption, whether the window is resizeable, whether the window is scrollable, icons associated with the window, messages, help files, and cursor types associated with the window, and many others. See *Using the Window Properties Dialog* below.
3. Close the **Window Properties** dialog.
4. Place controls in the window.

   See *Placing Controls in a Window* below. Also see the *Setting Control Properties* chapter.
5. Preview the window by choosing **Preview!** from the action bar, then make any adjustments necessary while still in the **Window Formatter**.
6. Choose **Exit!** from the action bar to return to the Application Generator or Text Editor.

Using the Window Properties Dialog

Use the **Window Properties** dialog to set *all* the properties, or attributes, of a *window*. Properties include the window caption, whether the window is *resizeable*, 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 display the **Window Properties** dialog from the **Window Formatter** you may:

❏ RIGHT-CLICK on the sample window and choose **Properties** from the popup menu.

❏ Select the sample window and press ENTER.

❏ Select the sample window and choose **Edit ➤ Properties** from the menu.

Additionally, each choice in the **New Structure** dialog leads to the **Window Properties** dialog.

General Properties Tab

Title

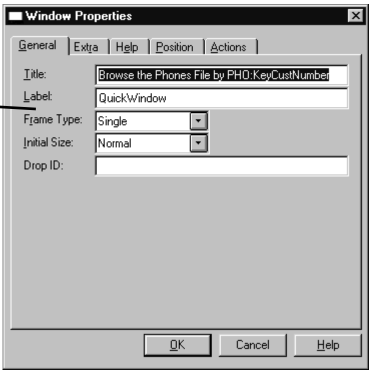
To specify caption bar text for your window, type a string constant in the **Title** field. The caption bar displays the name of the window.

Tip: You may dynamically alter the caption bar text at runtime. For example, you may place a file name variable on the title bar: 'Notepad - FileName.TXT.' To accomplish this, embed the following code after the window is opened and before the ACCEPT statement:

```
MyWindow{PROP:text} = 'My Caption - ' & FileNameVariable
```

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.

The Window Properties dialog displays the attributes you add to the WINDOW structure.



Title: Browse the Phones File by PHO:KeyCustNumber
Label: QuickWindow
Frame Type: Single
Initial Size: Normal
Drop ID:

Label

To specify the label for the window, type it in the **Label** field. 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

To choose the frame type for your window, pick a selection from the **Frame Type** drop down list. The frame defines the borders of the window. The selections are:

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).
Resizeable	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

To specify the initial size and state of your window, choose an option from the **Initial Size** drop down list. The choices are:

Normal	Display the window at the default size. If you don't specify a default size, Clarion's runtime library will set 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.

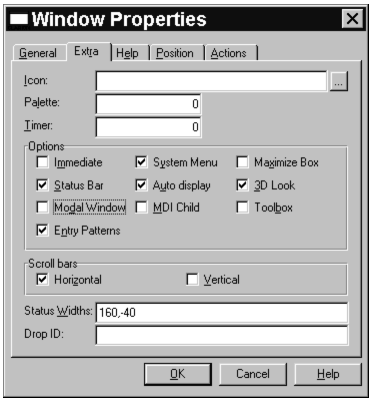
Tip: If you choose the *iconized* selection, be sure to specify a file name in the Icon field. If not, your window may not receive a Restore command on its system menu, which means it will always remain iconized. Specifying a file name also adds a minimize button to the window, allowing the user to iconize the window again, after restoring it.

Extra Properties Tab

Icon

To associate an icon with the window, specify an icon file name (.ICO file) in this field. You may type in a file name or press the ellipsis button (...), then select an icon file name using the standard **Open File** dialog.

You should *always* specify an icon for an application window, and for an MDI child window. Specifying an icon name automatically places a minimize button on the caption bar of your application or MDI child window.



Icon:

Palette: 0

Timer: 0

Options

- [ ] Immediate
- [✓] System Menu
- [ ] Maximize Box
- [✓] Status Bar
- [✓] Auto display
- [✓] 3D Look
- [ ] Modal Window
- [ ] MDI Child
- [ ] Toolbox
- [✓] Entry Patterns

Scroll bars

- [✓] Horizontal
- [ ] Vertical

Status Widths: 160,-40

Drop ID:

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 will automatically receive an EVENT:Timer once every second. This might be appropriate for adding a clock to a status bar.

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 for Windows application. At runtime, your application uses only one Windows timer.

Options *To toggle the following various options on or off*, check or uncheck the corresponding boxes.

Immediate *To generate an event each time the end user moves or resizes the window*, check the **Immediate** 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 the **Status Bar** box. See *Status Widths* below for information on segmenting or zoning your status bar.

Tip: A status bar in an application window is an excellent 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 the **Modal Window** 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 the **Entry Patterns** 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.

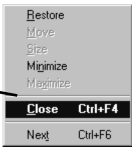
Tip: Entry Patterns are also known as Picture Tokens.

The entry patterns, or picture tokens, are specified on the **General Tab** of the **Entry Properties** dialog. See the *Setting Control Properties* chapter (*Setting Entry Box 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. A **System Menu** is always activated by the button, box, or icon in the upper left corner of the window. Standard **System Menu** choices include Restore, Minimize, Maximize, and Close.

A standard system menu.



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.

Tip: Even if you decide the window should NOT have a system menu when the application is complete, it's good practice to place a system menu on your application during development. By DOUBLE-CLICKING the system menu, or choosing Close, you can close your application if your normal exit procedure doesn't work.

Tip: Even if not absolutely necessary, a system menu serves as an added convenience for dialog boxes. Many users automatically DOUBLE-CLICK on the system menu box to close a dialog—since it always resides in a standard location, it is often easier to find than a Cancel button.

Auto Display	*To add the AUTO attribute to your window*, check the **Auto Display** 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 the **MDI Child** box. An MDI child window cannot move outside the main application window. A typical use of an MDI window might be to present a different arrangement of the data in your application's database.
Maximize Box	*To place a maximize button in your window*, check the **Maximize Box** 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 the **3D Look** 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 the **Toolbox** box. The TOOLBOX attribute makes your window always stay "on top" of other open windows.

Scroll Bars *To toggle the following scroll bar options on or off*, check or clear the corresponding check boxes.

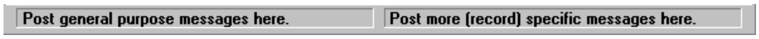
Horizontal *To add a horizontal scroll bar to your window*, check the **Horizontal** box.

Vertical *To add a vertical scroll bar to your window*, check the **Vertical** box.

Status Widths *To set the width of status bar zone(s)*, type a value, or a list of values separated by commas, in the **Status Widths** field. 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 the *Language Reference* for more information on the STATUS() attribute.

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* in the **Drop ID** field. 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.

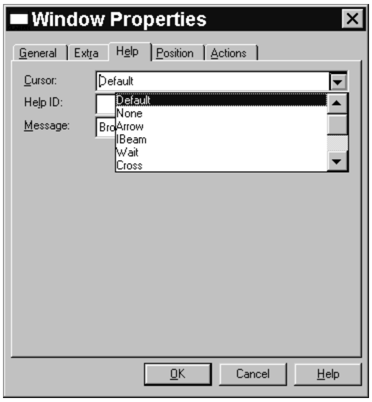
Drag and Drop capability means the end user can select an item in one window or control, hold down the left mouse button, “drag” the item to another window or control, and release the mouse button, “dropping” the item onto the other window or control, which can then look at the item that was “dropped” 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 the *Using the List Box Formatter* chapter, *Adding Drag and Drop Capability to the List Box* section of this book.

Help Properties 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* 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), in the **Help ID** field. 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.

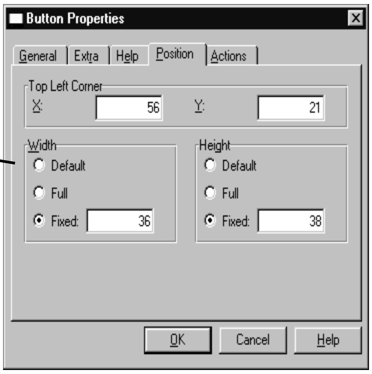
When generating code, the Application Generator calls the context string or keyword in the .HLP file you specify in the **Application Properties** dialog. See the *Using the Application Generator* chapter for further information.

Message *To display a message in zone one of the status bar* when the window is active, type the message in the **Message** field. 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 message will be displayed instead of the window's message. See *Status Bar* and *Status Widths* above for more information on the status bar. See the *Language Reference* for more information on the MSG attribute.

Position Properties Tab

Generally, you will want *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 from the **Position Properties Tab.**

The Position tab controls the position *and* size of the window. When you choose *default* positions, Windows places your window at a point dependent upon where the last window (even from another application) was opened.



Top Left Corner
X: 56
Y: 21

Width
Default
Full
[✓ Fixed:] 36

Height
Default
Full
[✓ Fixed:] 38

Top Left Corner *To specify the initial position (of the top left corner) of your window,* choose the desired **X** and **Y** coordinates. The choices are:

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. The choices are:

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**.

PLACING CONTROLS IN A WINDOW

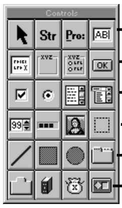
This sections explains how to *place* a control in a window. The *Setting Control Properties* chapter explains how to *customize* the controls you place in your windows.

Controls Toolbox

The **Controls** Toolbox appears when you start the **Window Formatter**. Hide or re-display the **Controls** toolbox by choosing **Options ➤ Toolbox.** All the controls in the toolbox are also available from the **Controls** menu. The **Controls** Toolbox works exactly like a palette of drawing tools, such as the toolbox in the Windows Paintbrush accessory. To place a control:

1. CLICK on an icon in the toolbox. Each control has its own icon.

   Position the cursor over any toolbox icon and wait for half a second—a tool tip appears telling you the *type* of control that will be created.



No Tool	**String**	**Prompt**	**Entry Box**
Text Box	**Group Box**	**Option Box**	**Button**
Check Box	**Radio Button**	**List Box**	**Combo Box**
Spin Box	**Progress Bar**	**Image**	**Region**
Line	**Box**	**Ellipse**	**Property Sheet**
Tab	**Dictionary Field**	**Custom Control**	**Control Template**

   When you have selected a control tool, then pass the cursor over the sample window, the cursor becomes a cross.

2. CLICK inside the window you wish to add the control to.

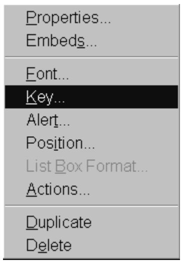
   The upper left hand corner of the control is placed at the intersection of the cursor cross when you CLICK the mouse.

3. 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.

THE WINDOW FORMATTER MENUS

Using the Popup Menu

Access the popup menu by RIGHT-CLICKING a window or a control. The popup menu on the **Window Formatter** allows you to 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.

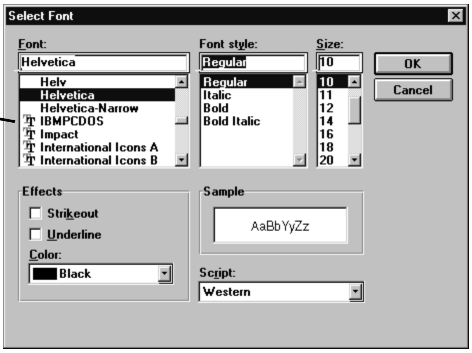
Tip: All of the popup menu commands are also available on the Window Formatter Edit menu.

Following is a description of the popup menu choices.

Properties	*To edit control or window properties*, select a control or window, and choose the **Properties** command. See *Using the Window Properties Dialog* above, or see the *Setting Control Properties* chapter for more information. You may also RIGHT-CLICK a control or window, and select the **Properties** command from the popup menu.
Embeds	*To add or edit embedded source associated with a control or window*, select it and choose the **Embeds** command. See the *Defining Embedded Source Code* section of the *Using the Application Generator* chapter.

Font

To control the appearance of the text displayed in a control or window, select the control or window and choose the **Font** command. Specify font, size, style, script, and color from drop down list boxes. 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.

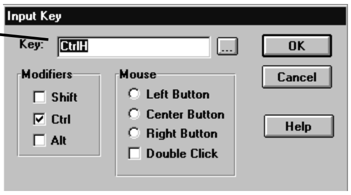


Setting text attributes with the Select Font dialog.

Key

To specify a "hot" key for a control, select the control and choose the **Key** command (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, will give *immediate focus* to the control, or, for an action control such as a command button, will *initiate 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 pressed will appear in the **Key** field, and will be supplied as parameters to the KEY attribute for this control.

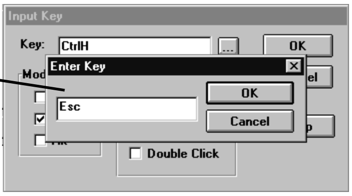


Adding a "hot" key for CTRL+H

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.

Adding ESC to the hot key sequence.



The ESC, ENTER, and TAB keys *cannot* be specified by pressing them. For these keys, press the ellipsis (...) button and type “esc,” “enter,” or “tab.”

The following controls receive focus from the KEY attribute:

Combo Box
Entry Field
Group Box
List Box
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
Custom Control
Radio Button

The KEY attribute is not applicable to the following controls:

String
Progress Bar
Image
Region
Line
Box
Ellipse

Alert

To specify an Alert key for a window or a control, select the window or control and choose the **Alert** command. 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 and choose the **Position** command. See *Position* in the *Using the Window Properties Dialog* section above 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** command (and therefore the **Position Tab** of the various control properties dialogs) to position your controls. See the *Setting Control Properties* chapter for more information. Also see *Grid Settings* in the *Using the Options Menu* section below.

List Box Format

To specify the appearance and functionality of a list box control, select the list box and choose the **List Box Format** command. See the *Using the List Box Formatter* chapter for more information.

Actions	*To specify the Actions associated with a control or window*, select it and choose the **Actions** command. See the *Setting Control Properties* chapter for more information.

Using the Edit Menu

The **Edit** menu on the **Window Formatter** allows you to 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 and 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 *Using the Popup Menu* section above:

Undo	*To reverse the last editing action*, choose the **Undo** command. All **Window Formatter** actions may be reversed, except deleting a control.
Redo	*To redo the undone action*, choose the **Redo** command. Not all actions may be redone.
Delete	*To delete a control or window*, select it and choose the **Delete** command, or select it and press the DELETE key.
Duplicate	*To place a copy of a control in the same window*, select the original and choose the **Duplicate** command. The copy will appear next to the original.

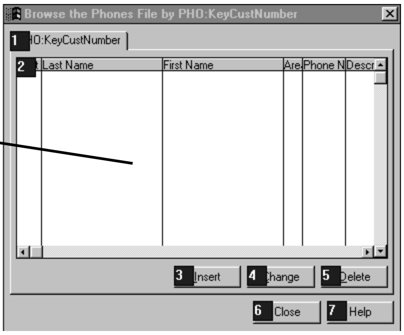
Tip: You may duplicate *multiple* controls by selecting multiple controls before invoking the Duplicate command. Use CTRL+CLICK to select multiple controls, or lasso multiple controls using CTRL+CLICK+DRAG.

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 the **Set Tab Order** command. This opens the **Ordering Type** dialog, which allows you to specify *how* the tab-stop order is set: Automatically or Manually, Horizontally or Vertically. See *Set Control Order* below 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.

Visually Setting Tab Order by the numbers.



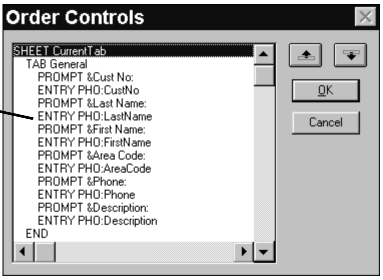
Alternatively, from the **Ordering Type** dialog, select the **Automatic** radio button, then choose either the **Horizontally** or **Vertically** radio button. Press the **OK** button to automatically set the tab-stop order based on the position of the controls. **Horizontally** numbers the topmost 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.

Control Templates *To add Control Templates to a control or window*, select the control or window and choose the **Control Templates** command. This opens the **Edit Control Templates** dialog, which allows you to access the **Prompts** dialogs of any control templates in the window. This is equivalent to RIGHT-CLICKING a control template, then choosing **Actions** from the popup menu. See the *Using Control, Code, and Extension Templates* chapter for more information.

Set Control Order *To set the tab key order, and move controls among overlapping tab controls*, choose the **Set Control Order** command. 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 and pressing the ↑ and ↓ buttons to move the control up or down within the list.

Setting Tab Order using the Order Controls dialog list.



```
SHEET CurrentTab
  TAB General
    PROMPT &Cust No:
    ENTRY PHO:CustNo
    PROMPT &Last Name:
    ENTRY PHO:LastName
    PROMPT &First Name:
    ENTRY PHO:FirstName
    PROMPT &Area Code:
    ENTRY PHO:AreaCode
    PROMPT &Phone:
    ENTRY PHO:Phone
    PROMPT &Description:
    ENTRY PHO:Description
  END
```

Using the Control Menu

The **Control** menu lists the controls that appear in the Controls Toolbox. 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* section above. Also see the *Setting Control Properties* chapter.

Using the 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 serves as a convenience.

For a list of Alignment tools, see the *Window Formatter Tools* section above.

Using the Menu Menu

The **Menu** menu lets you add, change, or delete menus from your window.

When you specify a menu for your application window, or for MDI child windows, Clarion automatically merges the application window menu with the MDI child menus, when an 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 *Creating Menus and Toolbars* chapter for directions on how to create menus and toolbars for your application.

Using the 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.

Please see the *Creating Menus and Toolbars* chapter for directions on how to create menus and toolbars for your application.

Using the 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	Allows you to place an entry field 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.

If you specified a prompt for the field when creating the data dictionary, the first CLICK places the prompt for the control. The second CLICK places the control. If you pre-formatted the field, on the **Window** tab of the **Field Properties** dialog (for example, specifying a spin control), the control you specified appears, rather than an entry box.

Multiple Fields Allows you to place an entry field 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.

If you specified a prompt for the field when creating the data dictionary, the first CLICK places the prompt for the control. The second CLICK places the control. If you pre-formatted the field, on the **Window** tab of the **Field Properties** dialog (for example, specifying a spin control), the control you specified appears, rather than an entry box.

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.

Tip: When placing file fields and memory variables with these commands, you can use the Embeds dialog to attach code to the events generated by the controls.

Control Template Allows you to add a control template to the window under construction. Select one from the **Select a Control Template** dialog.

A control template adds a control or controls to the window, plus the code to maintain them. For example, the Browse Box control template places a list box in the window, allows you to choose the fields for the list, and adds all the executable code for managing the list box (loading it, scrolling it, etc.).

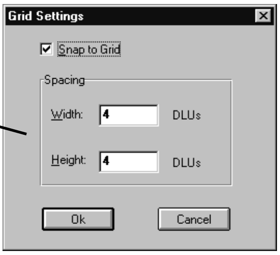
Once the control template is placed, you can specify its properties and actions by RIGHT-CLICKING and selecting **Properties** or **Actions** from the popup menu. See the *Using Control, Code, and Extension Templates* chapter for more information.

Using the Options Menu

The **Options** menu allows you to display and hide the various **Window Formatter** tools and toolboxes.

Show Toolbox	*To toggle the Controls toolbox display on and off*, choose the **Show Toolbox** command. 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** menu.
Show Alignbox	*To toggle the Alignbox display on and off*, choose the **Show Alignbox** command. This is a matter of individual preference. You may still access all the alignment commands by choosing them from the **Alignment** menu.
Show Propertybox	*To toggle the Propertybox display on and off*, choose the **Show Propertybox** command.
Show Fieldsbox	*To toggle the Fieldsbox display on and off*, choose the **Show Fieldsbox** command. This is a matter of individual preference. You may still access populate fields using the **Controls** toolbox or the **Populate** menu.
Grid Settings	*To turn on grid snap, as well as setting the grid values*, choose the **Grid Settings** command. Grid snap forces the upper left corner of new controls to align with a dot grid in the window. The end user does not see the grid at run time; it is a design tool only. To turn grid snap on, check the **Snap to Grid** box.

The default grid settings.



Snap to Grid: checked
Width: 4 DLUs
Height: 4 DLUs

Tip: Aligning the controls in your dialog boxes and windows will give your application a more professional look. See the appendix on Windows Design 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.

VBX Custom Control Registry *To add a custom control library to the registry*, choose the **VBX Custom Control Registry** command. Press the **Add** button in the *VBX Custom Control Registry* dialog. Then DOUBLE-CLICK on the .VBX file name.

This enables the **Window Formatter** to place controls from the library in your window (see the *Setting Control Properties* chapter). To remove a registration, press the **Remove** button.

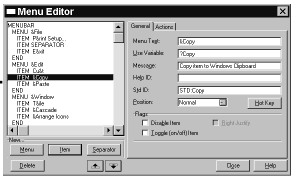
Using Preview!

To display an active window identical to the one that your user will see, choose the **Preview!** command. 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 3D 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.

CREATING MENUS AND TOOLBARS

Create menus and toolbars with the Menu Editor, which you access through the Window Formatter.



Menu Editor

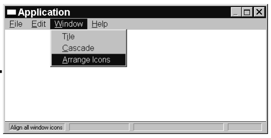
```
MENUBAR
  MENU  &File
    ITEM  P&rint Setup...
    ITEM SEPARATOR
    ITEM  E&xit
  END
  MENU  &Edit
    ITEM  Cu&t
    ITEM  &Copy
    ITEM  &Paste
  END
  MENU  &Window
    ITEM  T&ile
    ITEM  &Cascade
    ITEM  &Arrange Icons
  END
```

General | Actions

Menu Text: &Copy
Use Variable: ?Copy
Message: Copy item to Windows Clipboard
Help ID:
Std ID: STD:Copy
Position: Normal

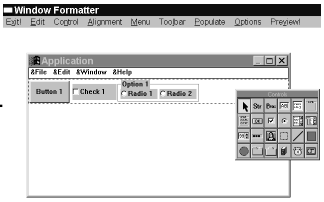
Flags: ☐ Disable Item ☐ Right Justify ☐ Toggle (on/off) Item

The Menu Editor dialog represents the MENUBAR structure in a visual tree. Add menu items at the touch of a button.



Clarion for Windows STD IDs (Windows Standard Functions) provide automatic implemenation of common Windows commands such as Copy and Paste. It even provides an automatic window list for MDI applications.

Clarion for Windows provides standard toolbar icons for common functions, such as copy and paste.



Easily place any type of control on a toolbar.

A menu is simply a list of the various actions your application may perform. In Clarion, this list of actions (menu) is displayed using the MENUBAR structure, MENU structures, and ITEMs. In this chapter, the word menu is used generically to refer to the list of actions your application may perform. The words MENUBAR, MENU, and ITEM are used to refer to Clarion Language statements that define your application's menu.

This chapter will:

- Discuss dynamic menu and toolbar management for Multiple Document Interface (MDI) applications.
- Show you how to call the **Menu Editor** and create a menu.
- Describe how to automatically implement standard Windows behavior for commands such as **Edit ➤ Copy** by linking a Clarion Standard ID (STD attribute) to an ITEM or MENU.
- Show you how to create a toolbar. Clarion even provides access to icons for standard actions such as **File ➤ Open**, so that your application will look more professional.

OVERVIEW: CREATING 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 the *Windows Design Issues* appendix for more information.

Merging Menus

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) is merged 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 the 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.

   You will 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 when Clarion merges this global menu with local menus, File and Help will keep their correct positions.

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.

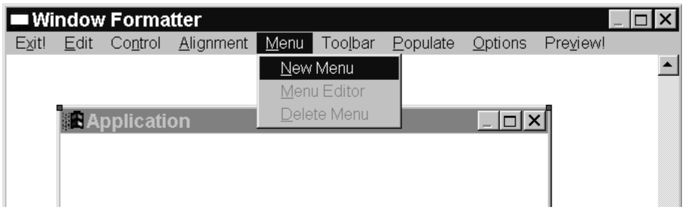
   Do they merge the way you planned? Are the correct selections available for the window with focus? Make any adjustments with the **Menu Editor**.
6. Repeat steps 3 through 6 for other child windows.

CALLING THE MENU EDITOR

To create a menu for your application, use the **Menu Editor**. You access the **Menu Editor** through the **Window Formatter**.

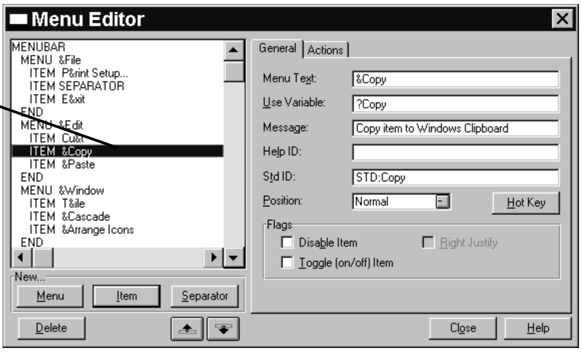
Tip: You can also create a toolbar for your application using the Window Formatter. You can place any type of control on your toolbar, though you will probably use command buttons the most.

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:

The Menu Editor displays the menu under construction as a hierarchical list.



```
MENUBAR
  MENU  &File
    ITEM  P&rint Setup...
    ITEM SEPARATOR
    ITEM  E&xit
  END
  MENU  &Edit
    ITEM  Cu&t
    ITEM  &Copy
    ITEM  &Paste
  END
  MENU  &Window
    ITEM  T&ile
    ITEM  &Cascade
    ITEM  &Arrange Icons
  END
```

Menu Text:	&Copy
Use Variable:	?Copy
Message:	Copy item to Windows Clipboard
Help ID:	
Std ID:	STD:Copy
Position:	Normal

Flags
- [ ] Disable Item
- [ ] Right Justify
- [ ] Toggle (on/off) Item

- 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 allows you to 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 allows you to 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.

CREATING YOUR APPLICATION'S MENU

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. In the **New** group box, press the **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 is displayed.

3. In the **Menu Text** field, type the text you want displayed for this MENU.

   For example, type &FILE, so the end user sees **File**.

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 within executable code by its Field Equate Label.

5. In the **New** group box, press the **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 you want 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 is *executed*.

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 be currently displayed. See *Adding a Hot Key* below for another method of accessing 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 executable code by its Field Equate Label. For example, within the ACCEPT loop of the generated source code, the CASE FIELD() structure will refer to the equate label ?FileOpen when testing to see if the user selected this ITEM.

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.

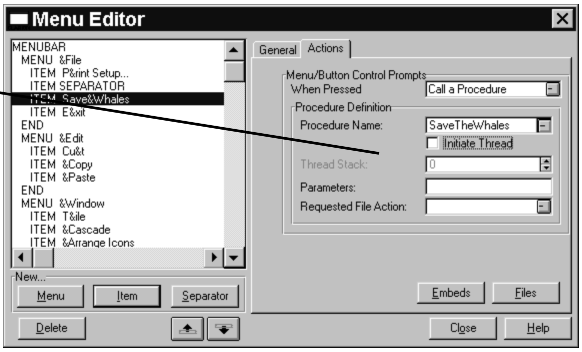
Tip: When authoring a Windows Help file, you indicate a keyword with the 'K' footnote. A Help context string is the arbitrary string which uniquely identifies each topic page for the Windows Help Compiler. When creating the Help file, the '#' footnote marks a context string. These tasks are all done for you by many third party help 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, 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).

Using the Actions tab to specify a procedure to call when a menu ITEM is selected.



```
MENUBAR
  MENU &File
    ITEM P&rint Setup...
    ITEM SEPARATOR
    ITEM Save&Whales
    ITEM E&xit
  END
  MENU &Edit
    ITEM Cu&t
    ITEM &Copy
    ITEM &Paste
  END
  MENU &Window
    ITEM T&ile
    ITEM &Cascade
    ITEM &Arrange Icons
```

Menu/Button Control Prompts
When Pressed: Call a Procedure
Procedure Definition
Procedure Name: SaveTheWhales
Initiate Thread: ☐
Thread Stack: 0
Parameters:
Requested File Action:

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 Commands* below.

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.

OTHER MENU EDITOR FUNCTIONS

Additional buttons and check boxes allow you to specify execution of standard windows actions, placement of menu separators, designation of hot keys, default menu states, and menu merging preferences.

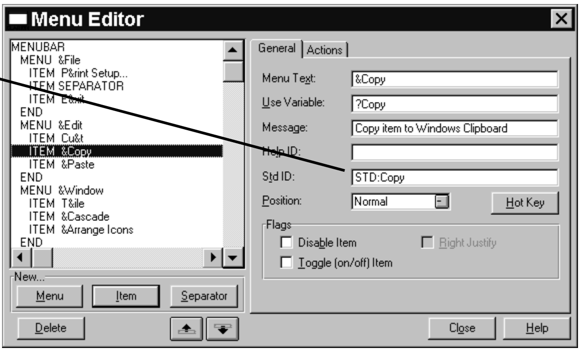
Implementing Standard Windows Commands - Std IDs

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 will automatically implement 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 C:\CW\LIBSRC\EQUATES.CLW file.

STD:PrintSetup	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 all title bars are visible.
STD:ArrangeIcons	Arranges iconized child windows.
STD:WindowList	Adds child window names to menu.
STD:Help	Opens .HLP file to the contents page.
STD:HelpIndex	Opens .HLP file to the index.
STD:HelpOnHelp	Opens Microsoft's .HLP file for the Windows Help system.
STD:HelpSearch	Opens Microsoft's Help Search utility for the .HLP file.

Implementing Standard Windows Behavior with an STD ID.



```
MENUBAR
  MENU  &File
    ITEM  P&rint Setup...
    ITEM SEPARATOR
    ITEM  E&xit
  END
  MENU  &Edit
    ITEM  Cu&t
    ITEM  &Copy
    ITEM  &Paste
  END
  MENU  &Window
    ITEM  T&ile
    ITEM  &Cascade
    ITEM  &Arrange Icons
  END
```

Menu Text: &Copy
Use Variable: ?Copy
Message: Copy item to Windows Clipboard
Help ID:
Std ID: STD:Copy
Position: Normal

☐ Disable Item ☐ Right Justify
☐ Toggle (on/off) Item

Specifying Menu Positions and Merging Behavior

The **Position** drop down list allows you to specify MENU and ITEM order priority when Clarion merges menus. The choices are:

Normal	*To allow normal ordering when merging menus*, choose ***Normal*** from the **Position** drop down list. In normal merging, Global selections precede Local selections. See *Merging Menus* above.
First	*To force the selected* MENU *or* ITEM *to the first position when merging menus,* choose ***First*** from the **Position** drop down list. This adds the FIRST attribute to the MENU or ITEM statement. See the *Language Reference* for more information. Also, see *Merging Menus* above.
Last	*To force the menu or item to the last position when merging menus*, choose ***Last*** from the **Position** drop down list. This adds the LAST attribute to the MENU or ITEM statement. See the *Language Reference* for more information. Also, see *Merging Menus* above.

The following two **Flags** allow you to specify whether or not your menu can be merged, and right justification of selections displayed on the menubar:

Do Not Merge	*To tell* Clarion *never to merge this* MENUBAR *with other* MENUBARs, check the **Do Not Merge** box. This is available only for the MENUBAR. See the *Language Reference* for more information on the NOMERGE attribute.
Right Justify	*To right justify the selected* MENU, check the **Right Justify** 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 a Hot Key

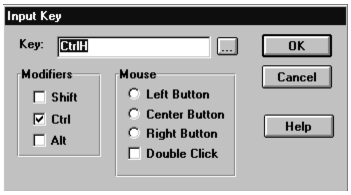
A hot key is very similar to an accelerator key. A hot key or hot key combination allows the end user to immediately display a MENU, or execute the action associated with an ITEM, without mouse clicking, and *without displaying the menu that contains the* ITEM. Customarily, hot keys take the form of CTRL + *character*, or CTRL + SHIFT + *character*. To add a hot key:

1. Press the **Hot Key** button.

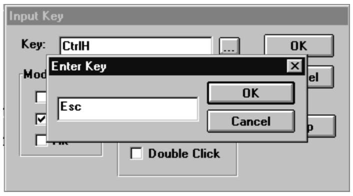
   The **Input Key** dialog appears. Use this dialog to add the KEY attribute to your MENU or ITEM. The KEY attribute specifies a "hot" key or key combination.

2. From the **Input Key** dialog, specify the hot key or key combination *by pressing the desired key or keys*.

   The keys you pressed will appear in the **Key** field, and will be supplied as the parameter to the KEY attribute for this menu item.



   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.



   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."

3. Press the **OK** button to return to the **Menu Editor** .

Tip: You may want to add the hot key combination to the menu text to signal its availability to the user. See the *Windows Design* appendix for a list of common hot keys associated with standard windows commands.

Other Menu Behavior - Disabling and Toggling

The following two **Flags** allow you to disable a selection, and to 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 the **Disable Item** 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 the **Toggle (on/off) Item** 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 *Use Variable* above. The **Menu Editor** adds the CHECK attribute to this ITEM.

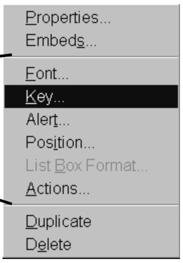
With the CHECK attribute, when the user selects the ITEM for the first time, the ITEM is "on," the Use Variable's value is one (1), and a check mark appears beside the ITEM. When the user selects the ITEM a second time, the ITEM is "off," the Use Variable's value is zero (0) and no check mark is displayed. You should add source code to control the application's behavior depending on the state of the Use Variable.

Managing Your Menu

Separator Button *To add a separator bar after the currently highlighted MENU or ITEM*, press the **Separator button**.

Tip: Separator bars can provide the user with a visual cue that a group of ITEMs on the menu perform related functions.

The Window Formatter's popup menu uses Separators to group related commands.



Delete Button *To delete the currently highlighted MENU or ITEM*, press the **Delete** button. If you delete a MENU statement, all ITEMs and MENUs within it, and its associated END statement are also deleted.

↑ and ↓ **Buttons** To move the currently highlighted MENU or ITEM up or down in the menu list, press the ↑ or ↓ button. When moving a MENU, all ITEMs and MENUs within it, and its associated END statement move also.

ADDING A TOOLBAR

You may add a toolbar to any window with a simple command in the **Window Formatter**. You may place any control on a toolbar, but the ones you will probably use the most are command buttons, check boxes, radio buttons, and drop down list boxes. As with menus, Clarion will automatically merge toolbars in certain situations.

Toolbar Merging

Global and Local Tools

The TOOLBAR structure declares the tools displayed for an APPLICATION or WINDOW. On an APPLICATION, the TOOLBAR defines the Global tools for the program. If the NOMERGE attribute is specified on the APPLICATION's TOOLBAR, the tools are local and are displayed only when no MDI child windows are open; there are no global tools. Global tools are active and available on all MDI child windows unless an MDI child window's TOOLBAR structure has the NOMERGE attribute.

MDI Windows

On an MDI window, the TOOLBAR defines tools that are automatically merged with the Global toolbar. Both the Global and the window's tools are then 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 in the window, not on any application which may have been previously opened.

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.

Note: To merge toolbars, the application's toolbar AT width must be less than the APPLICATION's frame width.

Adding a Push Button

The following describes how to add a toolbar to a window. The starting point is the **Window Formatter**, open to an empty window:

1. From the **Toolbar** menu, choose **New Toolbar**.

   A rectangular area appears at the top of the window. This is the toolbar. At runtime, it appears dark gray.

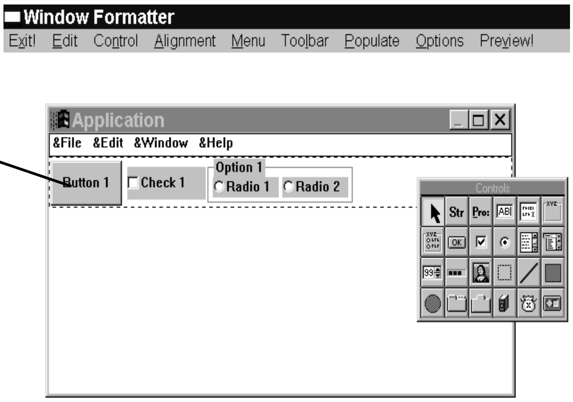
2. Optionally choose the **Options ➤ Grid Settings,** then check the **Snap to Grid** box.

   This makes sizing and placing the controls easier.

3. Press the **Button** icon (OK) in the Controls toolbox, then click inside the new toolbar in the sample window.

   A button control appears.

Placing buttons on a toolbar. After choosing a standard icon from the list, the "Button 1" button will display the appropriate icon.


Window Formatter
Exit! Edit Control Alignment Menu Toolbar Populate Options Preview!
Application
&File &Edit &Window &Help
Button 1
Check 1
Option 1
Radio 1
Radio 2
Controls

4. Press the RIGHT-CLICK on the button and select **Properties** from the popup menu, or choose **Edit ➤ Properties.**

   The **Button Properties** dialog for the new button appears.

5. Delete the default text in the **Parameter** field.

   This allows you to create a picture button without text.

6. From the **Extra** tab, choose an icon from the Icon drop down list, or type the name of an icon file (*.ICO) of your own.

   The icon list contains a number of default icons for such standard actions as File/Open, or Cut, Copy, and Paste.

7. From the **General** tab, type a descriptive Equate Label in the **Use** field.

   For a File/Open button, for example, you might type ?OpenButton. The Equate Label will appear in the **Embedded Source** dialog, making it easy to identify where you want to add functionality

8. Press the **OK** button to close the **Button Properties** dialog.

9. Resize the button to the size you want by dragging its handles.

   Handles are the tiny boxes that appear at the corners and sides of an item.

Tip: For toolbar buttons, Clarion for Windows uses .ICO files that are 32 x 32 pixels . Most toolbar buttons will be smaller—for example, 16 x 18 pixels. By using these larger files, we can create the "disabled" icon from 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.

Adding a “Latched” Button

A latched button “stays depressed” when CLICKED, then returns to its original state when CLICKED a second time. To place a latched button:

1. Press the **Check Box** icon in the Controls toolbox, then click inside the new toolbar in the sample window.

   The **Select Field** dialog appears.

2. Highlight Local Data, then press the Insert button.

   The **New Field Properties** dialog appears.

3. In the **Field Name** field, type a name, then choose ***BYTE*** from the data type drop down list.

   The **Check Box Properties** dialog appears. A button created from a check box control has two modes: on or off. When the check box is ‘on’ (it appears ‘pushed in’ to the user), and the value of its USE variable is one. When ‘off,’ it is zero.

4. From the **Extra** tab, choose an icon from the **Icon** drop down list, or type the name of an icon file (*.ICO) of your own.

5. Press the **OK** button.

   The button is complete; you need only adjust its position by dragging its center, if necessary.

Adding a Button Group

A button group provides the user with *mutually exclusive choices*. For example, in a group of three buttons, only one can be “depressed.” If button number two is currently “depressed,” push in button number one, and button number two pops out. For example, a button group can provide controls for left, right and center text justification—only one option can be active at a time.

To create a button group:

1. CLICK on the Option Box icon in the Controls toolbox, then CLICK inside the toolbar.

   The **Window Formatter** places an Option Box on the toolbar. You may resize it by dragging its handles. An Option Box—an OPTION structure—must always surround radio button choices, however, this Option Box will not appear on the toolbar, because you will hide it.

2. RIGHT-CLICK on the Option Box and choose **Properties** from the popup menu.

   The **Option Properties** dialog appears.

3. Type *JUSTIFICATION* in the **Use** field.

   The **Use** field takes the label of a variable. You must declare the variable with the **Global Data** dialog, the **Local Data** dialog, or by some other method. *This variable will receive a value indicating which button within the group the user selected*. The variable may be a string variable or a numeric variable. If it is a string, it will receive a text value, either the text from the selected button, or an alternative text value you specify. If it is a numeric, it will receive an integer value corresponding to the selected button, that is, button 1, 2, or 3.

4. From the **Extra** tab, uncheck the **Boxed** box.

   This hides the Option Box from the user. It appears in the **Window Formatter** dialog, but will not appear at runtime.

5. Press the **OK** button.

6. CLICK on the Radio Button icon in the Controls toolbox, then CLICK inside the Option Box.

   The Application Generator places a Radio Button where you clicked in the Option Box.

7. RIGHT-CLICK on the Radio Button and choose **Properties** from the popup menu.

   The **Radio Button Properties** dialog appears.

8. Clear the **Parameter** field.

   Clearing this field will remove text from the button so we can add an icon instead.

9. In the **Value** field, type *Left*.

   When the user presses this button, the string “Left” is assigned to the USE variable we specified above.

10. From the **Extra** tab, choose an icon from the **Icon** drop down list, or type the name of an icon file (*.ICO) of your own.

11. Press the **OK** button.

    The first button is complete; you need only adjust its position by dragging its center.

12. Repeat steps 6 through 11 for the “center” and “right” buttons.

Preview Your Menus and Toolbars

To test the runtime appearance of your toolbars and menus:

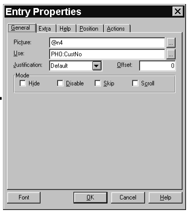
1. Choose **Preview!** from the **Window Formatter** menu.

This displays the window, including the toolbar and menus, as it would to the user at runtime. Test the latching and radio features by pushing the buttons. Press ESC when done previewing your window.

2. Choose **Exit!** from the **Window Formatter** menu to save your window.

SETTING CONTROL PROPERTIES

When you place controls using the Window Formatter, you must fill in the various control Properties dialogs to provide the functionality for the controls.

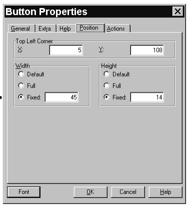


Entry Properties

Tabs: General | Extra | Help | Position | Actions

Picture: @n4
Use: PHO:CustNo
Justification: Default   Offset: 0
Mode: [ ] Hide [ ] Disable [ ] Skip [ ] Scroll

The Entry Properties dialog controls the appearance and functionality of the normal entry boxes which accept user entry.

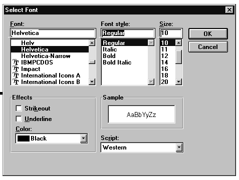


Button Properties

Tabs: General | Extra | Help | Position | Actions

Top Left Corner: X: 5   Y: 108
Width: Default Full [✓ Fixed:] 45
Height: Default Full [✓ Fixed:] 14

To set the location of any control, you can drag it with the mouse in the Window Formatter. For more precise control, use the Position dialog.



Select Font

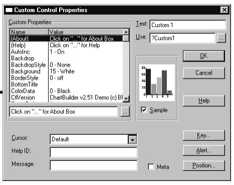
Font: Helvetica
Helv
Helvetica
Helvetica-Narrow
IBMPCDOS
Impact
International Icons A
International Icons B

Font style: Regular
Regular
Italic
Bold
Bold Italic

Size: 10
10
11
12
14
16
18
20

Effects: [ ] Strikeout [ ] Underline
Color: Black
Sample: AaBbYyZz
Script: Western

Set the font for any type of control.



Custom Control Properties

Custom Properties

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
CtlVersion	ChartBuilder v2.51 Demo (c) Bf

Click on "..." for About Box

Text: Custom 1
Use: ?Custom1
[✓] Sample
Cursor: Default
Help ID:
Message:
[ ] Meta

Use .VBX custom control libraries in your application.

This chapter teaches you how to set control properties. It assumes you understand how to use the **Window Formatter** to choose, place and size controls (See the *Using the Window Formatter* chapter). It provides an overview of the types of controls as they relate to data entry, discusses the properties applicable to all 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 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, which 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.
- How to set properties for .VBX custom controls.

When using the **Window Formatter** from within the Application Generator, any fields which accept user entry automatically open the **Select Field** dialog, so that you can indicate a field or variable to associate with the control. Once placed, you can access the control's Properties dialog from the **Edit** menu or from the popup menu.

OVERVIEW: TYPES OF CONTROLS

Controls are generally divided into three categories, **User Interactive Controls**, **Non-User Interactive Controls**, and **Custom Controls**.

User Interactive Controls

User interactive controls are clicked on or typed into by the user.

- Action controls—BUTTON—lead to an instantaneous result. This might involve closing a dialog box and completing the operations contained in it. Clarion also supports associating a continuous action with a button. For the user, this means pressing the button and holding it down is the same as clicking the button repeatedly.
- User Choice controls—RADIO, COMBO, SPIN and LIST—allow the user to enter data by choosing from a group of possible alternatives. No keyboard input is required.
- Entry controls—ENTRY and TEXT—allow data entry from the keyboard. Clarion for Windows provides extensive options for automatically validating user data entry.

Non-User Interactive Controls

Non-user interactive controls provide visual cues that help the user understand and operate the interactive controls.

- Non-user interactive controls—PROMPT, GROUP, BOX, LINE, SHEET, TAB etc.—don't perform an action, but instead guide the user to other controls. They can take the form of a group box, a tab, a line, or a graphic image, all of which visually organize or emphasize other controls.

Custom Controls

Custom controls are defined outside the Clarion Development Environment and may be either interactive or non-interactive.

- Custom controls—.VBX controls—are 'add-in' controls from third party vendors. These may perform a very wide variety of tasks.

SETTING COMMON CONTROL ATTRIBUTES

The attributes you add to a control determine how the control will look and act. Different controls support different functions, and so require different attributes. All Clarion controls allow you to set two common attributes: USE and AT. Additionally, most controls allow you to set KEY, ALRT, FONT, SKIP, HIDE, DISABLE, SCROLL, CURSOR, HLP, MSG, and TIP attributes. This section will explain how to set these common control attributes. Each attribute is discussed more fully in the *Language Reference*.

Setting the USE Attribute

The compiler internally references each control by a number it assigns. To make it easier for *you* to refer to the control in executable source code statements, you may define a "field equate label" for each control, so you can refer to the control by name. The USE attribute specifies this name, which is called a field equate label. A field equate label is a valid Clarion label prefixed by a question mark (?). We suggest using a meaningful label for field equate labels, such as ?EmployeeImage.

For *entry* controls (controls that accept data), the USE attribute is specified *without* the leading question mark (?). For example: EmployeeImage. For entry controls, the USE attribute serves a dual purpose. It not only supplies the field equate label for the control, it also specifies the *name* of the variable that holds the data from the control. This variable must be *defined* in the data dictionary, in one of the data dialogs, or by hand-coding.

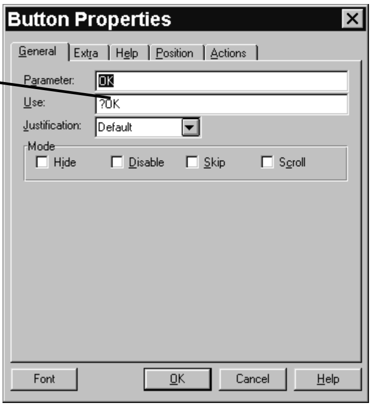
In source code, you refer to this control's data item as EmployeeImage. You refer to the control itself as ?EmployeeImage.

To set the USE attribute:

1. RIGHT-CLICK on the control, and choose **Properties** from the popup menu.

   This displays the **General** tab of the respective control properties dialog, which allows you to specify the USE attribute for your control.

Setting the USE attribute for your control, so you can refer to the control by name in your source code.



Parameter: OK
Use: ?OK
Justification: Default

2. In the **Use** field, for non-entry controls, type a descriptive, valid Clarion label, prefixed by a question mark(?), otherwise, name the variable the control updates (no question mark).

If the control is an entry control (Entry Box, Option Box, Spin Box, Text Box, List Box, Combo Box, Check Box, or Custom Control), press the ellipsis (...) button to choose, or create, a data dictionary field or memory variable from the **Select Field** dialog. Do *not* prefix with a question mark (?). Remember, for entry controls, the USE attribute not only supplies the field equate label for the control, it also specifies the *name* of the variable that holds the data from the control.

Note: Two or more entry controls may update the same variable. However, they must still have unique field equate labels. In this circumstance, the Window Formatter automatically creates unique field equate labels by appending a number to field equate labels that would otherwise be duplicated.

3. Press the **OK** button.

Tip: Field equate labels allow you to use executable statements and Clarion's property access syntax to modify the control at runtime. For example, you can use the DISABLE statement to "dim out" controls in situations when they should be unavailable to the user:

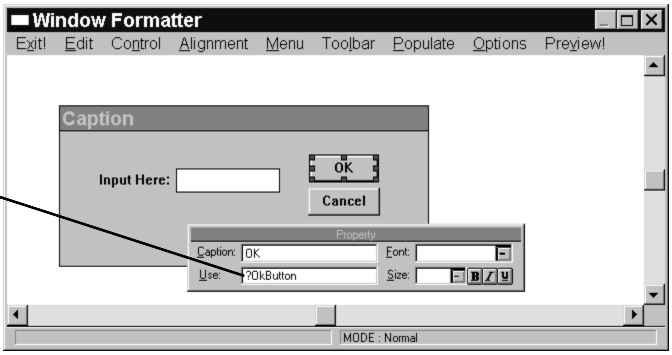
```
DISABLE(?EmployeeImage)
```

Following is an alternative method for setting the USE attribute. This method works best for non-entry controls, because no variable selection or definition is required.

1. From the **Window Formatter** menu, choose **Options** ä **Show Propertybox**.

   This displays the **Property** toolbox, which allows you to specify the USE attribute for your controls.

An alternative for setting the USE attribute.



2. CLICK on the control you want to change.

3. In the **Property** toolbox **Use** field, for non-entry controls, type a descriptive, valid Clarion label, prefixed by a question mark(?), otherwise, name the variable the control updates (no question mark).

   If the control is an entry control (Entry Box, Option Box, Spin Box, Text Box, List Box, Combo Box, Check Box, or Custom Control), do *not* prefix with a question mark (?). Remember, for entry controls, the USE attribute not only supplies the field equate label for the control, it also specifies the *name* of the variable that holds the data from the control. This variable must be *defined* in the data dictionary, in one of the data dialogs, or by hand-coding.

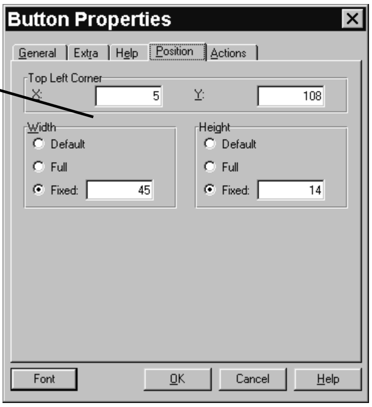
Setting the AT Attribute

The **Window Formatter** allows you to visually set the position and size of each control simply by dragging it wherever you want. You may also specify position by manually typing coordinates in a dialog box. This allows you to exactly line up a control, for example, halfway down a dialog box. To set the AT attribute, which defines the control's position:

1. RIGHT-CLICK on the control, and choose **Position** in the popup menu.

   This displays the **Position** tab of the respective control properties dialog, which allows you to specify the AT attribute for your control.

The Width and Height fields can expand the control to fill the window, or can set a precise size for the control.



Top Left Corner
X: 5
Y: 108

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 Glossary for 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 will automatically select a size based on the display picture on an entry control. To turn off automatic re-sizing, choose **Fixed** and type in 'X' and 'Y' values for the control.

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*.

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, and set the Width and Height of the TEXT control to Full.

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, Prompt, Property Sheet, Spin Box, Tab, Text Field, Button, Check Box, Custom Control, and Radio Button). It specifies a hot key which will give immediate focus to the control. For an action control, such as a command button, the hot key initiates the action. See the *Language Reference* for more information.

To set the KEY attribute:

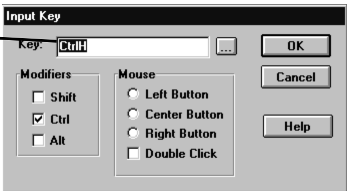
1. RIGHT-CLICK on the control, and choose **Key** in the popup menu.

   This displays the **Input Key** dialog, which allows you to specify the KEY attribute for your control.

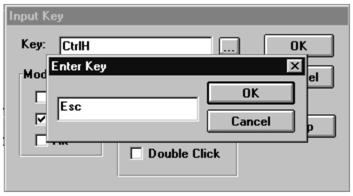
2. Press the desired key or key combination.

   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.

The control will receive focus when the user presses CTRL+H


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 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.



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 allows you to 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, and choose **Alert** in the popup menu.

   This displays 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 displays the **Insert Key** dialog, which allows you to 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 the *Language Reference* for more information.

To set the FONT attribute:

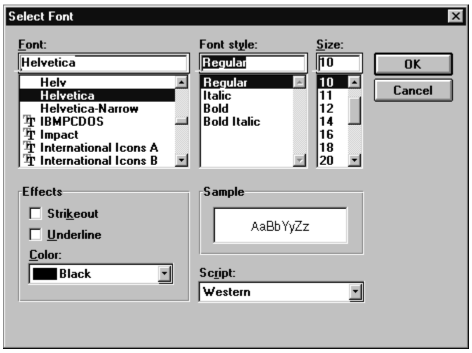
1. RIGHT-CLICK on the control and choose **Font** in the popup menu.

   This displays the **Select Font** dialog.

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.



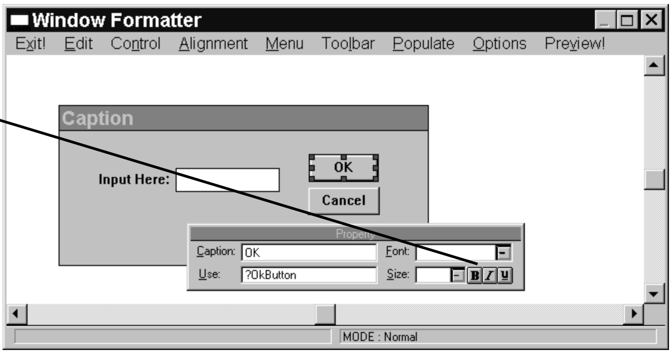
Tip: Be sure the font you pick is present on the user's system. If not, Windows will try to substitute an equivalent font; however, since you have no control over the substitution, you cannot be sure of the result. The TrueType fonts that Microsoft ships with Windows 3.1 are: Arial, Courier New, Symbol, Times New Roman, and Wingdings. In addition to these, Windows 95 ships with Algerian, Army, Braggadocio, Brush Script, Colonna MT, Desdemona, Impact, Matura MT, MS LineDraw, MT Extra, Playbill, and Zingbats. These fonts will be present on many systems.

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 allows you to specify the FONT attribute for your controls.

Setting control font with the Property toolbox.



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

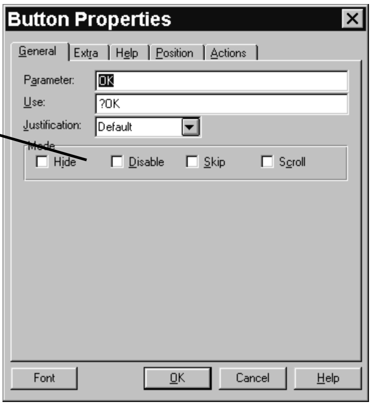
The **General** tab of the various control properties dialogs allows you to set four attributes that control the "mode" (appearance, disappearance, and availability) of your window controls. To set the control's mode:

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.

Setting control modes.



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*).
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*).

4. Press the **OK** button.

Setting Help Attributes

The **Help** tab of the various control properties dialogs allows you to set four attributes that supply information to the user about the control.

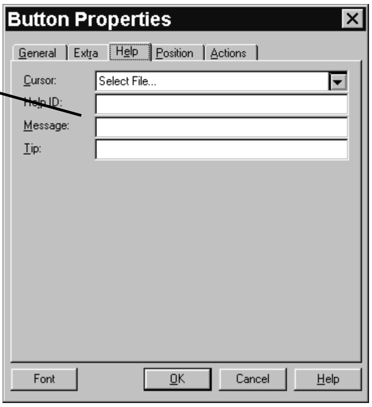
1. RIGHT-CLICK on the control, and choose **Properties** in the popup menu.

   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.

Setting cursor, help, and message attributes.



3. Optionally fill in any of the four entry fields.

   The fields and their effects are:

Cursor	Allows you to 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 third party help 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.

USER INTERACTIVE CONTROLS

Setting 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** allows you to 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.

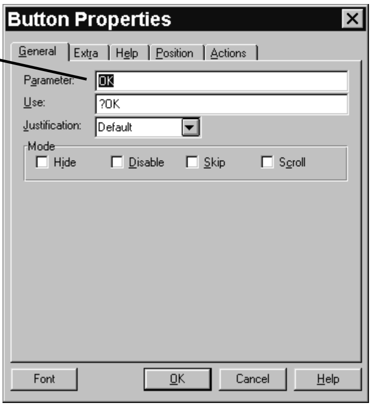
- The *Drop ID* specifying drag and drop operations for which the button is a valid target.

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

Creating an OK button.



Parameter: OK
Use: ?OK

1. In the **Parameter** field, type the text that you wish to appear on the button.

   The text in the **Parameter** field is a string constant. An ampersand (&) within the text means the next character is the accelerator key for the button. The character is underlined and when the user presses ALT + the corresponding key, the button's action initiates. Button text may also be specified in the **Caption** field of the **Property** Toolbox.

Tip: Microsoft recommends you do *not* place an accelerator key on buttons labeled 'OK,' or 'Cancel.'

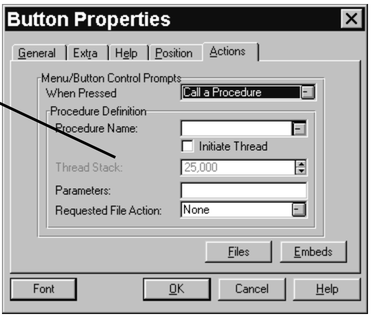
2. In the **Use** field, type a field equate label.

   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 *Setting the USE Attribute* above.

3. **Mode** options: see *Setting the Mode Attributes* above.

Extra Tab

A button that performs a standard windows action.



4. In the **Icon** field, optionally 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.

5. Check the appropriate **Options** boxes.

   There are three button options which you may toggle on or off independently.

Immediate	(the IMM attribute) allows you to 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.
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.

6. 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 will automatically execute any of the standard actions from the drop down list using standard 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 *C:\CW\LIBSRC\EQUATES.CLW* file.

Note: You may not combine a procedure or program call with an STD ID, because a control with an STD ID does not generate an ACCEPT event when the user activates the control.

7. 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. Also see the *Using the List Box Formatter* chapter, *Adding Drag and Drop Capability to the List Box*.

Help Tab

See *Setting Common Control Attributes* above.

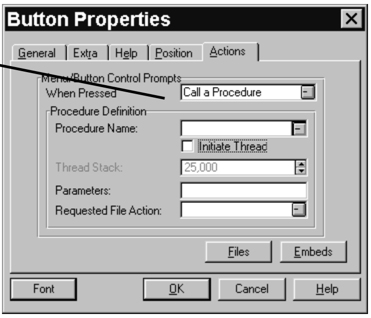
Position Tab

See *Setting the AT Attribute* above.

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 button controls. See the *Using Control, Code, and Extension Templates* chapter for more information. Also see the *Using Control, Code, and Extension Templates* chapter for more information.

Adding functionality to your button.



Menu/Button Control Prompts
When Pressed: Call a Procedure
Procedure Definition
Procedure Name:
☐ Initiate Thread
Thread Stack: 25,000
Parameters:
Requested File Action:

8. 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 will **Initiate a Thread**.
Procedure Name	From the **Procedure Name** drop down list, choose and existing procedure name, or type a new procedure name. A new procedure appears as a “ToDo” item in your Application Tree.
Initiate a Thread	Optionally check the **Initiate a Thread** box. If the procedure initiates a thread, specify the Thread Stack size. Clarion uses the START function to initiate a new execution thread. 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.

Tip: A BUTTON on an application frame toolbar that calls an MDI child procedure <u>must</u> 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 list of variables or data structures passed 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 Global Request variable gets the selected value. The called procedure can then check the value of the Global Request variable 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 program must reside in a .DLL or .LIB defined in your application’s project (.PRJ) file.

Parameters In the **Parameters** field, optionally type a list of values that are passed to the program.

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 (see *Extra Tab* above).

Note: You may combine a procedure or program call with embedded source, but not with an STD ID.

9. Optionally press the **Files** button to access the file schematic for this procedure.
10. Optionally press the **Embeds** button to embed source code at points surrounding the event handling for this button only.
11. Press the **OK** button to return to the **Window Formatter**.

Choice Controls

Choice controls allow user input without keyboard entry. They create streamlined user entry, as it is usually faster to pick an item from a list than to type in a name of an item you may not remember. Choice controls include radio buttons, check boxes, list boxes, combo boxes, and spin boxes.

- Use choice controls to force the user to choose only one of a group of mutually exclusive selections.
- Use choice controls to create special toolbar button effects, such as "latched" buttons that stay depressed until pressed again, or groups of radio buttons where only one button can be selected at a time. See the *Creating Menus and Toolbars* chapter of this book.
- Clarion also allows you to easily incorporate multiple selections from lists. The **List Box Formatter** allows you to design simple list boxes, or multi-column, multi-selection list box controls.

Setting Radio Button Properties

A Radio Button, also called an option button, provides the user a set of mutually exclusive choices. By default, a filled-in circle represents the current selection.

An option box—an OPTION structure—must always surround the radio button choices. Therefore, *in order to set radio button properties, you must also set properties for the option box*. 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.

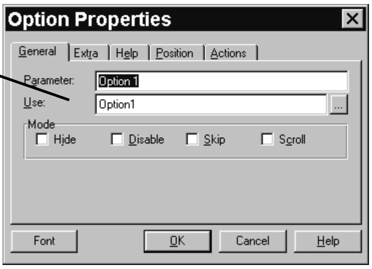
When the user selects a radio button, the OPTION's USE variable receives a value indicating which button was selected: the parameter text of the selected button, the button number, or another value that you specify. Your program can then take appropriate action based on the OPTION's USE variable value.

To place a radio button and an associated option box, activate the Radio Button tool, and CLICK in the sample window. The **Window Formatter** automatically prompts you to create an option box. CLICK on Yes. An option box and one radio button appear.

To set option box properties, RIGHT-CLICK the option box and select **Properties** from the popup menu; the **Option Properties** dialog appears.

General Tab (Option Box)

Setting the Option Properties dialog for a group of mutually exclusive radio buttons.



Parameter: Option 1
Use: Option1

1. In the **Parameter** field, type the Option Box label.

   The **Parameter** field requires a string constant containing the prompt for the group of controls. This string appears at run time in the top border of the option box. An ampersand (&) within the text means the next character is the accelerator key for the control. The character is underlined, and when the user presses ALT + the corresponding key, the first radio button receives focus. This text may also be specified in the **Caption** field of the **Property** Toolbox.

Tip: Though the OPTION structure *must* be present, it does not have to appear on screen. You may hide it from the user by un-checking the 'Boxed' box on the 'Extra' tab of this dialog.

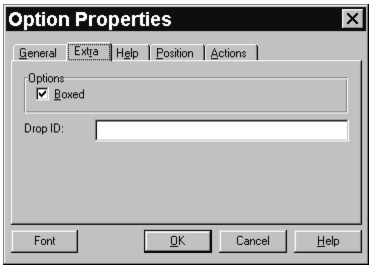
2. In the **Use** field, type the label of a variable.

   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 parameter text of the selected button, or another string value that you specify (see *General Tab* below). When the USE variable is a numeric data type, it receives the button number.

3. Check any combination of the **Mode** boxes.

   See *Setting the Mode Attributes* above.

Extra Tab (Option Box)



☑ Boxed
Drop ID:

4. Optionally, uncheck the **Boxed** box to hide the option box from the user at run time.

   This produces a slightly different effect than the HIDE attribute. The HIDE attribute hides the option box *and* the controls inside the box. Unchecking the **Boxed** box hides the option box and its caption, but does *not* hide the controls inside.

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 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. Also see the *Using the List Box Formatter* chapter, *Adding Drag and Drop Capability to the List Box*.

Help Tab (Option Box)

See *Setting Common Control Attributes* above.

Position Tab (Option Box)

See *Setting the AT Attribute* above.

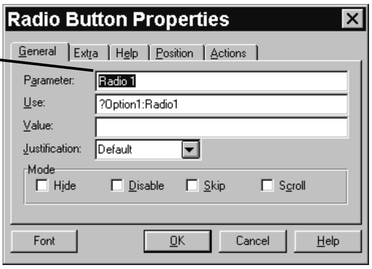
6. Press the **OK** button to finish setting attributes for the OPTION box.

   The OPTION box appears in the sample window. If you un-checked 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; the **Radio Button Properties** dialog appears.

General Tab (Radio Button)

A radio button whose text will say "Radio 1."



Parameter: Radio 1
Use: ?Option1:Radio1
Value:
Justification: Default

7. In the **Parameter** field, type the Radio Button text.

   The **Parameter** field requires a string constant containing the prompt for the radio button. An ampersand (&) within the text means the next character is the accelerator key for the control. The character is underlined, and when the user presses ALT + the corresponding key, the radio button receives focus. This text may also be specified in the **Caption** field of the **Property** Toolbox.

8. 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 *Setting the USE Attribute* above.

9. 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 **Parameter** 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.

10. From the **Justification** drop down list, choose ***Left Justification, Right Justification***, or ***Default***.

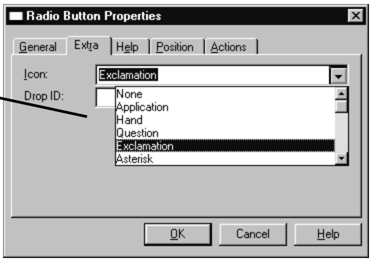
    Left Justification arranges the button (or icon) to the left of the parameter text. Right Justification arranges the button (or icon) to the right of the parameter text. Default arranges the button according to any applicable settings in the data dictionary.

11. Check any combination of the **Mode** boxes.

    See *Setting the Mode Attributes* above.

Extra Tab (Radio Button)

Choosing a Radio Button Icon.



12. 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. See the *Creating Menus and Toolbars* chapter for details.

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.

13. 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. Also see the *Using the List Box Formatter* chapter, *Adding Drag and Drop Capability to the List Box*.

Help Tab (Radio Button)

See *Setting Common Control Attributes* above.

Position Tab (Radio Button)

See *Setting the AT Attribute* above.

14. Optionally add additional radio buttons by placing more RADIO's inside the OPTION structure.

Tip: To create professional looking radio button groups, turn the Grid control on and use the Alignment tools. Grid Settings appears on the Window Formatter's Options menu. This will allow you to easily line up your radio buttons.

Setting Check Box Properties

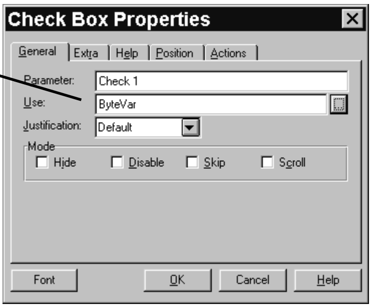
A check box manages a variable that the user may turn on or off. Activate the Check Box tool, or choose **Check Box** from the **Control** menu, then click in the sample window. The **Window Formatter** automatically opens the **Select Field** dialog, so that you can choose or create a data dictionary field or memory variable to associate with the check box.

Tip: Use a BYTE data type variable with your check boxes to avoid any unnecessary data type conversions.

Once placed, RIGHT-CLICK the check box and select **Properties** from the popup menu; the **Check Box Properties** dialog appears.

General Tab

The Check Box Properties dialog.


Check Box Properties

Parameter: Check 1
Use: ByteVar
Justification: Default
Mode: Hide, Disable, Skip, Scroll

1. In the **Parameter** field, type the text that you wish to appear on the check box.

The text in the **Parameter** field is a string constant. An ampersand (&) within the text means the next character is the accelerator key for the check box. The character is underlined and when the user presses ALT + the corresponding key, the check box receives focus. Check box text may also be specified in the **Caption** field of the **Property** Toolbox.

2. The **Use** field should already contain a variable label.

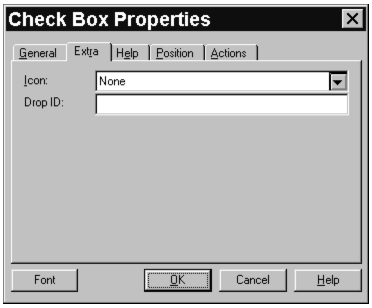
   If not, type the label of a variable, or press the ellipsis button to choose or create a data dictionary field or a memory variable with the **Select Field** dialog.

3. From the **Justification** drop down list, choose ***Left Justification, Right Justification***, or ***Default***.

   Left Justification arranges the check box (or icon) to the left of the parameter text. Right Justification arranges the check box (or icon) to the right of the parameter text. Default arranges the check box according to any applicable settings in the data dictionary.

4. **Mode** options: see *Setting the Mode Attributes* above.

Extra Tab



5. In the **Icon** field, optionally select a standard icon from the drop down list.

   Adding an icon to a check box makes the check box look like a command button.

   This displays a small bitmap on the button face in addition to any check box 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: If you add an icon and text, you get a check box with both! Make the resulting button large enough to display both.

6. 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. Also see the *Using the List Box Formatter* chapter, *Adding Drag and Drop Capability to the List Box*.

Help Tab

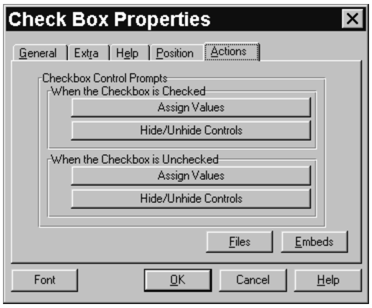
See *Setting Common Control Attributes* above.

Position Tab

See *Setting the AT Attribute* above.

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 check box controls. See the *Using Control, Code, and Extension Templates* chapter for more information. Also see the *Using Control, Code, and Extension Templates* chapter for more information.



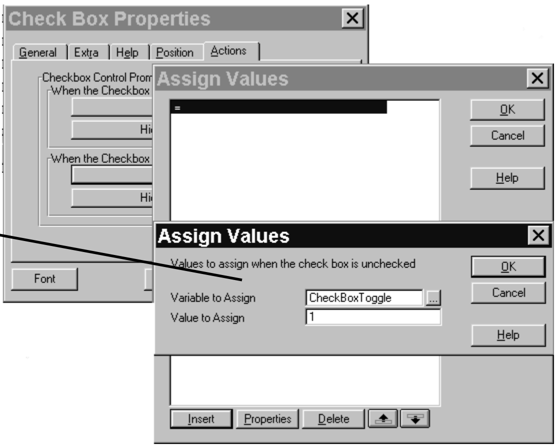
The **Actions** tab leads to dialogs allowing you to name variables (other than the USE variable) and change their values when the end user checks or unchecks the box. Additionally, you can hide or unhide other controls in the window.

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**.

6. 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. You can then add code to your program to take appropriate action based on the run time value of the variable(s). Press the **OK** button to end the dialogs.



Assign Values

Values to assign when the check box is unchecked

Variable to Assign: CheckBoxToggle

Value to Assign: 1

Setting toggling actions for a check box.

7. 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** entry box, type the control's equate label, or press the ellipsis (...) button to select from a list of control equate labels.

   In the **Hide or unhide control** entry box, select **Hide** or **Unhide**. Press the **OK** button to end the dialogs.

8. Optionally press the **Files** button to access the **File Schematic Definition** dialog for this procedure.

9. Optionally press the **Embeds** button to embed source code at points surrounding the event handling for this check box only.

10. Press the **OK** button to return to the **Window Formatter**.

Creating List Boxes

The LIST control is most useful for presenting a great number of choices to the user. It can convey a large amount of data in a minimal area, which has led to its use as an all-purpose data control. Using Clarion for Windows, you can create list boxes which look like spreadsheet grids, perform drag and drop tasks, and more.

When creating a list box, you define its data source, its format, and its functionality. The development environment divides these property definitions among several dialogs:

- The **List Properties** dialog specifies the file or queue that supplies the list data, a drop down list versus a regular list, and the general scrolling capability. 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 (Populate or Insert), delete, reorder, and resize the specific fields or columns that are displayed in the list box. This tool is discussed in the following chapter.
- The **List Field Properties** dialog is part of the **List Box Formatter**, and defines the appearance and behavior of individual list box columns. For example, you can define column headers, width, and individual column scrolling. This dialog is discussed in the following chapter.
- The **List Field Properties** dialog also defines the appearance and behavior of *groups* of columns within the list box. For example, spreading a header across several columns.

Setting List Properties

1. From the **Window Formatter**, select the List Box tool, or choose **List Box** from the **Control** menu, then click in the window.

   The **List Box Formatter** dialog appears. This dialog manages the columns or fields in your list box and is discussed in detail in the following chapter.

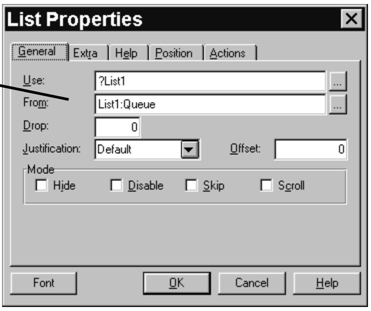
2. Press the **OK** button to return to the **Window Formatter.**

3. RIGHT-CLICK the list box and select **Properties** from the popup menu.

   The **List Properties** dialog appears.

General Tab

The List Properties dialog.



Use: ?List1
From: List1:Queue
Drop: 0
Justification: Default
Offset: 0
Mode
Hide
Disable
Skip
Scroll

4. In the **Use** field, type a descriptive, valid Clarion label, prefixed by a question mark(?), or, name the variable the list box updates (no question mark).

   The variable receives the value the user selects from the list. You can press the ellipsis (...) button to choose, or create, a data dictionary field or memory variable from the **Select Field** dialog to specify a variable name. Do *not* prefix with a question mark (?). Remember, for entry controls, the USE attribute not only supplies the field equate label for the control, it also specifies the *name* of the variable that holds the data from the control.

Note: Two or more entry controls may update the same variable. However, they must still have unique field equate labels. In this circumstance, the Window Formatter automatically creates unique field equate labels by appending a number to field equate labels that would otherwise be duplicated.

5. 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 define and load the QUEUE.

6. 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 box. To create a drop down list box, type the number of drop elements you wish to display.

7. From the **Justification** drop down list, choose ***Left***, ***Center***, ***Right***, ***Decimal***, or ***Default*** justification.

Adds the LEFT, CENTER, RIGHT, or DECIMAL attribute to the LIST. See the *Language Reference* for details. ***Left***, ***Center***, and ***Right*** 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. See *Offset* below.

These attributes are superseded by the FORMAT attribute, which is added by Clarion Browse templates. That is, if you use the **List Box Formatter** to populate your list box, the positioning of data will be determined by the Clarion generated FORMAT string, and not by these justification attributes.

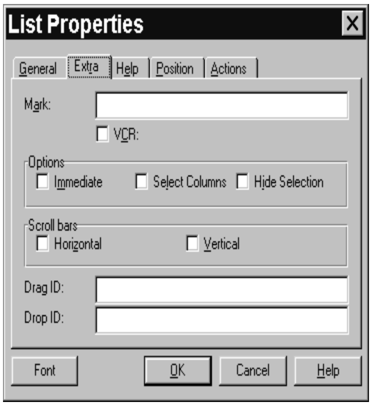
8. In the **Offset** field, specify a justification offset in dialog units.

   See the Glossary for definition of dialog units. Sets the offset value for the LEFT, RIGHT, CENTER, and DECIMAL attributes. See above. 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.

9. Check any combination of the **Mode** boxes.

   See *Setting the Mode Attributes* above.

Extra Tab



10. In the **Mark** field, type in the name of a QUEUE, or QUEUE field if you wish to allow the user to select more than one item from the list.

    The QUEUE field flags the selected items. Selected items get a ‘1’, unselected items get a ‘0.’

11. Check the **VCR** checkbox to provide VCR scrolling buttons for your list box.

These special scrolling buttons include the following: Top of List (|<), Page Up (<<), Entry Up (<), Locator (?), Entry Down (>), Page Down (>>), and Bottom of List (>|).

The entry box to the right of the **VCR** checkbox is used in conjunction with the Locator button. Optionally type the field equate label of an entry control in this entry box. When the user presses the *Locator* (?) button, focus shifts to the entry control identified by the equate label. The user may type in data, then press TAB to scroll the LIST to the closest matching entry.

12. 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 while the list box has focus. This feature allows you to display related information as the user scrolls or highlights new selections.

13. 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.

14. 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.

15. 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**, which is described in the next chapter.

16. 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. Also see the *Using the List Box Formatter* chapter, *Adding Drag and Drop Capability to the List Box*.

17. 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. Also see the *Using the List Box Formatter* chapter, *Adding Drag and Drop Capability to the List Box*.

Help Tab

See *Setting Common Control Attributes* above.

Position Tab

See *Setting the AT Attribute* above.

Actions Tab

The **Actions** tab will be blank unless you are using a control template to build your list box. See the Using Control, Code, and Utility Templates chapter for instructions on the **Actions** prompts for each control template.

Setting Combo Box Properties

The COMBO control combines an entry box with a list box. It is useful for when you expect string data which *usually* is a member of the list, but sometimes is not. The **Window Formatter** allows you to create a normal combo box, or a drop combo box.

Combo Box properties are set exactly like List Box properties except for the following four additional properties.

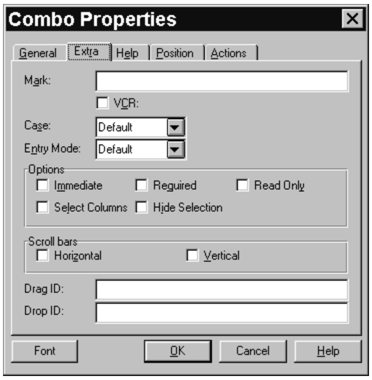
General Tab

1. In the **Picture** field, specify the picture token for the control.

Pressing the ellipsis button allows you to select the picture token from the **Edit Picture String** dialog .

The picture token forces the input data into a specific format. For example, a picture token of @P##/##/##P forces a typical date format.

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. **Capitals** (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 ***Insert***, ***Overwrite***, or ***As Is***.

   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. **As Is** causes each keystroke to behave according to current system settings.

   The **Entry Mode** applies only for windows with the MASK attribute set. See *Using the Window Formatter; Using the Window Properties Dialog; Entry Patterns* for more information.

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*).

Setting Spin Box Properties

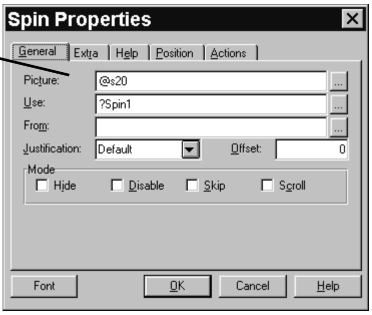
A SPIN control is a specialized entry box that only accepts values in a predefined range. They also provide the user with 'increase' and 'decrease' buttons, which many people like because they can use the mouse to change the value. The user can also type a value directly into the control.

1. From the **Window Formatter**, select the Spin Box tool, or choose **Spin Box** from the **Control** menu, then click in the window.
2. RIGHT-CLICK the spin box and select **Properties** from the popup menu.

   The **Spin Properties** dialog appears.

General Tab

The Spin Properties dialog.



Picture: @s20
Use: ?Spin1
From:
Justification: Default
Offset: 0
Mode
☐ Hide ☐ Disable ☐ Skip ☐ Scroll

1. In the **Picture** field, type a picture token.

   The Picture field takes a display picture token that specifies input format. You may press the ellipsis (...) button next to the field to pick a display picture from the **Edit Picture String** dialog.

   You may check the user entry against the picture at two points: as the user types the data in, or when the user closes the dialog box. 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 for the window in which the entry box resides. This turns the MASK attribute on for all controls in the window.

   If the MASK attribute is off, entry checking takes place when the user moves the focus to another control (for example, by TABBING to another field).

If the user types in data in a format different from the picture, the program will attempt to determine the format, then convert it to match the picture (if no MASK attribute was specified). For example, if the user types 'January 1, 1995' and the picture is @D1, the program formats the input to "1/1/95. If the program cannot determine the entry format, it will not update the USE variable. The user will receive an audible prompt (beep), and the focus will return to the entry control, ready for additional input.

2. In the **Use** field, type a variable label.

   The variable receives the value the user selects from the list. The same label, prefixed by a question mark (?) is the field equate label that references the spin box in source code statements.

3. In the **From** field, supply the data source for the spin box.

   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** limits fields are mutually exclusive.

4. From the **Justification** drop down list, choose ***Left***, ***Center***, ***Right***, ***Decimal***, or ***Default*** justification.

   Adds the LEFT, CENTER, RIGHT, or DECIMAL attribute to the LIST. See the *Language Reference* for details. ***Left***, ***Center***, and ***Right*** 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 the values on their decimal points. Each justification may be offset by a distance you specify. See *Offset* below.

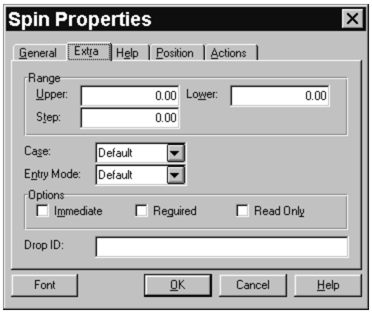
5. In the **Offset** field, specify a justification offset in dialog units.

   See the Glossary for definition of dialog units. Sets the offset value for the LEFT, RIGHT, CENTER, and DECIMAL attributes. See above. 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.

6. Check any combination of the **Mode** boxes.

   See *Setting the Mode Attributes* above.

Extra Tab



7. Specify the upper and lower **Range** limits, 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.

8. 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. **Capitals** (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.

9. In the **Entry Mode** drop down list, choose ***Insert, Overwrite***, or ***As Is***.

   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. ***As Is*** causes each keystroke to behave according to current system settings.

   The **Entry Mode** applies only for windows with the MASK attribute set. See *Using the Window Formatter; Using the Window Properties Dialog; Entry Patterns* for more information.

10. Check the appropriate **Options** boxes.

    There are three option flags you may toggle on or off independently.

Immediate	(the IMM attribute) specifies immediate event generation whenever the user presses any key.

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.

11. 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. Also see the *Using the List Box Formatter* chapter, *Adding Drag and Drop Capability to the List Box*.

Help Tab

See *Setting Common Control Attributes* above.

Position Tab

See *Setting the AT Attribute* above.

Setting 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, or an initial value.

- 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:

1. From the **Window Formatter**, select the Entry Box tool, or choose **Entry Box** from the **Control** menu, then click in the window.

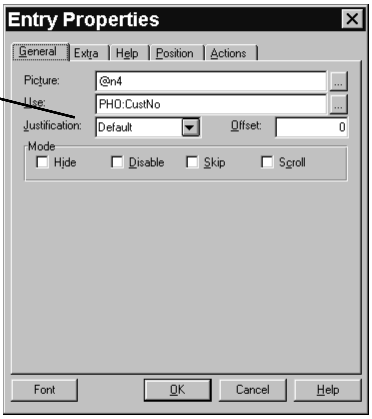
   The **Select Field** dialog appears allowing you to choose or create a data dictionary field or a memory variable to hold the data input to this control.

2. RIGHT-CLICK the Entry Box and select **Properties** from the popup menu.

   The **Entry Properties** dialog appears.

General Tab

The Entry Properties dialog.



Picture: @n4
Use: PHO:CustNo
Justification: Default
Offset: 0

1. Specify a **Picture**.

   The **Picture** field is a display picture token that specifies the input format for the data entered into the control. For example, you may specify a string of 20 characters as @s20. You may press the ellipsis (...) button next to the field to pick a display picture from the picture formatter.

   You may check the user entry against the picture at two points: as the user types the data in, or when the user closes the dialog box. 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 for the window in which the entry box resides. This turns the MASK attribute on for *all* controls in the window.

   If the MASK attribute is off, entry checking takes place when the user moves the focus to another control (for example, by TABBING to another field).

If the user types in data in a format different from the picture, the program will attempt to determine the format, then convert it to match the picture (if no MASK was specified). For example, if the user types ‘January 1, 1995’ and the picture is @D1, the program formats the input to “1/1/95. If the program cannot determine the entry format, it will *not* update the USE variable. The user will receive an audible prompt (beep), and the focus will return to the entry control, ready for additional input.

2 Specify a **Use** attribute.

Type the name of the variable to receive the value that the user will enter in the control.

3. From the **Justification** drop down list, choose ***Left***, ***Center***, ***Right***, ***Decimal***, or ***Default*** justification.

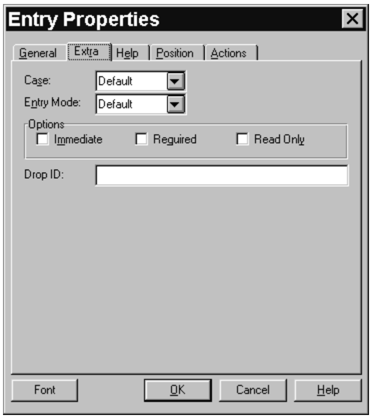
Adds the LEFT, CENTER, RIGHT, or DECIMAL attribute to the LIST. See the *Language Reference* for details. ***Left***, ***Center***, and ***Right*** position the entry data predictably, left, center, or right justified in the entry box. ***Default*** positions the data according to any applicable settings in the data dictionary. ***Decimal*** justification aligns values by their decimal points. Each justification may be offset by a distance you specify. See *Offset* below.

4. In the **Offset** field, specify a justification offset in dialog units.

See the Glossary for definition of dialog units. Sets the offset value for the LEFT, RIGHT, CENTER, and DECIMAL attributes. See above. 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.

5. **Mode** options: see *Setting the Mode Attributes* above.

Extra Tab



Case: Default
Entry Mode: Default
Options
☐ Immediate ☐ Required ☐ Read Only
Drop ID:

6. 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. **Capitals** (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.

7. Optionally specify an **Entry Mode**.

   Choose either **Insert** (the INS attribute) or **Overwrite** (the OVR attribute). The **Entry Mode** applies only for windows with the MASK attribute set.

8. Set the Option flags.

   There are three option flags you may toggle on or off independently.

Immediate	(the IMM attribute) specifies immediate event generation whenever the user presses any key.
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.

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. Also see the *Using the List Box Formatter* chapter, *Adding Drag and Drop Capability to the List Box*.

Help Tab

See *Setting Common Control Attributes* above.

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 *Setting the AT Attribute* above.

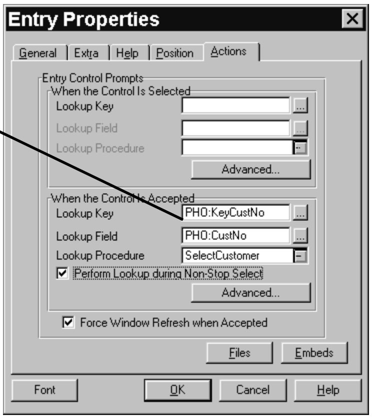
Actions Tab

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 *Using 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.

Specifying a lookup validation procedure for an entry field.



Lookup Key: PHO:KeyCustNo
Lookup Field: PHO:CustNo
Lookup Procedure: SelectCustomer
☑ Perform Lookup during Non-Stop Select
☑ Force Window Refresh when Accepted

10. 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 CUS:KeyCustNumber.

    The **Select Key** dialog allows you to 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.

11. 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 CUS:CustNumber.

12. 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 (or 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 SelectCustomer.

    Based on the previous three entries Clarion builds the following validation code into your procedure:

```
LookupField = EntryField              !get user’s entry
GET(LookupFile,LookupKey)             !search lookup file
IF ERRORCODE()                        !if user entry not found...
   GlobalRequest = SelectRecord       !ask user to select
   LookupProcedure                    !from this lookup procedure
   LocalResponse = GlobalResponse     !get procedure’s response
   IF LocalResponse = RequestCompleted !if user did select...
```

```
        EntryField = LookupField      !put selection in entry box
        DISPLAY(?EntryField)          !refresh the screen
      ELSE                            !if user did not select...
        SELECT(?EntryField)           !give focus to entry box
        CYCLE                         !give control back to user
      END
  END
```

13. 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, and further customization, press the **Embeds** button (see *Embeds* below). Also see *Defining Embedded Source Code* in the *Using the Application Generator* chapter.

14. 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.

15. 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.

16. Optionally press the **Files** button to access the **File Schematic Definition** dialog for this procedure.

17. Optionally press the **Embeds** button to embed source code at points surrounding the event handling for this check box only.

18. Press the **OK** button to return to the **Window Formatter**.

Setting Text Control Properties

The TEXT control provides a multi-line data entry field. This control is especially suitable for holding a long string.

To set the properties for a text box:

1. From the **Window Formatter**, select the Text Box tool, or choose **Text Box** from the **Control** menu, then click in the window.

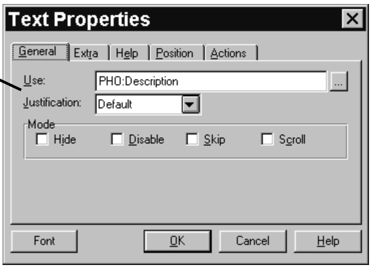
The **Select Field** dialog appears allowing you to choose or create a data dictionary field or a memory variable to hold the data input to this control.

2. RIGHT-CLICK the text box and select **Properties** from the popup menu.

   The **Text Properties** dialog appears.

General Tab

The Text Properties dialog.



3. Specify a **Use** attribute.

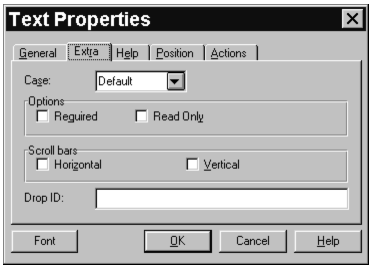
   Type the name of the variable to receive the value that the user enters in the control. When using multi-line controls, be sure the variable is large enough to hold the amount of data you expect your users to enter in the control.

4. From the **Justification** drop down list, choose ***Left***, ***Center***, ***Right***, ***Decimal***, or ***Default*** justification.

   Adds the LEFT, CENTER, RIGHT, or DECIMAL attribute to the LIST. See the *Language Reference* for details. ***Left***, ***Center***, and ***Right*** 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.

5. **Mode** options: see *Setting the Mode Attributes* above.

Extra Tab



6. 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. **Default** (no attribute) accepts input in the case the user types it.

7. Set the Option flags.

   There are two 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.

8. Optionally, check the **Horizontal** or **Vertical** boxes to add scroll bars to your text box.

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. Also see the *Using the List Box Formatter* chapter, *Adding Drag and Drop Capability to the List Box*.

Help Tab

See *Setting Common Control Attributes* above.

Tip: The I-Beam, which signals text entry, is the standard choice for the active cursor for a text control.

Position Tab

See *Setting the AT Attribute* above.

NON USER INTERACTIVE CONTROLS

Non user 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.

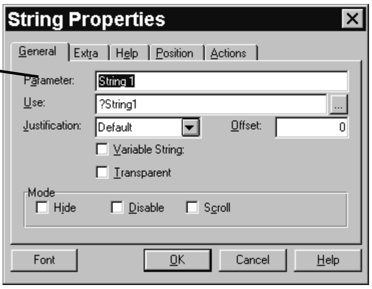
Setting String Control Properties

The String control allows you to place a string constant on screen. It optionally allows you to substitute a variable.

The **String Properties** dialog contains the following options.

General Tab

The String Properties dialog.



Parameter: String 1
Use: ?String1
Justification: Default
Offset: 0
☐ Variable String
☐ Transparent
Mode: ☐ Hide ☐ Disable ☐ Scroll

1. In the **Parameter** field, type the string constant, or a picture token.

   A string constant is displayed as typed. A picture token is used to format a variable string for display.

2. In the **Use** field, type a field equate label, or name a variable for display.

   Type a field equate label to reference the control in executable code.

3. From the **Justification** drop down list, choose ***Left***, ***Center***, ***Right***, ***Decimal***, or ***Default*** justification.

Adds the LEFT, CENTER, RIGHT, or DECIMAL attribute to the LIST. See the *Language Reference* for details. ***Left***, ***Center***, and ***Right*** 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. See *Offset* below.

4. In the **Offset** field, specify a justification offset in dialog units.

   See the Glossary for definition of dialog units. Sets the offset value for the LEFT, RIGHT, CENTER, and DECIMAL attributes. See above. 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.

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 **Parameter** field, such as @s24.

6. Specify whether you wish the control background to be **Transparent**.

   This instructs Windows to suppress the rectangular region around the text — the background. Normally, Windows will paint this the same uniform color as the window below the control,

Tip: When you place text on top of an IMAGE, or a colored graphic such as a BOX, turn the TRN attribute on, so that the text doesn’t obscure the graphic.

7. **Mode** options: see *Setting the Mode Attributes* above.

Extra Tab

8. 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. Also see the *Using the List Box Formatter* chapter, *Adding Drag and Drop Capability to the List Box*.

Help Tab

See *Setting Common Control Attributes* above.

Position Tab

See *Setting the AT Attribute* above.

Setting Prompt Control Properties

The PROMPT control allows you to place a string on screen which will automatically provides an accelerator key to the next active control following the prompt. It is almost identical to the STRING control, except that it has no variable capability. See *Setting String Properties* above.

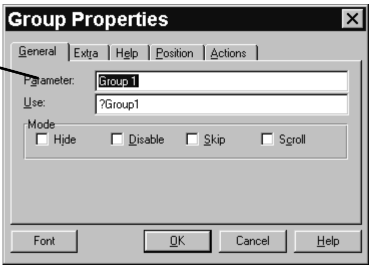
Setting Group Box Control Properties

A GROUP control places a box around two or more controls. It visually associates the controls for the user, and *allows you to 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

The Group Properties dialog.



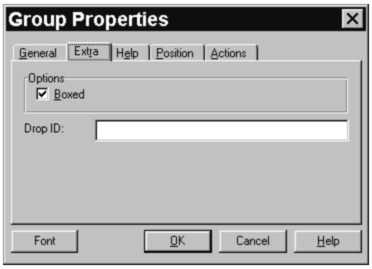
Parameter: Group 1
Use: ?Group1
Mode
☐ Hide ☐ Disable ☐ Skip ☐ Scroll

1. In the **Parameter** field, type a string constant .

The **Parameter** field requires a string constant containing the prompt for the group of controls. This string appears at run time in the top border of the group box. An ampersand (&) within the text means the next character is the accelerator key for the group. The character is underlined, and when the user presses ALT + the corresponding key, the first control in the group receives focus. This text may also be specified in the **Caption** field of the **Property** Toolbox.

2. In the **Use** field, type a field equate label.
3. **Mode** options: see *Setting the Mode Attributes* above.

Extra Tab



4. Optionally specify that the group box should be invisible.

   By un-checking the **Boxed** box, you may make the group box and its parameter text invisible to the user. The group box will be visible in the **Window Formatter**, but invisible in **Preview!** mode.

5. 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. Also see the *Using the List Box Formatter* chapter, *Adding Drag and Drop Capability to the List Box*.

Help Tab

See *Setting Common Control Attributes* above.

Position Tab

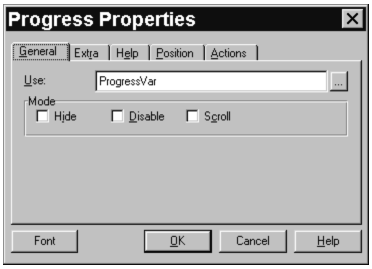
See *Setting the AT Attribute* above.

Setting 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


Progress Properties

General | Extra | Help | Position | Actions

Use: ProgressVar

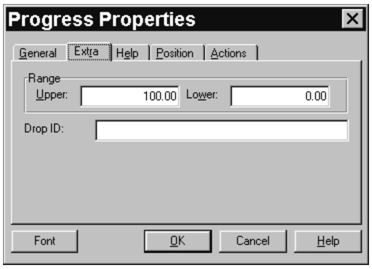
Mode
- [ ] Hide
- [ ] Disable
- [ ] Scroll

1. In the **Use** field, type a field equate label.

   The field equate label references the progress bar in program statements.

   If a variable is named as the USE attribute, the progress bar is automatically updated whenever the value in that variable changes. If the USE attribute is a field equate label, you must directly update the display by assigning a value (within the range defined by the RANGE attribute) to the control's PROP:progress property.
2. **Mode** options: see *Setting the Mode Attributes* above.

Extra Tab



Range
Upper: 100.00 Lower: 0.00
Drop ID:

3. Specify the **Range** of values the progress bar displays.

   If omitted, the default range is from zero (0) to one hundred (100).

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 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. Also see the *Using the List Box Formatter* chapter, *Adding Drag and Drop Capability to the List Box*.

Help Tab

See *Setting Common Control Attributes* above.

Position Tab

See *Setting the AT Attribute* above.

Setting Image Control Properties

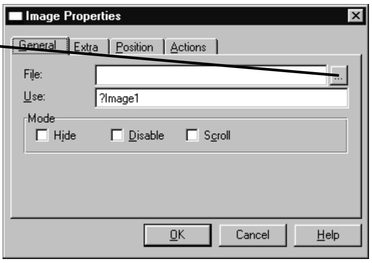
The IMAGE control allows you to place bitmapped and vector images in a window. The bitmap file formats supported are .BMP, .PCX, .GIF, .ICO and .JPG. The vector file format supported is .WMF. Clarion for Windows can support up to 16.7 million color resolution.

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 contains the following options.

General Tab

Press the ellipsis button to choose an image file from an Open File dialog.



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.

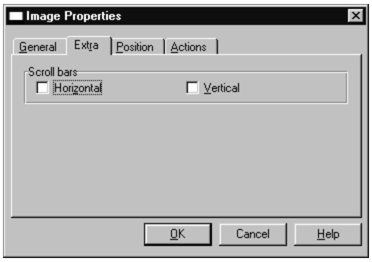
Tip: Bitmaps can take up *lots* of memory. If your application utilizes many bitmaps, test it, and monitor the free memory after displaying large bitmaps.

2. Place a field equate label in the **Use** field.

   The field equate label references the image in program statements.

3. **Mode** options: see *Setting the Mode Attributes* above.

Extra Tab



4. Optionally add scroll bars.

   By checking the **Horizontal** and/or **Vertical** check boxes, you may specify that you wish the control to automatically add scroll bars if the image is larger than the control size.

Position Tab

See *Setting the AT Attribute* above.

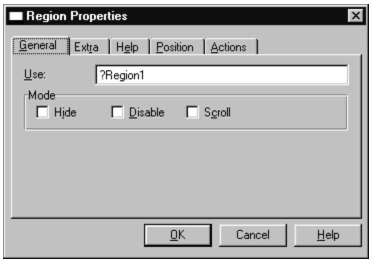
Tip: For IMAGE controls, Default displays the picture at the size it was created.

Setting Region Control 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, or provide for a cursor change when the user passes the mouse over the region.

General Tab

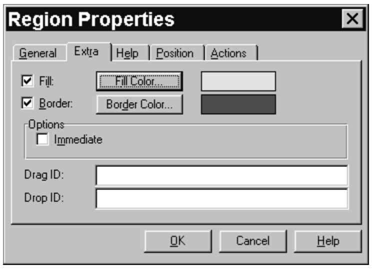

Use: ?Region1

1. Place a field equate label in the **Use** field.

   The field equate label references the region in program statements.

2. **Mode** options: see *Setting the Mode Attributes* above.

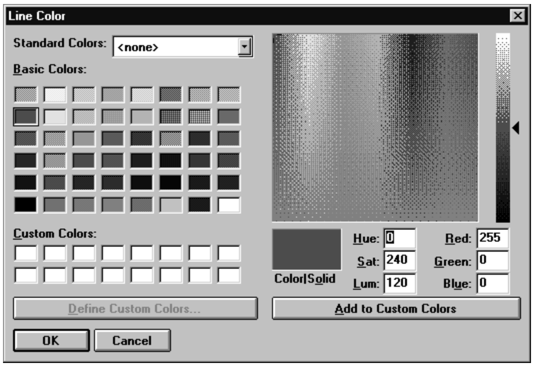
Extra Tab



☑ Fill:
☑ Border:
☐ Immediate
Drag ID:
Drop ID:

3. Specify the **Fill** and **Border** colors.

   Check the **Fill** and **Border** check boxes first—unless you wish either of these elements to be transparent. Then press the **Fill Color...** or **Border Color...** buttons. The standard **Color** dialog appears. Select a color by clicking on the color selection square, or add a custom color. A color sample is displayed beside each button.



Standard Colors: <none>
Hue: 0
Sat: 240
Lum: 120
Red: 255
Green: 0
Blue: 0

4. You may add the IMM attribute to the Region control by checking the **Immediate** checkbox.

   This allows you to monitor whenever the user passes the mouse over the region; however, it incurs much overhead at runtime, so should be used sparingly.

5. 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. Also see the *Using the List Box Formatter* chapter, *Adding Drag and Drop Capability to the List Box*.

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. See the *Language Reference* for more details and examples. Also see the *Using the List Box Formatter* chapter, *Adding Drag and Drop Capability to the List Box*.

Help Tab

See *Setting Common Control Attributes* above.

Position Tab

See *Setting the AT Attribute* above.

Setting Line Control Properties

The LINE control allows you to place a straight line in your windows. You may choose a line color.

The **Line Properties** dialog contains the following options.

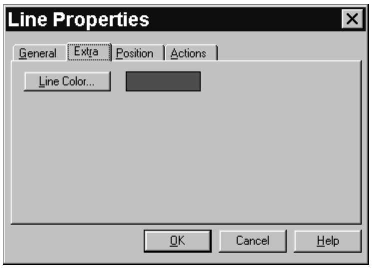
General Tab

1. Place a field equate label in the **Use** field.

   The field equate label references the line in program statements.

2. **Mode** options: see *Setting the Mode Attributes* above.

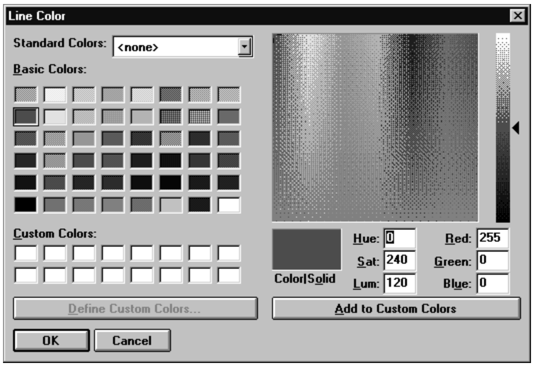
Extra Tab



3. Specify the **Line Color**.

   Press the **Line Color...** button. The standard *Color* dialog will appear. Select a color by clicking on the color selection square, or add a custom color. A color sample is displayed beside the button.

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 *Setting the AT Attribute* above.

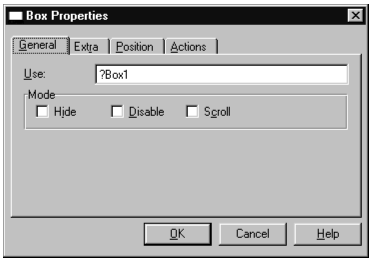
Setting Box Control Properties

The Box control allows you to 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

Mode

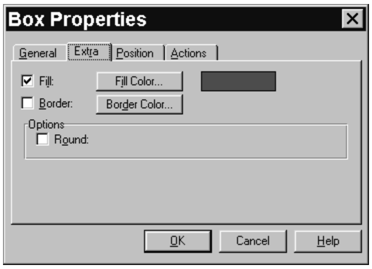
☐ Hide ☐ Disable ☐ Scroll

1. Place a field equate label in the **Use** field.

   The field equate label references the Box in program statements.

2. **Mode** options: see *Setting the Mode Attributes* above.

Extra Tab



☑ Fill:

☐ Border:

Options

☐ Round:

3. Specify the **Fill** and **Border** colors.

   Check the **Fill** and **Border** check boxes first—unless you wish either of these elements to be transparent. Then press the **Fill Color...** or **Border Color...** buttons. The standard **Color** dialog appears. Select a color by clicking on the color selection square, or add a custom color. A color sample displays beside each button.

4. Optionally specify the Box should appear as a rounded Box by marking the checkbox.

   The corners of the box will be rounded.

Position Tab

See *Setting the AT Attribute* above.

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.

Setting Ellipse Control Properties

The ELLIPSE control allows you to 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.
2. **Mode** options: see *Setting the Mode Attributes* above.

Extra Tab

3. Specify the **Fill** and **Border** colors.

   Check the **Fill** and **Border** check boxes first—unless you wish either of these elements to be transparent. Then press the **Fill Color...** or **Border Color...** buttons. The standard **Color** dialog appears. Select a color by clicking on the color selection square, or add a custom color. A color sample displays beside each button.

Position Tab

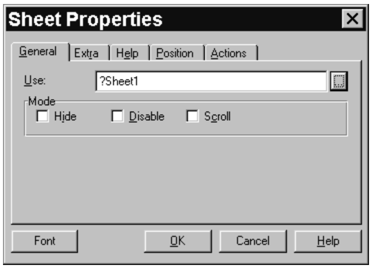
See *Setting the AT Attribute* above.

Tip: While you can set the size of the ellipse and other graphic controls by manually typing in coordinates, it is *much* easier to draw it directly in the Window Formatter.

Setting Property 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



Use: ?Sheet1

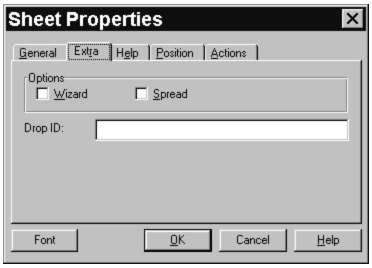
Mode: ☐ Hide ☐ Disable ☐ Scroll

1. Place a field equate label in the **Use** field.

   The field equate label references the property sheet in program statements.

2. **Mode** options: see *Setting the Mode Attributes* above.

Extra Tab



Options: ☐ Wizard ☐ Spread

Drop ID:

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". See *How to Create a Wizard* in the on-line help. Also see the CW\EXAMPLES\APPS\WIZDEMO\WIZ.APP application.

Tip: Do not check this box until you are finished designing the window!

4. 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 length of the text displayed on each tab, the number of tabs, and the available space on the property sheet.

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. Also see the *Using the List Box Formatter* chapter, *Adding Drag and Drop Capability to the List Box*.

Help Tab

See *Setting Common Control Attributes* above.

Position Tab

See *Setting the AT Attribute* above.

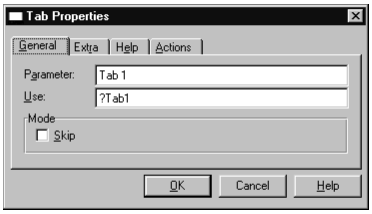
Setting Tab Control Properties

The TAB structure declares a group of controls. This group is one of many 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



Parameter: Tab 1
Use: ?Tab1
☐ Skip

1. In the **Parameter** field, type a string constant.

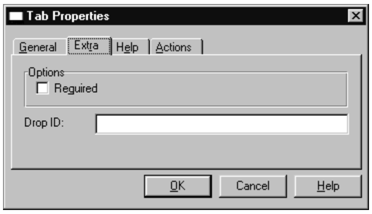
   If the control is to display a variable, type a picture token in this field.

2. In the **Use** field, type a field equate label.

   The field equate label references the tab control in program statements.

3. **Mode** options: see *Setting the Mode Attributes* above.

Extra Tab



☐ Required
Drop ID:

4. Check the **Required** box to enforce input to required input fields.

   When checked, your program automatically checks that all ENTRY controls with the REQ attribute are neither blank nor zero.

   Specify this type of tab 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 clicks on a tab with the REQ attribute and an ENTRY field is blank or zero, the first required control which is blank or zero receives the focus.

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. Also see the *Using the List Box Formatter* chapter, *Adding Drag and Drop Capability to the List Box*.

Help Tab

See *Setting Common Control Attributes* above.

Position Tab

The position of the TAB is determined by the position of the parent SHEET.

CUSTOM CONTROLS

Setting Custom Control Properties

Custom controls are "add-in" controls sold by many third party vendors. These perform a very wide variety of tasks, from sliders and gauge controls to TWAIN image capture add-ins. The Window Formatter allows you to directly place these controls once you "register" the external libraries.

The specific custom 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*. Custom 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.

Custom control libraries usually require a license file (*.LIC) 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 Your .VBX Custom Control Libraries

Before you can place a custom control in a window, you must register the .VBX file which contains it. To do so:

1. From Clarion’s main menu, choose **Setup ➤ VBX Custom Control Registry**.
2. Press the **Add** button in the **VBX Custom Control Registry** dialog box.
3. Select the .VBX file within the **Add Custom Control** dialog, and press **OK**.

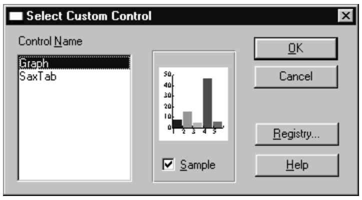
   Some .VBX vendors install their .VBX’s to the \Windows\System directory, while others prefer private directories. When you install a .VBX library to your hard drive, make a note of where you put it, so that you can locate it with the Open File dialog.

   For Clarion for Windows, the .VBX must be in the application’s directory, or somewhere in the system path.
4. Press **OK** to close the **VBX Custom Control Registry** dialog.

Adding a Custom Control to a Window

1. From the **Window Formatter**, select the Custom Control tool, or choose **Custom Control** from the **Control** menu, then CLICK in the window.

The **Select Custom Control** dialog appears. This dialog allows you to 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 box will display a copy of the control in its default settings.



2. Press the **OK** button to return to the **Window Formatter.**
3. RIGHT-CLICK the custom control and select **Properties** from the popup menu.

   The **Custom Control Properties** dialog appears.
4. Optionally type a label for the control in the **Text** field.

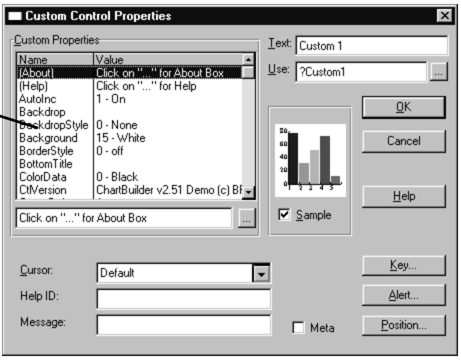
   If the control supports a label, it will appear as part of the control. In practice, most controls will require you to specify a title label as a Visual Basic Control property, explained below.
5. Type a field equate label or variable name in the **Use** field.

   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, explained below.

   .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, which is also explained below.
6. 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.

Some of the VBX properties available for a particular control.


Custom Control Properties

Custom Properties

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
CtlVersion	ChartBuilder v2.51 Demo (c) BF

Click on "..." for About Box

Text: Custom 1

Use: ?Custom1

Cursor: Default

Help ID:

Message:

When you highlight a Visual Basic Control property in the list, either an edit box or a drop down list appears at the lower left corner of the dialog. Type a value or variable in the edit box, or choose from the drop.

The documentation from the .VBX library vendor will describe the Visual Basic Control properties you may set. See below to find out how to change them in executable code, or how to retrieve user-entry from the custom control.

7. Set optional properties.

   These include CURSOR, POSITION, and others.

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 to represent the control.

9. Press the **OK** button.

Setting Visual Basic Control Properties

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 via 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 utilizes a VBX's functionality. Think of a property as a variable which both the application and the VBX can access (this is a very loose comparison).

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's 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.

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 in executable code, and of course, retrieve a value from a property after user entry.

❏ To alter properties in executable code, use the property expression syntax. Access the control's Visual Basic Custom Control properties by referring to the specific property in quotes:

```
?vbxVariable { 'VBProperty' } = value
```

❏ To retrieve the current value of a Visual Basic Custom Control property, use the property expression syntax again:

```
value = ?vbxVariable { 'VBProperty' }
```

Monitoring .VBX Events

Besides properties, the other "channel" by which the .VBX "talks" to your application is via 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.

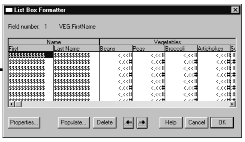
Your application can examine the event, and take appropriate action by interrogating PROP:VBXevent. When working with the Application Generator, you place code similar to the example below at either the embed point labelled "Control Event Handling, before generated code (VBXevent)" or "Control Event Handling, after generated code (VBXevent)." For example, the following can take place in the ACCEPT loop of a dialog box containing a .VBX control.

```
SomeString = ?vbxVariable{PROP:VBXevent}
IF SomeString = ‘UserWantsToDoX’
  SomeProcedure
END
```

USING THE LIST BOX FORMATTER

Contents

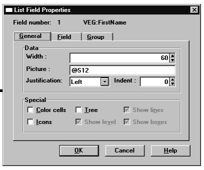
The List Box Formatter is a flexible tool for visually creating and modifying the appearance and function of your list boxes. It displays a sample to show how your editing affects the appearance of the control.



Field number: 1 VEG:FirstName

Name		Vegetables				
First	Last Name	Beans	Peas	Broccoli	Artichokes	Sc
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#

The List Box Formatter presents a sample showing how the list box under construction looks.

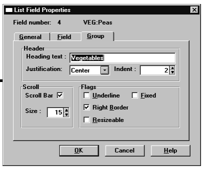


Field number: 1 VEG:FirstName

Data
Width : 60
Picture : @S12
Justification: Left Indent : 0

Special
☐ Color cells ☐ Tree ☑ Show lines
☐ Icons ☑ Show level ☑ Show boxes

The List Field Properties dialog allows you to format a single column at a time.



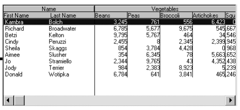
Field number: 4 VEG:Peas

Header
Heading text : Vegetables
Justification: Center Indent : 2

Scroll
Scroll Bar ☑
Size : 15

Flags
☐ Underline ☐ Fixed
☑ Right Border
☐ Resizeable

The List Field Properties Group tab allows you to define two or more fields which share common formatting elements.



Name		Vegetables				
First Name	Last Name	Beans	Peas	Broccoli	Artichokes	Squ
Kambra	Bolch	3,245	761	556	6,423	[illegible]
Richard	Broadwater	6,785	5,677	9,675	545	,667
Betzi	Kelton	9,795	5,767	464	34	,546
Cindy	Peruzzi	2,455	8	2,345	2,399	,945
Sheila	Skaggs	854	3,784	4,428	0	968
Aimee	Slusher	354	6,345	78	5,663	,652
Jo	Stramiello	2,344	9,765	43	4,352	,438
Jody	Terrier	984	2,383	8,923	5	,239
Donald	Wotipka	6,784	641	3,841	465	,246

Sample list boxes and tips for extending the functionality of your list box controls.

The **List Box Formatter** provides a wide degree of flexibility to create and modify your list boxes, drop down list boxes, and combo boxes.

Once you specify a QUEUE to provide the data for the list (done automatically when specifying a browse template), the **List Box Formatter** allows you to customize your list in the following ways:

- You can set the number of columns, with or without resizeable borders.
- You can specify that a record (row) from the QUEUE occupies more than one list box row.
- You can add horizontal scrollbars for each column or group of columns in the list box.
- You can specify the focus on rows or individual “cells,” spreadsheet fashion.
- You can specify headers for the list box columns.
- You can add a special locator control that allows users to quickly find the item they need.
- You can enable selection of multiple rows in the list.

OVERVIEW

A *list box*, by convention is a read-only display of data. It is scrollable and may contain many records and fields. It efficiently displays large amounts of data.

A *drop down list box*, 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 simply a list box with the ability to accept user input.

When creating a list box 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 file or queue that supplies the data, and specifies the general scrolling capability, that is, all the properties of the list box that are *not* column-specific. This dialog is discussed in the previous 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. This dialog is discussed in this chapter.
- The **List Field Properties** dialog defines the appearance and behavior of individual list box columns. For example, define individual column headers, widths, and scrolling. This dialog is discussed in this chapter
- The **List Field Properties** dialog also defines the appearance and behavior of *groups* of columns within the list box. For example, you can spread a header across several columns.

After you've started defining your list box with the **List Properties** dialog, these are the general steps for completing your list box.

❏ Add columns, one by one, using the **List Box Formatter** and the **Select Field** dialog.

1. From the **Window Formatter**, RIGHT-CLICK on the list box control, and choose **List Box Format** from the popup menu to display the **List Box Formatter** dialog.

   The **List Box Formatter** displays a representation of the list box. Each field appears as a column in the list box, the data represented by "$" characters for strings, or "<" and "#" characters for numbers.

2. Press the **Populate** button to add a new field. (When working from the Text Editor, the **Insert** button replaces the **Populate** button).

   When working from within the Application Generator, choose a field from the **Select Field** dialog. The **List Box Formatter** reappears, with the new column added. In the Text Editor, the **Select Field** dialog does not appear; you go directly to the **List Field Properties** dialog, so skip step *3*.

3. Press the **Properties** button.

   The **List Field Properties** dialog appears. Use this dialog to define column headers, widths, borders, scrolling, etc. Formatting for the first column becomes the default format for subsequent columns.

4. Specify the column width in dialog units.

   Provide about four dialog units for each character to be displayed.

5. Specify a picture token for the data.

The picture token determines how the data is displayed. For example, the picture token @P(###) ###-####P displays a phone number as (555) 555-5555.

6. Specify optional formatting.

   You can choose the justification and set indentation. You can specify a column header, borders, underlining, and more.

7. Specify optional functionality.

   For example, add a scroll bar for a single column. Allow column searches. You can specify resizeable borders, that allow the end user to adjust column widths with the mouse.

8. Press the **OK** button to return to the **List Box Formatter** dialog.

   For each modification you make to the list box on screen, the **List Box Formatter** creates the appropriate FORMAT attribute for the LIST statement that defines your list box. The LIST statement, in turn, resides in the WINDOW structure. See the *Language Reference* for a complete explanation.

❑ Optionally group fields.

1. In the **List Field Properties** dialog, select the **Group** tab .

   This specifies that the previous field and the next field you add to the list box share formatting elements.

2. Specify the group heading text.

   The simplest shared element is a common header. The group header appears directly above the field headers for the two fields.

3. Optionally specify additional formatting.

   You can, for example format the fields so that no separator appears between the members of the group, but a separator does appear at the end of the group. To do so, be sure the **Right Border** box is unchecked for the first field(s) in the group, and is checked only for the last.

4. Press the **OK** button to close the **List Field Properties** dialog.

❑ Optionally “stack” multiple fields in a single row of a single column within the list box.

1. On the **Field** tab in the **List Field Properties** dialog, check the **Last on Line** box.

   This option is equivalent to adding a carriage return immediately after the current field. The next field within the group appears directly below the current field, within the same column—under the group header.

2. Press the **OK** button to close the **List Field Properties** dialog.

3. In the **List Box Formatter** dialog, press the **Populate** (or **Insert**) button.

   This allows you to enter the next field. Format the next field following the same steps as for the previous field. Then continue formatting until all fields are added to the list box.

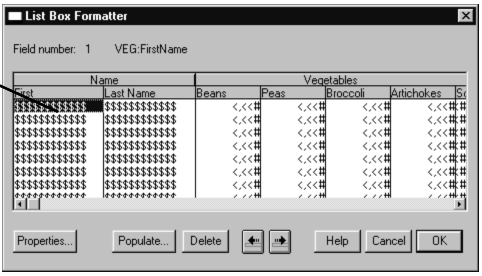
UNDERSTANDING THE LIST BOX FORMATTER

The **List Box Formatter** displays a representation of the list box under construction. The **Field Number:** prompt near the top of the **List Box Formatter** shows you which field is currently selected. Each field appears as a column in the list box, 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 in the **List Field Properties** dialog, which updates the sample list box. 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 list box.

The List Box Formatter, showing a “sample” with two groups.


List Box Formatter

Field number: 1 VEG:FirstName

Name		Vegetables				
First	Last Name	Beans	Peas	Broccoli	Artichokes	Sc
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#

The **List Box Formatter** dialog buttons allow you to add, delete, reorder, resize, and reformat fields in the list box.

❏ To add a field to the list box, press the **Populate** (or **Insert)** button.

  When the **List Box Formatter** is called from the Application Generator, the **Populate** button displays the **Select Field** dialog. From there, you can select any data dictionary field or memory variable for use as a list box column. The generated code puts the selected data dictionary fields into a queue for use in the list box.

When the **List Box Formatter** is called from the Text Editor instead of the Application Generator, the **Insert** button replaces the **Populate** button. You are responsible for building the queue that fills the list box.

❑ To remove a field from the list box, press the **Delete** button.

❑ To display the help file, press the **Help** button.

❑ To cancel the formatting changes, press the **Cancel** button.

❑ To accept the formatting changes, press the **OK** button.

❑ To move the selected field to the left, CLICK the ← button, or press SHIFT+LEFT 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 SHIFT+RIGHT 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.

❑ To format a field, press the **Properties** button.

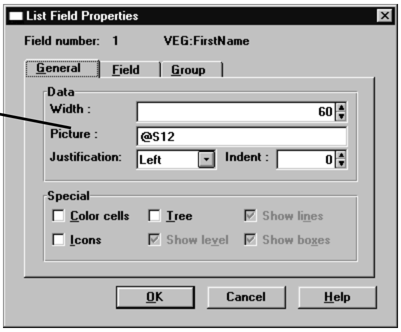
The **List Field Properties** dialog allows you to specify the width of the column, a picture token, heading text, plus other options such as a horizontal scroll bar for the selected field. Additionally, it allows you to “group” fields, which places an extra header across the top of the grouped columns, to visually indicate the fields are linked. The dialog is described in detail, below.

The List Field Properties Dialog

Press the **Insert** or **Populate** button to add a field to the list box, then format it in the **List Field Properties** dialog. For each choice you make in the dialog, the **List Box Formatter** creates the appropriate FORMAT attribute for the LIST statement that defines your list box.

General Tab

The dialog allows you to set the following **Data** formatting options.


List Field Properties

Field number: 1 VEG:FirstName

General | Field | Group

Data

Width : 60

Picture : @S12

Justification: Left Indent : 0

Special

[ ] Color cells [ ] Tree [✓] Show lines

[ ] Icons [✓] Show level [✓] Show boxes

Specifying the List Field Properties.

Width	Specify the width in dialog units for the column. 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.

After you've placed a field, the **List Box Formatter** dialog allows you to drag the column separators to resize a 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."

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.

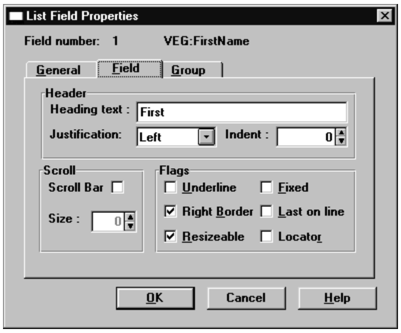
Justification	Choose from the drop down list to specify left, right, center or decimal. Decimal justification aligns decimal numbers by their decimal points.

The justification appears in the FORMAT string following the data width, as in "40R."

Indent	Optionally specify an indent, in dialog units, for the list box 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 list box readability. The indent appears within the FORMAT string surrounded by parentheses and preceded by a letter indicating the justification, as in “L(8).”
Color Cells	Checking this box displays this column with a solid or filled background at runtime. Color cells appears as an asterisk “*” in the FORMAT string.
Icons	Checking this box creates an area to the left of the data in the column that displays a graphic icon that you supply. Adds an “I” to the FORMAT string.
Tree	Checking this box displays this column in a hierarchical tree diagram. See *Relation Tree* in the *Using Control, Code, and Extension Templates* chapter. See also Relation Tree in the on-line help. Adds a “T” to the FORMAT string.
Show Level	Checking this box causes each descending level of the Tree hierarchy to be indented. Unchecking this box appends “(I)” to the “T” in the FORMAT string, resulting in “T(I)” to suppress indention.
Show Lines	Checking this box adds connecting lines between related items in the tree diagram. Unchecking 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.

Unchecking this box appends “(B)” to the “T” in the FORMAT string, resulting in “T(B)” to suppress boxes.

Field Tab


List Field Properties

Field number: 1 VEG:FirstName

Header
Heading text : First
Justification: Left
Indent : 0

Scroll
Scroll Bar ☐
Size : 0

Flags
☐ Underline ☐ Fixed
☑ Right Border ☐ Last on line
☑ Resizeable ☐ Locator

Heading Text

Optionally specify header text for the column. The header appears as a gray row above the list box data items. To specify no header, leave this field blank. If any field included in the list box has a header, a header appears across the entire list box; those fields with no header text will have a blank header.

The heading appears within the FORMAT string enclosed in tilde (~) characters, as in “~My Header~.”

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 list box readability.

This appears within the FORMAT string following the header, as in “~My Header~L(8).”

Scroll Bar	Check this box to specify a horizontal scroll bar for this column only. If the overall list box already has a scroll bar, the column scroll bar appears above the list box scroll bar.
Size	Specifies, in dialog units, how far the column scrolls. This value determines the width of the scrolling area that is *not* displayed in the list box.
	For example, if your *data* item is fifty (50) characters, and your list box *column* width is about forty (40) characters (one hundred sixty (160) *dialog units*), you should specify a **Size** of fifty (50). Fifty (50) additional dialog units are enough to display the ten characters that extend beyond the width of the list box column.
	The scroll bar and size appear in the FORMAT string together, as in “S(4).”
Underline	Check the **Underline** box to add the underline style to the list box text. In effect, this creates a bottom border for each row in the column, giving your list box a spreadsheet or cell-like appearance.
	The FORMAT string includes the underscore character, immediately preceding the header text, as in “_~My Header~.”
Right Border	Check the **Right Border** box to specify column separators between fields in the list box at run time.
	The FORMAT string includes the pipe symbol (|), immediately preceding the header text, as in “|~MyHeader~.”
Resizeable	Check the **Resizeable** 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~.”

Tip: At runtime, the PROP:Format property always contains the current format of the list box, including any user changes. To save the user's column sizes, use GETINI and PUTINI to save and restore the PROP:Format values:

```
PUTINI('List Settings','UserList',?List{PROP:Format},"MYAPP.INI')
```

Fixed	Check the **Fixed** box to specify that the column always remains visible in the list box. 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 within a single selection, below the group header. The FORMAT string includes the "/" character, immediately preceding the header text as in "/ ~MyHeader~."
Locator	Check the **Locator** box to specify that this column works with a locator entry control. When the user types a character in the locator entry control, the list box scrolls to the first matching entry in the column. The FORMAT string includes the "?" character, immediately preceding the header text as in "?~MyHeader~."

Creating Column Groups

List box groups contain two or more fields which share common formatting elements, such as a header, or two fields which appear "stacked" within a single selection. Specify a group header to visually link adjacent fields. Check the **Last on Line** box on the first field to set up field "stacking."

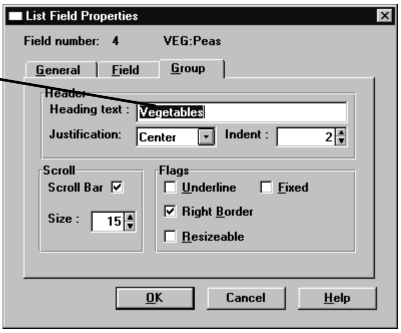
Create a group by selecting the first field in the group, then selecting the **Group** tab in the **List Field Properties** dialog. The next field you choose from the **Select Field** dialog or the next field in the QUEUE becomes the next member of the group. Alternatively, use the ← and → buttons on the **List Box Formatter** to move fields into and out of an existing group.

From the **List Field Properties** dialog, you can specify a group header that appears above the column headers. Because the right border attribute is set separately from the fields, you can create a header which appears to stretch across columns.

With creative formatting, you can use group headers to visually link related data residing in different columns. For example, you can place "Name" in the group header for column one, then "First" and "Last" in the header fields for the first two fields. You can also use a group header to break up the header text into two lines when the column label is longer than the column width.

Group Tab

Specifying a Group Header called "Vegetables."


List Field Properties

Field number: 4 VEG:Peas

Header
Heading text : Vegetables
Justification: Center
Indent : 2

Scroll
Scroll Bar ☑
Size : 15

Flags
☐ Underline
☐ Fixed
☑ Right Border
☐ Resizeable

The following items appear in the **Group** tab **List Field Properties** dialog:

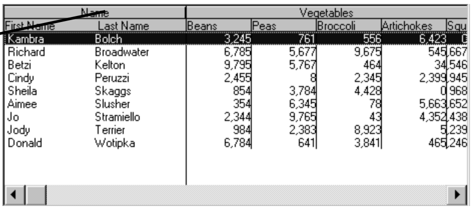
Heading Text	Optionally specify header text for the group. The header appears as a gray row above the list box data items. To specify no header, leave this field blank. If any field included in the list box has a header, a header appears over each field in the list box; those fields with no header text will have a blank header.
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. **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 list box readability.
Scroll Bar	Check the **Scroll Bar** box to specify a single horizontal scroll bar for the entire group. If the overall list box already has a scroll bar, the group scroll bar appears above the list box scroll bar. If individual columns in the group have scroll bars, they are superseded by the group scroll bar. This is an effective way to present related list items to the user in an organized manner.
Size	Specify the range of the scroll bar. This value determines the width of the scrolling area that is *not* displayed in the list box. For example, if your *data* item is fifty (50) characters, and your list box *column* width is about forty (40) characters (one hundred sixty (160) *dialog units*), specify a **Size** of fifty (50). Fifty (50) additional dialog units are enough to display the ten characters that extend beyond the width of the list box column. The scroll bar and size appear in the FORMAT string together, as in "S(4)."
Underline	Check the **Underline** box to add the underline style to the list box text. In effect, this creates a bottom border for each row in the column, giving your list box a spreadsheet or cell-like appearance. The FORMAT string includes the underscore character, immediately preceding the header text, as in "_~My Header~."
Right Border	Check the **Right Border** box to specify column separators between fields in the list box at runtime.

	The FORMAT string includes the pipe symbol (|), immediately preceding the header text, as in "|~MyHeader~."
Resizeable	Check the **Resizeable** 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~."
Fixed	Check the **Fixed** box to specify that the group always remains visible in the list box. The FORMAT string includes the "F" character, immediately preceding the header text as in "F~MyHeader~."

Creating a Header Above Two Adjacent Columns

A sample list box control with Group Headers.



Name		Vegetables				
First Name	Last Name	Beans	Peas	Broccoli	Artichokes	Squ
Kambra	Bolch	3,245	761	556	6,423	0
Richard	Broadwater	6,785	5,677	9,675	545	667
Betzi	Kelton	9,795	5,767	464	34	546
Cindy	Peruzzi	2,455	8	2,345	2,399	945
Sheila	Skaggs	854	3,784	4,428	0	968
Aimee	Slusher	354	6,345	78	5,663	652
Jo	Stramiello	2,344	9,765	43	4,352	438
Jody	Terrier	984	2,383	8,923	5	239
Donald	Wotipka	6,784	641	3,841	465	246

```
FORMAT('[60L(2)M ~First Name~L(0)@S12@60L|M ~Last
Name~L(2)@S12@]|M ~Name~[43R|M ~Beans~L@n5@43R|M ~Peas~L@n5@43R|M
~Broccoli~L@n5@43R|M ~Artichokes~L@n5@40R|M ~Squash~L@n5@]|M
~Vegetables~')
```

1. In the **List Box Formatter**, press the **Populate** (or **Insert)** button to add a field to the list box.

   The **Select Field** dialog appears.

2. Use the **Select Field** dialog to select the field to add to your list box.
3. Press the **Properties** button to open the **List Field Properties** dialog.

   Use the **List Field Properties** dialog to format the column normally.

4. In the **List Field Properties** dialog, select the **Group** tab.
5. Press the **OK** button when asked if you want to create a new group.
6. In the **Heading Text** field, type the group heading text.

   This text is shared by all the fields in the group. By default, it is centered above the group, however, you may specify other justifications.

7. Press the **OK** button to close the **List Field Properties** dialog.

   The **List Box Formatter** reappears.

8. Press the **Populate** (or **Insert)** button in the **List Box Formatter** dialog.

   The **Select Field** dialog appears.

9. Use the **Select Field** dialog to select the field to add to your list box.

   The field is added to the group established in the previous dialogs.

10. Press the **Properties** button to open the **List Field Properties** dialog.

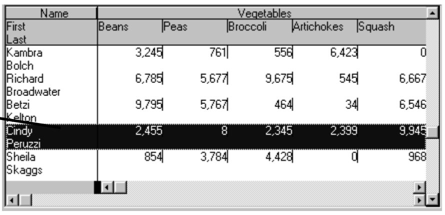
    Use the **List Field Properties** dialog to format the column normally.

11. Press the **OK** button to close the **List Field Properties** dialog.

    The **List Box Formatter** reappears. Additional fields added in this fashion will be added to the group. To “end” the group, press the → button. This moves the last field added to the group, out of the group.

Stacking Two Fields in the Same Column

The same sample list box showing a “stacked” row.



Name	Vegetables				
First Last	Beans	Peas	Broccoli	Artichokes	Squash
Kambra Bolch	3,245	761	556	6,423	0
Richard Broadwater	6,785	5,677	9,675	545	6,667
Betzi Kelton	9,795	5,767	464	34	6,546
Cindy Peruzzi	2,455	8	2,345	2,399	9,945
Sheila Skaggs	854	3,784	4,428	0	968

```
FORMAT('[60L|M~First~@S12@/60L|M~Last~@S12@]|M ~Name~C(2)[43R|M
~Beans~L@n5@43R|M ~Peas~L@n5@43R|M ~Broccoli~L@n5@43R|M
~Artichokes~L@n5@40R|M ~Squash~L@n5@]| ~Vegetables~C(2)S(15)')
```

1. In the **List Box Formatter**, press the **Populate** (or **Insert)** button to add a field to the list box.

   The **Select Field** dialog appears.

2. Use the **Select Field** dialog to select the field to add to your list box.

3. Press the **Properties** button to open the **List Field Properties** dialog.

4. In the **List Field Properties** dialog, check the **Last on Line** box.

5. In the **List Field Properties** dialog, select the **Group** tab.

6. Press the **OK** button when asked if you want to create a new group.

7. Press the **OK** button to close the **List Field Properties** dialog.

   The **List Box Formatter** reappears.

8. In the **List Box Formatter**, press the **Populate** or **Insert** button to add a second field to the list box.

   The field is added to the group established in the previous dialog.

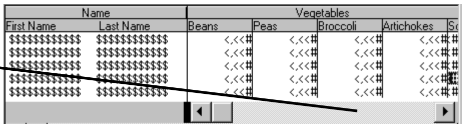
9. Use the **List Field Properties** dialog to format it normally.

10. Press the **OK** button to close the **List Field Properties** dialog.

    The **List Box Formatter** reappears. Additional fields added in this fashion will be added to the group. To "end" the group, press the → button. This moves the last field added to the group, out of the group.

Grouping Fields With a Group Scroll Bar

Adding a scroll bar for the "Vegetables" group. The data is random characters and numbers.



Name		Vegetables				
First Name	Last Name	Beans	Peas	Broccoli	Artichokes	Sc
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#
$$$$$$$$$$$$	$$$$$$$$$$$$	<,<<#	<,<<#	<,<<#	<,<<#	#

```
FORMAT('[60L(2)M~ First Name~L(0)@S12@60L|M ~Last Name~L(2)@S12@]|M
~Name~[43R|M ~Beans~L@n5@43R|M ~Peas~L@n5@43R|M
~Broccoli~L@n5@43R|M ~Artichokes~L@n5@40R|M ~Squash~L@n5@]|M
~Vegetables~S(35)')
```

1. In the **List Box Formatter**, press the **Populate** (or **Insert)** button to add a field to the list box.

   The **Select Field** dialog appears.

2. Use the **Select Field** dialog to select the field to add to your list box.

3. Press the **Properties** button to open the **List Field Properties** dialog.

4. In the **List Field Properties** dialog, select the **Group** tab.

5. Press the **OK** button when asked if you want to create a new group.

6. On the **Group** tab, check the **Scroll Bar** box.

7. Press the **OK** button to close the **List Field Properties** dialog.

   The **List Box Formatter** reappears.

8. In the **List Box Formatter**, press the **Populate** or **Insert** button to add a second field to the list box.

   This field is added to the group established in the previous dialog.

9. Use the **List Field Properties** dialog to format it normally.

10. Press the **OK** button to close the **List Field Properties** dialog.

The **List Box Formatter** reappears. Additional fields added in this fashion will be added to the group. To "end" the group, press the → button. This moves the last field added to the group, out of the group.

Tip: Most likely, you'll leave the Right Border box unchecked for the first fields in the group, and checked for the last field in the group. This separates the group as a whole from other fields in the list box.

LIST BOX EVENTS AND OTHER FUNCTIONALITY

If you stick to the templates, you may never need to manage the list box events. The Browse template, for example, allows you to name the Update procedure. This relieves you of the coding chores required to monitor for events like DOUBLE-CLICKS. However, if you need to extend the functionality of the list box to allow drag and drop, or multiple selections, you may need to add statements to the appropriate ACCEPT loop, to check for specific list box events.

Tip: Some of the following sections have little to do with the List Box Formatter. They reside in this chapter, however, because we thought you would look here for them first.

Trapping a Double Click on a List Box

Trapping a DOUBLE-CLICK on a list box is built into the Clarion Browse templates. To trap a DOUBLE-CLICK on a list control in hand-code:

1. Establish an ALRT(double-click) on the list control.
2. Trap for EVENT:AlertKey on the list control.
3. Trap for the MouseLeft2 keycode, as in the following example:

```
ACCEPT
  CASE FIELD()
  OF ?List
    CASE EVENT()
    OF EVENT:AlertKey
      IF KEYCODE() = MouseLeft2
        CurrentSel = CHOICE(?List1) ! Get current selection in list box
        GET(TheQueue, CurrentSel)   ! Get corresponding data from queue
. . . .
```

The above code finds out what item the user DOUBLE-CLICKED on using the CHOICE() function, then uses the GET() function to retrieve the item from the QUEUE.

You can add the two lines of code within the above IF structure to the Browse Double Click Handler embed point to handle DOUBLE-CLICKS for lists populated with the BrowseBox control template in the Application Generator.

Adding Drag and Drop Capability to the List Box

Drag and Drop capability for lists means the user can select an item in a list box, hold down the left mouse button, "drag" the item to another control, release the mouse button to "drop" the item on the control, which can look at the data that was "dropped" on it, and then do something with it.

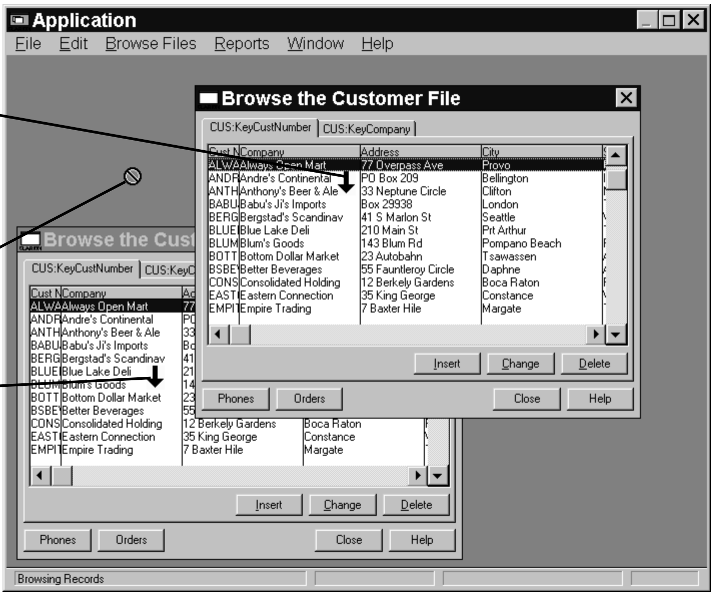
Adding Drag and Drop to a Clarion for Windows list box is a simple operation. This section provides an example of dragging an item from one list box to another, within the same application. You can also "Drag and Drop" to or from another application—for example, File Manager—see the *Language Reference* for more details.

To implement Drag and Drop, you must add the DRAGID and DROPID attributes to the controls. You can add either or both to a control. The simplest, quickest way to do this is with Property Syntax statements. Assume for this example that the field equate labels for the two list boxes are ?FromList and ?ToList. Assume you want the end user to be able to drag *from* ?FromList *to* ?ToList.

Two List Boxes, both of which can be *either* Drag and Drop hosts *or* targets. The Cursor changes to a down-arrow when selecting an item in the top list.

It changes to a "No Smoking" icon over areas where it can't be dropped.

Finally, the cursor changes to a down arrow, indicating an area where it can be dropped.



Cust N	Company	Address	City
ALWA	Always Open Mart	77 Overpass Ave	Provo
ANDR	Andre's Continental	PO Box 209	Bellington
ANTH	Anthony's Beer & Ale	33 Neptune Circle	Clifton
BABU	Babu's Ji's Imports	Box 29938	London
BERG	Bergstad's Scandinav	41 S Marlon St	Seattle
BLUE	Blue Lake Deli	210 Main St	Prt Arthur
BLUM	Blum's Goods	143 Blum Rd	Pompano Beach
BOTT	Bottom Dollar Market	23 Autobahn	Tsawassen
BSBE	Better Beverages	55 Fauntleroy Circle	Daphne
CONS	Consolidated Holding	12 Berkely Gardens	Boca Raton
EAST	Eastern Connection	35 King George	Constance
EMPI	Empire Trading	7 Baxter Hile	Margate

❏ Set up ?FromList as a drag host:

1. RIGHT-CLICK on the list control and choose **Properties** from the popup menu.
2. Select the **Extra** tab.
3. In the **Drag ID** field, type “FromList.”
4. Press the **OK** button.

   This sets the Drag ID signature which identifies the source of any “drag” operation from this control.

❏ Set up ?ToList as a drop target:

1. RIGHT-CLICK on the list control and choose **Properties** from the popup menu.
2. Select the **Extra** tab.
3. In the **Drop ID** field, type “FromList.”
4. Press the **OK** button.

   This sets the Drop ID signature which specifies that the list will accept any “drop” operation with a Drag ID signature of “FromList.”

❏ Add drag functionality to the drag host, that is, detect a drag event and provide something to drag and drop:

1. RIGHT-CLICK on the FromList control and choose **Embeds** from the popup menu.
2. Locate the “Control Event Handling, before generated code;” event:Drag embed point and embed the following code:

```
IF DRAGID()                ! Doesn’t matter who dropped it for now
  SETDROPID(‘string to drag and drop’)    ! Passing a simple string
END
```

   This code detects a drag event—at the time the user *releases* the mouse button over a valid drop target—and places a string to drag with the SETDROPID function.

   You can just as easily use the CHOICE() and GET() functions to retrieve an item from the local QUEUE for the first list box, then place the item in a global QUEUE. Then, upon detecting a drop event in the second list box, you could ADD from the global QUEUE to the local QUEUE for the second list box.

❏ Add drop functionality to the drop host, that is, detect a drop event and retrieve whatever was dropped:

1. RIGHT-CLICK on the ToList control and choose **Embeds** from the popup menu.
2. Locate the “Control Event Handling, before generated code;” event:Drop embed point and embed the following code:

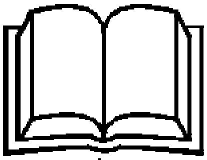
```
MyField = DROPID()    ! Retrieve the passed string
CallMyProcedure       ! Handle the rest in procedure
```

   This code detects a drop event—at the time the user *releases* the mouse button over the drop target—and retrieves the “dropped” string with the DROPID function.

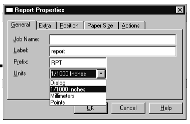
   You can just as easily use the CHOICE() and GET() functions to retrieve an item from the local QUEUE for the first list box, then place the item in a global QUEUE. Then, upon detecting a drop event in the second list box, you could ADD from the global QUEUE to the local QUEUE for the second list box.

USING THE REPORT FORMATTER

Contents

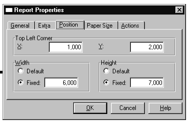


Visually lay out your applications's reports. The Report Formatter automatically generates the REPORT data structure, which handles page formatting and monitors page overflow.



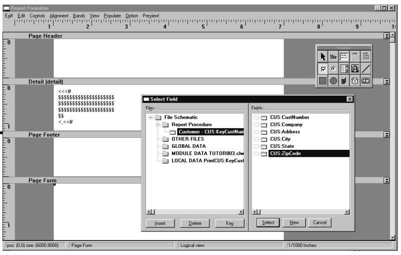
Job Name:
Label: report
Prefix: RPT
Units: 1/1000 Inches
Dialog
1/1000 Inches
Millimeters
Points

Specify the report name, default measurement unit, and page orientation in the Report Properties dialog.



Top Left Corner
X: 1,000
Y: 2,000
Width: Default [✓ Fixed:] 6,000
Height: Default [✓ Fixed:] 7,000

Specify default margins in the Position dialog.



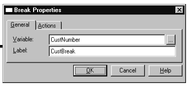
```
<<<#
$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$
$$
<,<<#
```

File Schematic
Report Procedure
Customer - CUS:KeyCustNum
OTHER FILES
GLOBAL DATA
MODULE DATA TUTOR003.clw
LOCAL DATA PrintCUS:KeyCust

CUS:CustNumber
CUS:Company
CUS:Address
CUS:City
CUS:State
CUS:ZipCode

pos: (0,0) size: (6000,9000) | Page Form | Logical view | 1/1000 Inches

Easily populate the report with text, variables, lines, and graphics. Design report headers, detail, and footers.



Variable: CustNumber
Label: CustBreak

Divide the report into sections with a Group BREAK structure.

Using the **Report Formatter**, you *visually* lay out your application's reports. The **Report Formatter** automatically generates and places the structures for the report elements in the REPORT data structure.

This chapter will:

- Tell you how to call the **Report Formatter** to create or edit a report.
- Provide an overview of the parts or sections of a report.
- Discuss page layout considerations, such as orientation and automatic widow and orphan handling.
- Tell you how to place both data and graphic controls on the page.
- Tell you how to set control properties so your report can automatically calculate totals and averages by group, page, or grand total.

OVERVIEW: REPORT PROCESSING

Before learning how to create a report using the **Report Formatter**, it's important to understand how Clarion executes a report—in other words, the division of labor between the print engine and your source code, and the order in which Clarion processes all the sections of your report. Each *section* of the report is a data structure, which in turn is contained in the REPORT structure.

Smart Processing

The REPORT data structure contains all the information necessary for formatting and printing each page. Clarion's internal "print engine" automatically handles page overflow management, including widow and orphan control. This frees you from worrying about the "mechanics" and makes the Clarion executable code to print a report simple and clean.

The following example shows how a total of six lines of executable code can access the file and print a fully-formatted listing of all Customers. Since the **Report Formatter** writes the entire REPORT data structure, this is all the code the programmer has to write:

```
OPEN(CustReport)                  !Open report for processing
SET(Cus:NameKey)                  !Top of file, alpha order
LOOP                              !Process the entire file
  NEXT(CustomerFile)              !one record at a time
    IF ERRORCODE() THEN BREAK.    !Check for errors
  PRINT(Rpt:Detail)         !Tells the REPORT struc to do the work.
END
```

Of course, if you're using the Application Generator, you don't even have to write that much, since the code is written for you! In the example above, the PRINT statement prints a DETAIL structure for each record in the file retrieved with the NEXT statement inside the LOOP.

The REPORT data structure contains the items that belong on the page, plus the attributes that determine how they appear there. Since you visually design these in the **Report Formatter**, the code example above really is all you have to do to print the report.

The PRINT statement automatically initiates the page overflow handling. This means that when the LOOP goes through enough records to fill up the page, it automatically generates any other structures on the page—the FOOTER, for example, before it sends the entire page to the print spooler.

REPORT Structures

Each report contains sections, which are also Clarion data structures. These sections are the FORM, HEADER, BREAK, DETAIL, and FOOTER.

- The FORM structure prints as a "background layer." Typically, you may display "overlays" such as graphics and field labels in the FORM layer, then print the actual data in the DETAIL. The FORM content remains constant from page to page.
- The HEADER structure traditionally prints at the top of each page of the report. Typically, you place the report title, graphics and other "introductory" elements in the HEADER.
- The DETAIL structure is the "body" of the report. It contains the basic data, either in table or record format.
- The FOOTER structure traditionally prints at the bottom of the report. Typically, you may place a page number, or totals in the FOOTER.
- A Group BREAK structure can contain its own HEADER, DETAIL, and FOOTER structures, plus nested BREAK structures with their own HEADER, DETAIL, FOOTER structures. The print engine

composes these sections whenever the group BREAK variable changes. BREAKs are typically used to display section summaries, for example, department totals.

Clarion for Windows provides complete flexibility in placing the FORM, page HEADER, and page FOOTER structures at any position on the page. DETAIL structures print in the area defined for them by the REPORT's AT attribute—the detail print area—and are printed relative to the end of the last section printed in the detail print area. The group HEADER and FOOTER also print inside the detail print area, at an offset relative to the last detail to print.

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
```

Processing Order

As mentioned above, a REPORT data structure is comprised of five sub-structures: FORM, HEADER, DETAIL, FOOTER, and BREAK. Each BREAK structure can contain its own HEADER, DETAIL, and FOOTER.

Normally, you want to design reports with only one DETAIL. When you generate reports using the Application Generator and **Report Formatter**, they will have only one DETAIL. This DETAIL should be inside any group BREAK structure you create. Alternatively, When you generate reports using the Text Editor and **Report Formatter**, there is a DETAIL for each BREAK. You can delete any DETAILs not used.

Once you know the order in which the report sections generate at print time, you can understand how to use them better. For the following example, assume a report containing one BREAK structure, with a DETAIL section inside it. Upon execution of the PRINT command:

1. The print engine composes the FORM, but does not send it to the print spooler yet. The FORM section generates only once; the print engine does not recompose it for additional pages.
2. The print engine composes the page HEADER.
3. The print engine composes the group HEADER.
4. The application composes the DETAIL section (within the BREAK) as many times as will fill the first page, continuously checking for group BREAKs.

   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 application generates the DETAIL section for the next group of records, continuously checking for further 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 first page to the print spooler.
11. For page two, since the FORM section was composed already, it does not get regenerated, though it will print on the page. The print engine proceeds directly to the page HEADER.
12. The application repeats the procedures above for this and all additional pages.

Flexibility

The page-oriented nature of the **Report Formatter** is the key to its flexibility. The print engine composes each page in its entirety before sending it to the printer. This means you may arrange the parts of the report into any page layout you wish.

You can place the FORM, page HEADER, and page FOOTER structures anywhere on the page, within certain limitations. Their page placement does not affect the order that the application generates these sections of the report.

That means you can physically place a page FOOTER above a page HEADER. Since the FOOTER generates only after the report processes all the records on the page, this allows you, for example, to place a page total *above* the records on the page.

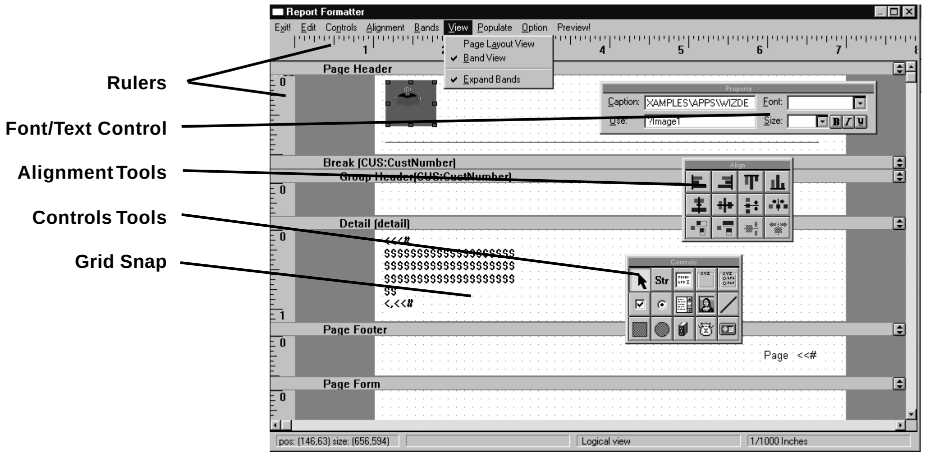
You set the position and size of the DETAIL structure as an offset relative to the last DETAIL printed. The print engine prints each DETAIL from page top to page bottom. If the DETAIL is narrow enough so that more than one fits across the width of the page, they print left to right, then top to bottom. Group BREAK structures—group HEADER, group DETAIL and group FOOTER—all print as offsets within the DETAIL area, one after the other.

You can do some fancy footwork in cooperation with the print engine. For example, because the DETAIL structure must be printed with the PRINT statement, you can use embedded source to place conditional statements within your executable code, to print one DETAIL for one condition, and another for a different condition.

As long as you remember the order in which the print engine generates each section, which determines the current record and the values of the totals, tallies and other operations on the fields in each structure, you can build in a great deal of flexibility within the REPORT data structure, and let the print engine worry about fitting it all onto the page at runtime.

OVERVIEW: THE REPORT FORMATTER

Design your report visually—by placing text, variables, and graphics—on a representation of a page. Rulers, alignment tools, and grid snap allow you to place these items precisely.



Page Oriented Printing

By Page Oriented, we mean that Clarion's print engine composes everything on the page, and only *after* everything is processed does it send the page to the printer. This may be a radical change for programmers switching from DOS report writers, which send output to the printer line by line.

Page oriented printing provides these benefits:

- You can place totals anywhere on the page. If the total is in a page FOOTER, you can move the FOOTER anywhere, including over the entire page.
- You can easily print graphic elements.
- You can place a "preprinted" graphic form in the FORM structure, then format the report so that data prints inside the areas specified for each data element.
- Clarion for Windows handles page breaks and group breaks automatically, even allowing you to specify how to handle "widow" and "orphan" details.
- You can easily build in a "Print Preview" command in your applications. In fact, to implement it in the Application Generator, you just check a box in the **Report Properties** dialog.

Opening the Report Formatter

This section describes how to open the **Report Formatter**, from both the Application Generator and from the Text Editor. It also discusses how to set up your database beforehand, to ensure that group data prints in the right order.

In general, you call the **Report Formatter** either to create a new report, or to edit an existing report. The **Report Formatter** allows you to visually place your report elements on the page. It then automatically generates the REPORT data structure. 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. Choose **Procedure ➤ New** from the menu.

   The **Select Procedure Type** dialog appears

2. Choose the Report template, uncheck the **Use Procedure Wizard** box, and press the **Select** button.

   The **Procedure Properties** dialog appears

3. Press the **Report** button.

   The **Report Formatter** appears. You're now ready to define the report by section, specify headers, detail, and footers, and add controls.

❏ *To create a new report 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 statements which create the report.
3. Choose **Edit ➤ Format Structure** from the menu. You may also press the keyboard accelerator, CTRL+F.
4. When the **New Structure** dialog appears, choose **Report**.

   The **Report Formatter** appears. You're now ready to define the report by section, specify headers, detail, and footers, and add controls.

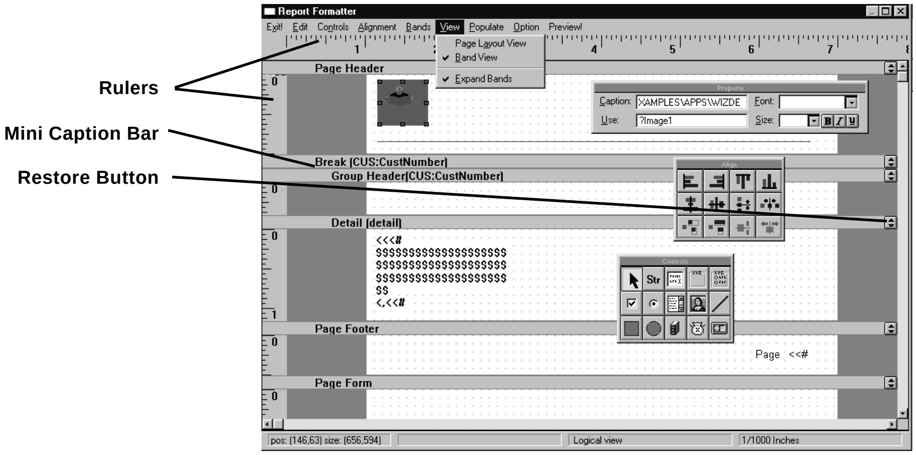
Tip: To edit an existing report from the Text Editor, open the source code file and place the cursor on any line within the REPORT structure, then choose Edit ➤ Format Structure from the menu, or press CTRL+F.

Views

Band View

When you first open the **Report Formatter**, your report appears in Band View. That is, each section, HEADER, DETAIL, FOOTER, and FORM, appears in a separate band inside the window.

Band View features What You See is What You Get (WYSIWYG) editing *one section at a time*. You may specify that sections print side by side or even overlap on the page, but in band view, each section appears in separate band, and each band appears in its proper hierarchical order (grouped within any BREAK structures).



Caption: XAMPLES\APPS\WIZDE Font:
Use: ?Image1 Size:

pos: (146,63) size: (656,594) | Logical view | 1/1000 Inches

- ❏ *To quickly determine each section's position, look at the rulers*. Each band has its own vertical (Y) ruler, set according to your default measurement units (see below). The X ruler shows the position relative to the left edge of the page. The Y rulers show the positioning relative to the top of the band.

- ❏ *To identify a band's contents*, look at its mini caption bar. This is particularly useful when you add group breaks—the name of the variable the group breaks on appears on the band's caption.

- ❏ *To control the band's display*, use its restore button. This appears at the right side of the caption bar, and toggles the band open and closed. The band's caption bar is always available.

- ❏ *To close all the bands at once*, choose **View ➤ Expand Bands** from the menu.

Place controls in the bands you wish. The controls may contain constant strings, variable strings, graphic elements, or, in the case of a group box, it may even contain other controls.

❑ *To select the type of control to place*, choose one from the **Controls** toolbox, or the **Controls** menu.

❑ *To place the control*, CLICK inside the band representing the section in which you wish it to print.

❑ *To modify the control*, DOUBLE-CLICK on it, or RIGHT-CLICK on it and choose **Properties** from the popup menu.

Other Views

You can edit your report only in Band View. You may view your report as it appears on the printed page by choosing **View ➤ Page Layout View**, or **Preview!**. Page Layout additionally allows you to move and resize the HEADER, FOOTER, DETAIL, and FORM sections by CLICKING and DRAGGING.

❑ *To view your report in page layout mode*, choose **View ➤ Page Layout** from the action bar. Page Layout view allows you to select, relocate, and resize report sections by dragging their handles.

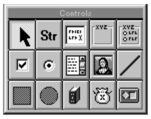
❑ To preview a sample of how your report will look on the printed page, choose **Preview!** from the action bar.

Report Formatter Tools

Controls Toolbox

The **Report Formatter** contains a floating **Controls** toolbox, similar to the one found in the **Window Formatter**. Simply choose a control from the toolbox (CLICK on it), then CLICK in a report band to place the control in the report.

Display or hide the **Controls** toolbox by choosing **Options ➤ Show Toolbox.** All the controls in the toolbox are also available from the **Controls** menu. See *Placing Controls in a Report* below. Also see *Setting Report Control Properties* below.

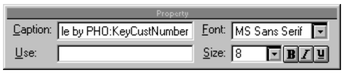


Tip: Position the cursor over any tool and wait for half a second. A tool tip appears telling you the type of control that will be created by this tool.

Property Toolbox

The **Report Formatter's Property** toolbox allows you to quickly specify the appearance and content of the *text* on each control within the report. Control the font, size, style, and content of all your text, using standard word processor buttons and drop down lists.

Display or hide the **Property** toolbox by choosing **Options ➤ 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.

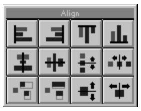


Align Toolbox

The **Report Formatter's Align** toolbox allows you to quickly, professionally, and precisely align the controls in your report. Select the controls to align (CTRL+CLICK allows you to select multiple controls, or you can "lasso" multiple controls with CTRL+DRAG), then click on the appropriate alignment tool. All the alignment actions are also available from the **Align** menu.

Display or hide the **Align** toolbox by choosing **Options ➤ 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.

Tip: For most alignment functions, the first controls 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.



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.

Report Controls

Variable String Controls

Variable string controls are the basic unit for printing data on the page. The **Report Formatter** places a STRING control in the REPORT data structure with a variable as the control's USE attribute. This establishes a string control that display a variable value rather than a constant value.

By using the USE variable, the application accesses the variable data you want to print. This may be a memory variable, or a data dictionary field. The **Report Formatter** formats the data according to the picture token you specify.

The process of creating the report DETAIL relies on placing variable string controls that reference the data you want. You may place the controls in any layout you wish.

You may place other types of controls on the report page—but you will probably find that variable strings are the basic building blocks.

Total Fields

A total field is a variable STRING control with the SUM attribute added. The AVE, CNT, MAX and MIN attributes similarly create averages, counts, maximum, and minimum fields.

In general, you place total fields in a page or group FOOTER, so that it can total the records since the beginning of the report, since the beginning of the page, or since the beginning of the BREAK group. You can also place a total field in a DETAIL structure to provide a running subtotal. A tally (CNT) field in the DETAIL can number the records as they appear on the report.

To specify a total field, DOUBLE-CLICK on a variable string control, then choose a Total type from the **Total type** drop down list. Use the **Reset** drop down list to reset the total to zero after each page to provide a page total.

Graphics

A simple graphic element such as a line can visually set off the different parts of a report, making its printed content much easier for the reader to understand.

For example, drawing a line in a group FOOTER separates each new group from the last. Drawing a shaded box around a group of controls can link them together visually, just as a box filled with color can link screen elements together. Placing your company logo in the HEADER of the report is a simple way to make a report more professional looking. Use the image control to place graphics in your report.

DESIGNING YOUR REPORT

Because the **Report Formatter** gives you total freedom to design the report sections anywhere you wish, *plan* the *type of report* you wish to design. Then, before you place data elements, you should set up the "basics." The basics are the page orientation, the page margins, and the measurement unit.

Report Types

The **Report Formatter** allows you to freely design reports any way you wish. Here are a few design types for you to consider, and tips on executing them:

- A tabular report prints each section consecutively, below the previous. When placing data elements, lay them end-to-end horizontally across the page.
- In a form-style report, which is generally best for printing a single record to a page, you usually create one column of field labels, and one column of data elements next to it. You may optionally place the field labels in the FORM structure and embellish it with graphic elements such as lines and boxes, creating a "template" inside which the data prints.
- For label reports, (such as a three-across format), you create a DETAIL placing the data for the label row-by-row. The width of the DETAIL should be one third the width of the detail print area.

  When the print engine prints the DETAIL, if it is less than the page width, it automatically prints from left to right, top to bottom, on a "best-fit" basis.
- 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. This provides word-wrap.
- For conditional text, you can create a DETAIL for each variation, then edit the executable code to PRINT the proper DETAIL

depending on a value from the current record.

Positioning and Alignment

You can set a specific position for the DETAIL as an offset relative to the last DETAIL printed. This allows you to exactly place, to the thousandth of an inch, the body of the report. By additionally setting the grid spacing, you can exercise precise control over every item in the report.

To set the position for the Detail, DOUBLE-CLICK in the Detail band. When the **Detail Properties** dialog appears, press the **Position** tab.

When the **Position** tab appears, enter new coordinates for **X, Y, Height,** and **Width**. These values set the AT attribute for the DETAIL structure.

The AT attribute on print structures performs two different functions, depending upon the structure on which it is placed.

When placed on a FORM, or page HEADER or FOOTER (not within a BREAK structure), the AT attribute defines the position and size on the page at which the structure is printed. The position specified by the *x* and *y* parameters is relative to the top left corner of the page.

When placed on a DETAIL, or group HEADER or FOOTER (contained within a BREAK structure) the print structure is printed according to the following rules (unless the ABSOLUTE attribute is also present):

- The width and height parameters of the AT attribute specify the minimum print size of the structure.
- The structure is actually printed at the next available position within the detail print area (specified by the REPORT´s AT attribute).
- 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 print structure on the page is printed at the top left corner of the detail print area (at the offset specified by its AT attribute).
- Next and subsequent print structures are printed relative to the ending position of the previous print structure:

  If there is room to print the next structure beside the previous structure, it is printed there.

  If not, it is printed below the previous.

The values contained in the AT attribute's *x*, *y*, *width*, and *height* parameters default to dialog units unless the THOUS, MM, or POINTS attribute is also present. Dialog units are defined as one-quarter the average character width by one-eighth the average character height. The size of a dialog unit is dependent upon 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 the printer's default font.

Tip: Use the Position dialog to precisely position Detail structures. When you position a structure on screen, the smallest unit you can move it is usually 1/96 inch. However, the dialog box allows you to specify position by thousandths of inches.

To set grid spacing for the entire report, choose **Option ➤ Grid Size**. In the **Grid Size** dialog, enter the space between each dot in thousandths of inches or your default measurement unit. You may enter a different value for the X and Y axes.

To turn grid snap on and off, choose **Option ➤ Snap to Grid**. Grid snap forces the handles of newly created objects to align themselves along the dot grid on screen. By default, grid snap is on.

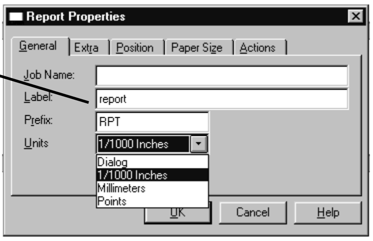
Page Orientation

To change the page orientation, choose **Edit ➤ Report Properties**, and press the **Paper Size** tab. Then check or uncheck the **Landscape** box. New reports default to portrait mode. Landscape means the printed text is parallel to the longest edges of the page. Portrait means the printed text is parallel to the shortest edges of the page.

Measurement Unit

To change the page measurement unit, choose **Edit ➤ Report Properties**, and press the **General** tab. Then select dialog units, thousandths of inches, millimeters, or points from the **Units** drop down list.

Specify the report name, default measurement unit, and page orientation in the Report Properties dialog.

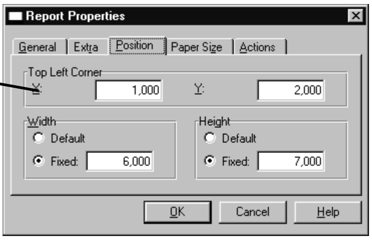


Job Name:
Label: report
Prefix: RPT
Units: 1/1000 Inches

Page Margins

The default margins for the *detail print area* are one inch from the left edge of the page, and two inches from the top. This setting leaves space for a HEADER at the top of the page. You specify the margins on the **Position** tab of the respective **Report Properties, Page/Group Header Properties, Detail Properties,** and **Page/Group Footer** dialogs.

Specify default margins in the Position dialog.



Top Left Corner
X: 1,000
Y: 2,000
Width
Default
(●) Fixed: 6,000
Height
Default
(●) Fixed: 7,000

❏ *To specify the left margin*, enter a value in the **X pos** box.

❏ *To specify the top margin*, enter a value in the **Y pos** box.

The units for the margin values are the measurement units you specify in the **Report Properties** dialog.

❏ To set the dimensions of each section, type the height and width on the **Position** tab:

❏ *To specify the height*, enter a value in the **Height** box.

❏ *To specify the width*, enter a value in the **Width** box.

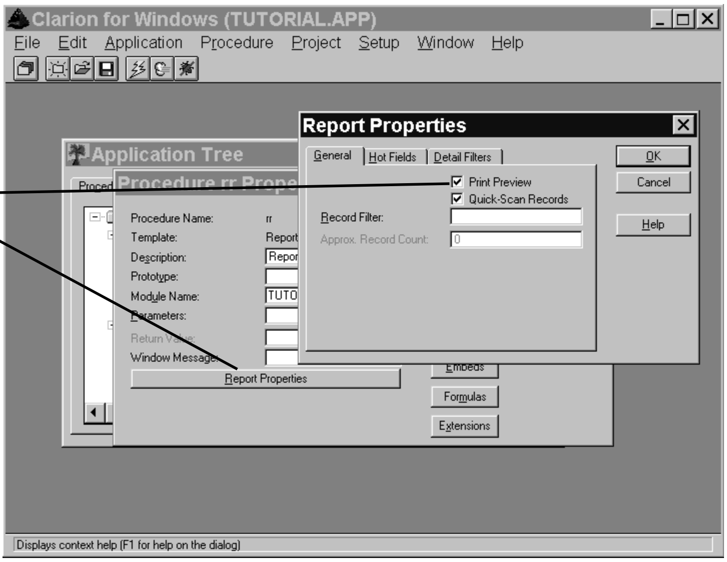
Print Preview

You can easily build in a “Print Preview” command in your applications. To do so, generate a report procedure with the Report template or the Report Wizard. Then, simply check the **Print Preview** box in the **Report Properties** dialog.

If you prefer to hand code your print preview process, see PREVIEW in the *Language Reference*, for more information and examples.

Implementing Print Preview in your application.

Press Report Properties, then check the Print Preview box.



Procedure Name: rr
Template: Report
Description: Repor
Prototype:
Module Name: TUTO
Parameters:
Return Value:
Window Message:

☑ Print Preview
☑ Quick-Scan Records
Record Filter:
Approx. Record Count: 0

Displays context help (F1 for help on the dialog)

PLACING STRUCTURES

Lay out your report structure by structure: FORM, HEADER, FOOTER, and DETAIL. Each of these structures appears in a separate band. To temporarily hide, then re-display a band, press the restore button on the right side of the band (this button has a double triangle on its face).

Form

To specify constant text or graphics which print on every page, place it in the FORM. Choose **Bands ➤ Page Form**. This is a free-floating section which can overlap the other sections.

Use the FORM as a layer, to 'hold' graphic frames or preprinted *forms* into which the data from the other sections 'fit.' Another use for the FORM is to hold a 'watermark,' which prints underneath the report.

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.

The FORM defaults to the same size as the page, less the margins.

The application 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 normal data fields, 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 of the report.

The FORM should guide the user to the data. You might use lines and boxes, for example, to divide the DETAIL into 'compartments,' grouping data and columns for the user. You may even create a 'greenbar paper' effect by alternating gray or light green color blocks.

Header

❏ *To specify text and data to compose at the start of each page,* place it in the HEADER.

Choose **Bands ➤ Page Header**. Typically, the HEADER includes a report title, and the page number. Optionally, it is a very useful place to display your company logo. This section describes some of the things you can do in a simple HEADER, incorporating these basic elements.

❑ *To set the font for any controls that appear in the header*:

1. RIGHT-CLICK in the HEADER band and choose **Font** in the popup menu.

   The **Select Font** dialog appears. Select typeface, size, style, color, and script from standard drop down lists. The **Select Font** dialog shows you a sample of the font you select. Normally, the default font is the default printer font. While this is excellent for the screen, a TrueType serif font, such as Times New Roman, is much better for hard copy.

2. Press the **OK** button.

❑ *To add a logo graphic to the HEADER*:

1. Choose **Controls ➤ Image**, or select the Image tool from the Controls toolbox, then CLICK at the top left hand corner of the HEADER band.

2. DOUBLE-CLICK on the image control you just placed.

   The **Image Properties** dialog appears.

3. Press the ellipsis (...) button to specify a **File** with the standard open file dialog.

   The image box appears as a square. Resize it, if necessary, by dragging a corner handle. Move the square, if necessary, by dragging the inside of the image box

4. Press the **OK** button.

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.

Think visually when designing your reports. A small picture or logo can create a professional looking report.

❑ *To add a rule the width of the report to the HEADER*:

1. Choose **Controls ➤ Line**, or select the Line tool from the Controls toolbox, and CLICK below the image box.

2. DOUBLE-CLICK on the line control you just placed.

   The **Line Properties** dialog appears.

2. In the **Line Properties** dialog, set the line **Width** to “**Full**.”

   Full indicates the line extends from margin to margin.

3, Set the line **Height** to **Fixed**, then type the zero in the blank field next to **Fixed**.

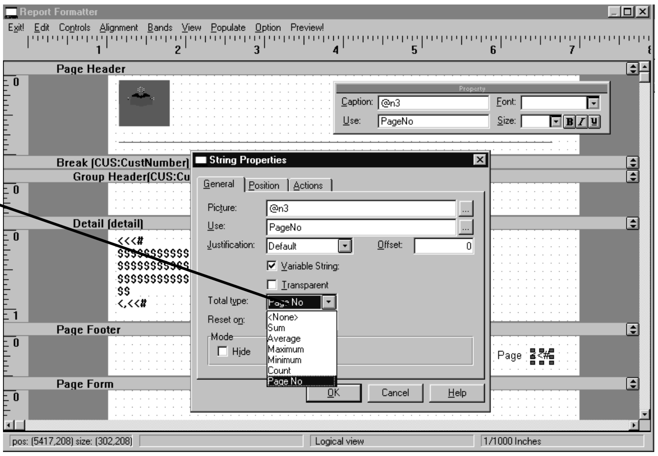
The zero indicates the line has no height—it's horizontal. Rules and lines should be key elements of your report layouts. They can help organize data for the user's eyes.

❏ *To place a control that prints the page number*.

1. Add a string variable control for the page number; choose **Controls ➤ String** or select the string tool from the Controls toolbox, CLICK in the header band.
2. DOUBLE-CLICK on the string control you just placed.

   The **String Properties** dialog appears.
3. In the **Parameter** field of the *String Properties* dialog, type @n3.
4. In the **Use Field**, type an equate label to reference the string in source code.
5. From the **Total type** drop down list, choose **Page No.**
6. Press the **OK** button to close the **String Properties** dialog.

Specifying a Page Number field.



Property: Caption: @n3 Use: PageNo Font: Size:

String Properties
Picture: @n3
Use: PageNo
Justification: Default Offset: 0
[✓] Variable String
[ ] Transparent
Total type: Page No
Reset on:
Mode: [ ] Hide
<None>
Sum
Average
Maximum
Minimum
Count
Page No

pos: (5417,208) size: (302,208) Logical view 1/1000 Inches

Footer

To specify text and data to compose at the end of each page, place it in the FOOTER. Choose **Bands ➤ Page Footer**. Placing controls in the FOOTER is similar to placing controls in the HEADER. You can just as easily duplicate the examples above in a FOOTER, and the example below in a HEADER.

❏ *To place a "Date Printed" in a FOOTER structure*:

1. Choose **Controls ➤ String**. Click in the FOOTER band.

2. DOUBLE-CLICK on the string control you just placed.

   The **String Properties** dialog appears.
3. In the **Parameter** field, type a date picture (@d1, for example).
4. In the **Use Field,** type a variable name.

   Assign the variable a value of TODAY().
5. Press the **OK** button to close the **String Properties** dialog.

Detail

To specify the data for the body of the report, place it in the DETAIL. Choose **Bands ➤ Detail.The** DETAIL carries most of the data. Additionally, the group HEADER and group FOOTER structures appear inside the DETAIL print area on the page.

When you work with the **Report Formatter** from the Text Editor, it will add another DETAIL structure every time you add a BREAK. You can delete the ones you don't use.

When working with the Application Generator, the REPORT will probably have one DETAIL.

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.

The most commonly used control for displaying data is the STRING control. You may fill the string with the contents of a data dictionary field by specifying the field identifier in the USE attribute. You can access the USE attribute in the **String Properties** dialog. You may also specify a display picture token to control the format of the displayed value.

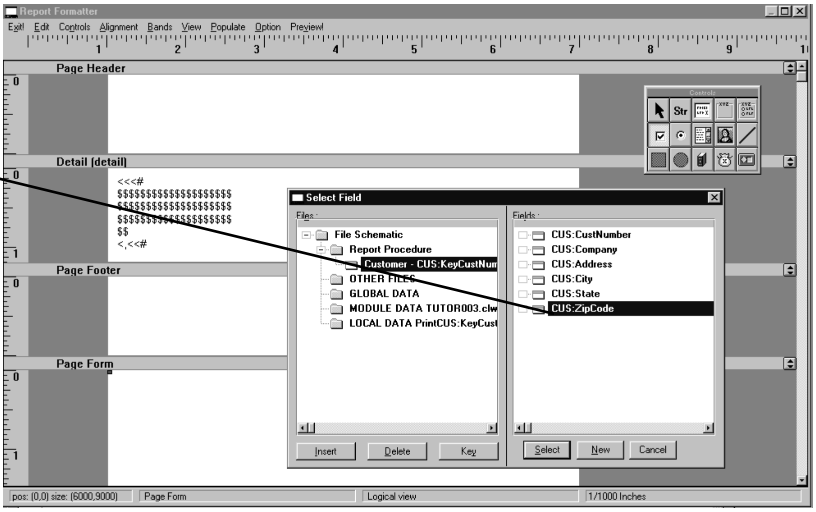
A report may have multiple DETAIL structures. This allows you to create alternate report layouts for a single report, then use control statements in the source code to allow the user to choose which to print at runtime. Each DETAIL structure requires its own PRINT statement. Multiple DETAIL structures also give you complete control over what prints where and when.

The typical steps required to place a data dictionary field in the DETAIL band when working within the Application Generator are:

1. Select the Dictionary Fields tool from the Controls toolbox then CLICK in the DETAIL band, or choose **Populate ➤ Dictionary Field** from the menu.
2. Select a field from the **Select Field** dialog.
3. CLICK inside the detail band.

   This places the control specified for this field in the data dictionary. A series of dollar signs ($) appears to show the control's print position and size.

Populating the Report.



That's all there is to placing a field in a report. There are many language elements to customize the report behavior and field formatting which the later part of this chapter will discuss.

Group Breaks

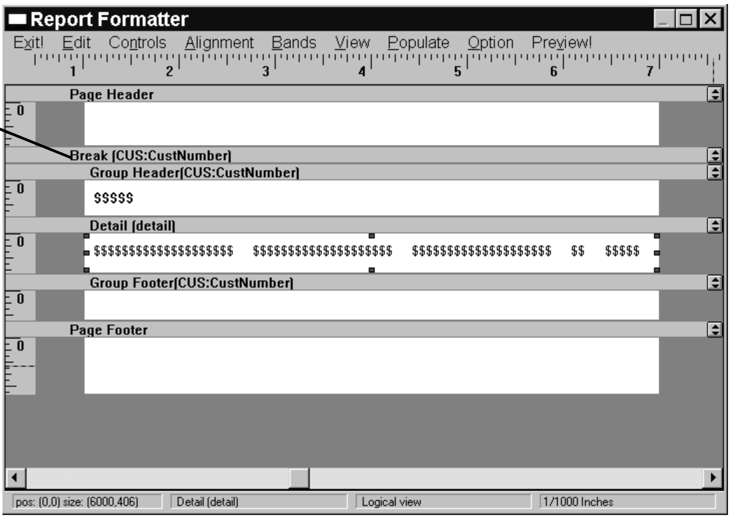
Group breaks provide a means of grouping report data into sections and optionally displaying subheadings, subtotals, or other information associated with the subgroup. Each group consists of a set of records, all sharing the same value in the BREAK field.

In order to generate meaningful groups, the records should be sorted in the same sequence as the BREAKs are declared. See *Sorting for Group Breaks* below.

Within a report, you may visually separate these groups, and add a subtotal or other summary information, above the group, below the group, or both. Group breaks are also called group bands by some popular database applications.

The **Report Formatter** displays group breaks in a tree structure, which allows you better visualize nested group breaks. The group break may contain the same elements as the report: a group HEADER, group DETAIL, and group FOOTER.

Dividing the report into bands with a Group BREAK structure.


Report Formatter
Exit! Edit Controls Alignment Bands View Populate Option Preview!
Page Header
Break (CUS:CustNumber)
Group Header(CUS:CustNumber)
$$$$$
Detail (detail)
Group Footer(CUS:CustNumber)
Page Footer
pos: (0,0) size: (6000,406)
Detail (detail)
Logical view
1/1000 Inches

Perhaps you wish to print an invoice report by customer. Set a BREAK on customer, and sort by customer. You can then summarize the invoice totals for each customer in alphabetical order. The report prints the group DETAIL, HEADER, and FOOTER for the set of records with the same customer. These structures all print inside the detail print area at the position you specify.

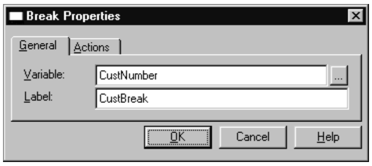
The print engine composes the group HEADER before the group DETAIL. The group HEADER is a good place to identify the group, for example, with a label saying “Customer:” followed by a variable string for the customer name field.

The group FOOTER, is composed after the group DETAIL. You can place a string saying “Total:” followed by a string variable which contains the field to be summed, with the SUM attribute.

o *To create a group break:*

1. Be sure the DETAIL band is visible; if not, press the restore button.
2. Choose **Bands ➤ Surrounding Break**.
3. When the cursor changes to a crosshair, CLICK in the DETAIL band.

   The **Break Properties** dialog appears.
4. In the **Break Properties** dialog, type a valid Clarion label to use as a name for the break.



Variable: CustNumber
Label: CustBreak

5. Type the name of a variable or field, including the prefix, to break on.

   You can press the ellipsis (...) button to select a break field from the **Select Field** dialog.

6. Press the **OK** button.

   This inserts the group BREAK. When the report prints, it groups together all records with the same value for the BREAK field, and prints any group HEADER and FOOTER defined for the break.

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.

7. Choose **Bands ➤ Group Header** from the menu to define a group HEADER for the BREAK.

8. When the cursor changes to a crosshair, CLICK in the BREAK mini caption bar.

   The **Page/Group Header Properties** dialog appears. Specify a field equate label and any special page breaking behavior. See *Page Breaks* below.

9. 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

Sorting for Group Breaks

The sort sequence of a file is determined by a KEY or INDEX defined in the Data Dictionary's **Field/Key Definition** dialog. Keys or indexes are selected for use in this particular report procedure, with the **File Schematic Definition** dialog. When you select a file and a key for your procedure, the key will determine the order in which you define your group breaks.

For example, if you report on the Customer file, and select the CUS:LastNameKey as your key, then your BREAK fields should be among those fields listed as components of the CUS:LastNameKey. You can see the key's component fields in the Data Dictionary's **Field/Key Definitions** dialog.

Further, you may specify more than one file to report on. You may specify a primary file and secondary files. The secondary files must be related to the primary file by a common field. These file relationships are defined in the Data Dictionary's **Relationship Properties** dialog. Adding secondary files to your procedure gives you another logical field to break on: that is, the common field(s) linking the two files.

❏ *To specify the sort sequence for your report*:

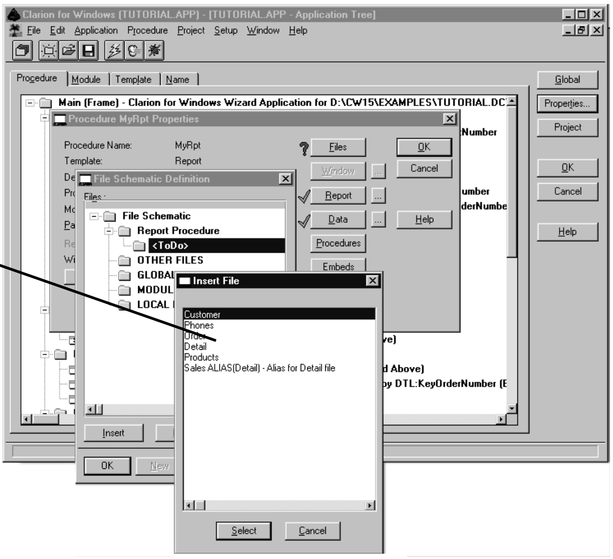
1. From the **Application Tree** dialog, DOUBLE-CLICK on the report procedure name.

   The **Procedure Properties** dialog appears.

2. Press the **Files** button.

   The **File Schematic Definition** dialog appears. Use this dialog to tell the Application Generator which files and keys your report procedure will access.

Specifying the file(s) you will report on.


Clarion for Windows (TUTORIAL.APP) - [TUTORIAL.APP - Application Tree]
File Edit Application Procedure Project Setup Window Help
Procedure Module Template Name
Global
Properties...
Project
OK
Cancel
Help
Main (Frame) - Clarion for Windows Wizard Application for D:\CW15\EXAMPLES\TUTORIAL.DC
Procedure MyRpt Properties
Procedure Name: MyRpt
Template: Report
Files
Window
Report
Data
Procedures
Embeds
OK
Cancel
Help
File Schematic Definition
Files :
File Schematic
Report Procedure
<ToDo>
OTHER FILES
GLOBAL
MODUL
LOCAL
Insert
OK
New
Insert File
Customer
Phones
Order
Detail
Products
Sales ALIAS(Detail) - Alias for Detail file
Select
Cancel

3. DOUBLE-CLICK the ToDo item for your procedure.

   The **Insert File** dialog appears.

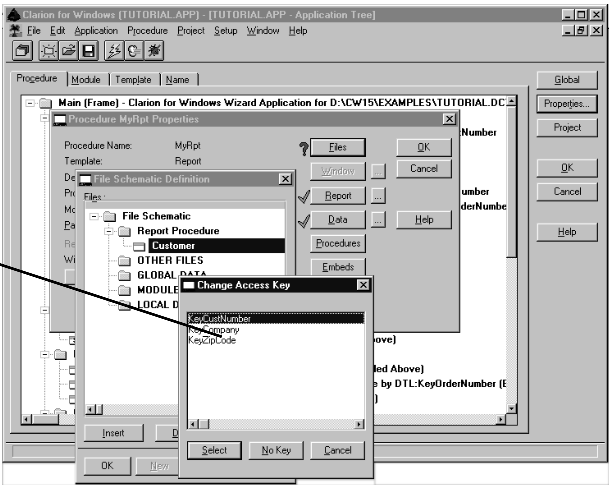
4. DOUBLE-CLICK the file you wish to report from.

   The **File Schematic Definition** dialog reappears.

5. Press the **Key** button, to specify which key is used for this procedure.

   The **Change Access Key** dialog appears.

Specifying the sort key for your report.


Clarion for Windows [TUTORIAL.APP] - [TUTORIAL.APP - Application Tree]
File Edit Application Procedure Project Setup Window Help
Procedure Module Template Name
Global
Properties...
Project
OK
Cancel
Help
Main (Frame) - Clarion for Windows Wizard Application for D:\CW15\EXAMPLES\TUTORIAL.DC
Procedure MyRpt Properties
Procedure Name: MyRpt
Template: Report
Files
Window
Report
Data
Procedures
Embeds
OK
Cancel
Help
File Schematic Definition
Files :
File Schematic
Report Procedure
Customer
OTHER FILES
GLOBAL DATA
MODULE
LOCAL D
Insert
OK
New
Change Access Key
KeyCustNumber
KeyCompany
KeyZipCode
Select
No Key
Cancel

6. DOUBLE-CLICK the key you want for this report.

   The **File Schematic Definition** dialog reappears.

7. Press the **OK** button.

Page Breaks

One of the main considerations when laying out a report is planning page breaks. You can't always predict how long or short a group will be. Therefore you should plan on how your report will behave when it reaches the end of the page, and there are still more items (in the group) to print.

There are several options available. The options are Page Before, Page After, With Next, and With Prior.

To access the dialog which allows you to set these options, DOUBLE-CLICK the DETAIL section. This displays the **Detail Properties** dialog, containing the options described below. You may also set the Page Before and Page After options for break HEADERs or FOOTERs. These options are available in the **Page/Group Header Properties** and **Page/ Group Footer Properties** dialogs as well.

PAGEBEFORE

To force a page break immediately before printing an item, check the **Page before** box in the respective section properties dialog. This sets the PAGEBEFORE attribute. Applicable to DETAILs, HEADERs, and FOOTERs. When applied to a DETAIL, this prints the full DETAIL starting at the top of a new page. Any associated FOOTER prints on the previous page.

You can then optionally specify the page number of the new page. The page number automatically increments, unless you reset it. To reset the page number to a value you specify, type it in the **New Page No** field.

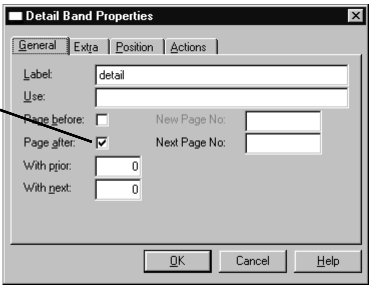
PAGEAFTER

To force a page break immediately after printing an item, check the **Page after** box in the respective section properties dialog. This sets the PAGEAFTER attribute. Applicable to DETAILs, HEADERs, and FOOTERs. When applied to a DETAIL, this prints the DETAIL and the FOOTER, then begins a new page.

You can then optionally specify the page number of the next page. The page number automatically increments, unless you reset it. To reset the page number to a value you specify, type it in the **Next 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 specify the PAGEAFTER attribute in the Detail Properties dialog.

Specifying page breaking behavior.



Label: detail
Use:
Page before: ☐ New Page No:
Page after: ☑ Next Page No:
With prior: 0
With next: 0

WITHNEXT

To prevent 'widow' elements in a printout, type a value in the **Keep next** field in the **Detail Properties** dialog. A 'widowed' print element is one which prints, but then is separated from the succeeding elements by a page break. Checking this box sets the WITHNEXT attribute, which is normally placed on a group HEADER to keep the HEADER on the same page as its associated details.

The value specifies the number of succeeding elements to print—a value of ‘1,’ for examples, specifies that the next element must print on the same page, else page overflow puts them both on the next page.

WITHPRIOR

To prevent ‘orphan’ elements in a printout, type a value in the **Keep prior** field in the **Detail Properties** dialog. An ‘orphaned’ print element is one which prints on a following page, separated from its related items. This sets the WITHPRIOR attribute, which is normally placed on a group FOOTER to keep the FOOTER on the same page as its associated details.

The value specifies the number of preceding elements to print—a value of “1,” for example, specifies that the previous element must print on the same page.

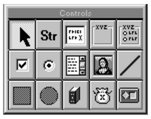
Tip: When placing subtotals or totals in a DETAIL, use the WITHPRIOR attribute to ensure they print with at least one member of their detail when a page break occurs.

PLACING CONTROLS IN A REPORT

This sections explains how to *place* a control in a report. The *Setting Report Control Properties* section below explains how to *customize* the controls you place in your reports.

The **Controls** Toolbox appears when you start the **Report Formatter**. Hide or re-display the **Controls** toolbox by choosing **Options ➤ Show Toolbox.** All the controls in the toolbox are also available from the **Controls** menu. The **Controls** Toolbox works exactly like a palette of drawing tools, such as the toolbox in the Windows Paintbrush accessory. To place a control:

1. CLICK on an icon in the toolbox, or choose a control from the menu.

   When you have selected a control, then pass the cursor over a report band, the cursor becomes a crosshair.

2. CLICK inside the band you wish to add the control to.

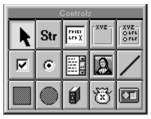
   The upper left hand corner of the control is placed at the intersection of the cursor crosshair when you CLICK the mouse.

3. If necessary, CLICK and drag on a control's *handle* to *resize* the control. CLICK and drag on the *interior* of the control to *move* the control.

Report Formatter Tools

Controls Toolbox

The **Report Formatter** contains a floating **Controls** toolbox, similar to the ones in the **Window Formatter**. Simply choose a control from the toolbox (CLICK on it), then CLICK in a report band to place the control in the report.



String	Allows you to place a STRING control on the report under construction. See *Setting String Control Properties*.
Text Field	Allows you to place a TEXT control on the report under construction. See *Setting Text Control Properties*.
Group Box	Allows you to place a GROUP control (group box) on the report under construction. See *Setting Group Box Control Properties*.
Option Box	Allows you to place an OPTION control (OPTION structure, which appears as a group box with radio buttons) on the report under construction. See *Setting Option Box Control Properties*.
Check Box	Allows you to place a CHECKBOX control on the report under construction. See *Setting Check Box Properties*.
Radio Button	Allows you to place a RADIO control on the report under construction. See *Setting Radio Button Properties*.

List Box	Allows you to place a LIST control (list box, or drop down list box) on the report under construction. See *Creating List Boxes* in the *Setting Control Properties* chapter.
Image	Allows you to place an IMAGE control (graphic image) on the report under construction. See *Setting Image Control Properties*.
Line	Allows you to place a LINE control on the report under construction. See *Setting Line Control Properties*.
Box	Allows you to place a BOX control on the report under construction. See *Setting Rectangle Control Properties*.
Ellipse	Allows you to place an ELLIPSE control on the report under construction. See *Setting Ellipse Control Properties*.
Dictionary Field	Allows you to select a field defined in the Data Dictionary, and place the control specified in the data dictionary, plus an associated PROMPT control, on the report under construction.
Custom Control	Allows you to place a CUSTOM control (Visual Basic custom control) on the report under construction. See *Setting Custom Control Properties*.
Control Template	Allows you to add one or more controls to your report, along with associated source code.

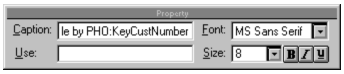
Display or hide the **Controls** toolbox by choosing **Options ➤ Show Toolbox.** All the controls in the toolbox are also available from the **Controls** menu. See *Placing Controls in a Report* above. Also see the *Setting Control Properties* chapter.

Tip: Position the cursor over any tool and wait for half a second. A tool tip appears telling you the type of control that will be created by this tool.

Property Toolbox

The **Report Formatter's Property** toolbox allows you to quickly specify the appearance and content of the *text* on each control within the report. Control the font, size, style, and content of all your text, using standard word processor buttons and drop down lists.

Display or hide the **Property** toolbox by choosing **Options ➤ 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.

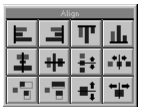


Align Toolbox

The **Report Formatter's Align** toolbox allows you to quickly, professionally, and precisely align the controls in your report. Select the controls to align (CTRL+CLICK allows you to select multiple controls), then click on the appropriate alignment tool. All the alignment actions are also available from the **Align** menu.

Display or hide the **Align** toolbox by choosing **Options ➤ 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.

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).

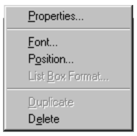
Align Horizontally	Along a vertical axis, aligns the centers of the selected controls with the center of the last control selected (red handles).
Align Vertically	Along a horizontal axis, aligns the centers of the selected controls with the center of the last control selected (red handles).
Spread Vertical	Equalizes the vertical spaces between the selected controls.
Spread Horizontal	Equalizes the horizontal spaces 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).
Center Vertical	As a group (relative positions of selected controls don't change), centers the selected controls horizontally within the report.
Center Horizontal	As a group (relative positions of selected controls don't change), centers the selected controls vertically within the report.

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.

Report Formatter Menus

Using the Popup Menu

Access the popup menu by RIGHT-CLICKING a band or a control. The popup menu on the **Report Formatter** allows you to manipulate and customize the report bands, *and* the controls in the report bands, depending on whether the band or a control is selected.



- ❏ To select a *control*, place the cursor on the control and RIGHT-CLICK.

❏ To select a *report band*, place the cursor anywhere on the band, but *not* on other controls, then RIGHT-CLICK.

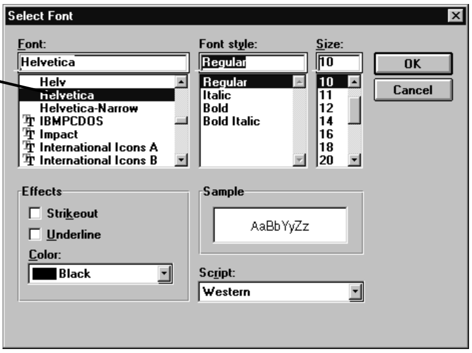
Tip: Many of the popup menu commands are also available on the Report Formatter Edit menu.

Following is a description of the popup menu choices.

Properties *To edit control or report band properties,* select a control or report band, and choose the **Properties** command. See *Setting Report Control Properties* below for more information. You may also DOUBLE-CLICK a control or report band, or RIGHT-CLICK and select the **Properties** command from the popup menu.

Font *To control the appearance of the text displayed in a control or report band,* select the control or band and choose the **Font** command. Specify font, size, style, script, and color from drop down list boxes. 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.

Specifying fonts for your report text.



Position	*To specify the position of a control or a report band*, select it and choose the **Position** command. *To position controls,* you will normally CLICK and DRAG the controls, use the **Align** tools, or both. However, you may use the **Position** command (and therefore the **Position Tab** of the various control properties dialogs) to position your controls. See the *Setting Control Properties* chapter for more information. Also see *Grid Settings* in the *Using the Options Menu* section below.
List Box Format	*To specify the appearance and functionality of a list box control*, select the list box and choose the **List Box Format** command. See the *Using the List Box Formatter* chapter for more information.
Delete	*To delete a control or a report band*, select it and choose the **Delete** command, or select it and press the DELETE key.
Duplicate	*To copy a control*, select it and choose the **Duplicate** command,.

Using the Edit Menu

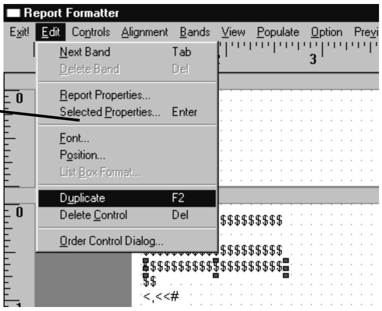
The **Edit** menu in the **Report Formatter** allows you to manipulate and customize the report and the controls in the report.

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	Not implemented in this release.
Delete Band	*To delete a report band*, select it and choose the **Delete Band** command, or select it and press the DELETE key. This deletes the band and all controls in it.
Report Properties	*To edit report properties*, choose the **Report Properties** command.

Selected Properties *To edit control or report band properties*, select a control or report band, and choose the **Selected Properties** command. See *Setting Report Control Properties* below for more information. You may also DOUBLE-CLICK a control or report band, or RIGHT-CLICK and select the **Properties** command from the popup menu.

The Report Formatter Edit Menu.



Font *To control the appearance of the text displayed in a control or report band*, select the control or band and choose the **Font** command. Specify font, size, style, script, and color from drop down list boxes. 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.

Position *To specify the position of a control or a report band*, select it and choose the **Position** command. *To position controls*, you will normally click and drag the controls, use the **Align** tools, or both. However, you may use the **Position** command (and therefore the **Position Tab** of the various control properties dialogs) to position your controls. See the *Setting Control Properties* chapter for more information. Also see *Grid Settings* in the *Using the Options Menu* section below.

List Box Format *To specify the appearance and functionality of a list box control*, select the list box and choose the **List Box Format** command. See the *Using the List Box Formatter* chapter for more information.

Delete Control *To delete a control*, select it and choose the **Delete Control** command, or select it and press the DELETE key.

Duplicate	*To copy a control*, select it and choose the **Duplicate** command.
Set Control Order	*To set the control order,* choose the **Set Control Order** command. This 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.

Using the Controls Menu

The **Controls** menu lists the controls that appear in the Controls Toolbox. Executing a command from the **Controls** menu is identical to clicking on the corresponding toolbox icon. The menu serves as a convenience.

For a list of toolbox controls, see the *Report Formatter Tools* section above. Also see the *Setting Control Properties* chapter.

Using the 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 serves as a convenience.

For a list of Alignment tools, see the *Report Formatter Tools* section above.

Using the Bands Menu

Page Header	*To add a page header band to your report,* choose **Page Header** from the menu. The HEADER structure traditionally prints at the top of each page of the report. Typically, you place the report title, graphics and other "introductory" elements in the HEADER.
Page Footer	*To add a page footer band to your report,* choose **Page Footer** from the menu. The FOOTER structure traditionally prints at the bottom of the report. Typically, you may place a page number, or totals in the FOOTER.

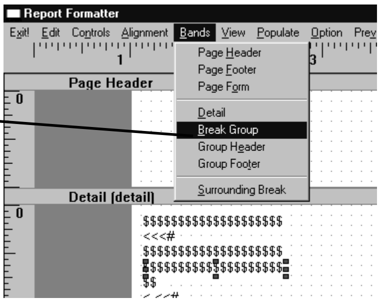
Page Form

To add a page form band to your report, choose **Page Form** from the menu. The FORM structure prints as a "background layer." Typically, you may display "overlays" such as graphics and field labels in the FORM layer, then print the actual data in the DETAIL. The FORM remains constant from page to page.

Detail

To add a detail band to your report, choose **Detail** from the menu. The DETAIL structure is the "body" of the report. It contains the basic data, either in table or record format.

The Report Formatter Bands Menu.



Break Group

To add a new detail, break, group header and group footer to your report, choose **Break Group** from the menu. Place the crosshair where you want the new group of bands to appear, and CLICK. The **Break Properties** dialog appears. Specify the variable to break on and press **OK.**

Group Header

To add a group header band to an existing break, choose **Group Header** from the menu. Place the crosshair on the caption bar of the break you wish to modify, and CLICK. The **Page/ Group Header Properties** dialog appears.

Group Footer

To add a group footer band to an existing break, choose **Group Footer** from the menu. Place the crosshair on the caption bar of the break you wish to modify, and CLICK. The **Page/Group Header Properties** dialog appears.

Surrounding Break	*To specify a break around an existing detail,* choose **Surrounding Break** from the menu. Place the crosshair on the detail you want to break on, and CLICK. The **Break Properties** dialog appears. Specify the variable to break on and press **OK.**

Using the View Menu

You can edit your report only in Band View. You may view your report as it appears on the printed page by choosing **View ➤ Page Layout View**, or **Preview!**. Page Layout View additionally allows you to move and resize the HEADER, FOOTER, DETAIL, and FORM sections by CLICKING and DRAGGING. See also *Using Preview!*

Page Layout View	*To view your report in page layout mode,* choose **View ➤ Page Layout View** from the action bar. Page Layout view allows you to select, relocate, and resize report sections by dragging their handles.
Band View	*To edit your report,* choose **View ➤ Band View** from the menu.
Expand Bands	*To close or open all the bands at once*, choose **View ➤ Expand Bands** from the menu.

Using the Populate Menu

The **Populate Menu** appears in the **Report Formatter** only when the Application Generator is active. It places a field or memory variable in the report, along with an appropriate control. For *fields*, the control *type* depends on how the field is defined in the data dictionary.

Dictionary Field	Allows you to place an entry control tied to a data dictionary field or a memory variable. When you CLICK in the report, the **Select Field** dialog appears. Select a field or variable, then CLICK in the report. If you specified a prompt for the field when creating the data dictionary, the first CLICK places the prompt for the control. The second CLICK places the control. 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 appears, rather than an entry box.

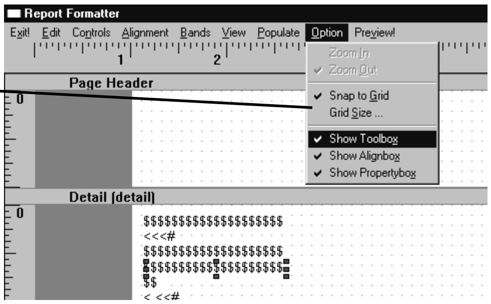
Control Template	Allows you to add one or more controls to your report, along with associated source code.

Using the Options Menu

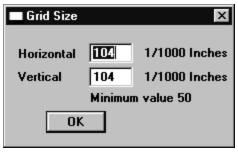
The **Options** menu allows you to display and hide the various **Report Formatter** tools and toolboxes.

Zoom In	Magnifies the “view” in Preview mode.
Zoom Out	Reduces the “view” in Preview mode.

The Report Formatter Options Menu.



Show Toolbox	*To toggle the Controls toolbox display on and off,* choose the **Show Toolbox** command. When designing large reports, it may be useful to hide the toolbox, gaining additional room for the report. You may still access all the control tools by choosing them from the **Control** menu.
Show Alignbox	*To toggle the Alignbox display on and off,* choose the **Show Alignbox** command. This is a matter of individual preference. You may still access all the alignment commands by choosing them from the **Alignment** menu.
Show Propertybox	*To toggle the Propertybox display on and off,* choose the **Show Propertybox** command.
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.



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.

Using Preview!

Preview! allows you to experiment with various report formats, without actually compiling and running the report. You can quickly “preview” alternative layouts for DETAILs, 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 in the report bands. Fonts, sizes, colors, and positions of report controls are all displayed.

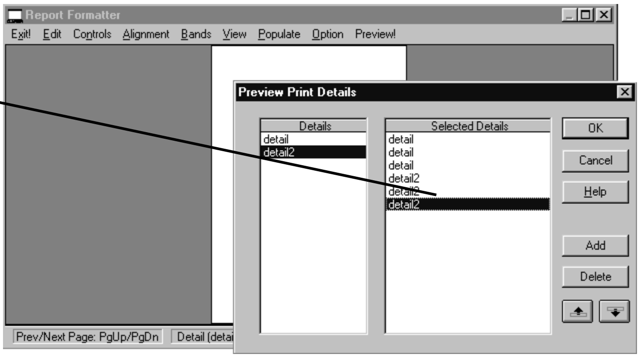
To generate a simulated report similar to the one that your user will see:

1. Choose the **Preview!** command.

   The **Preview Print Details** dialog appears. 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.

Experimenting with various detail formats and combinations.

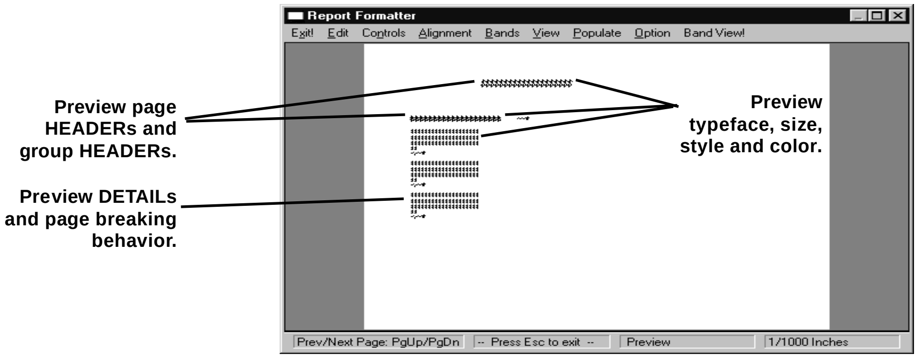


2. Press the **Add** button to generate a DETAIL placeholder.

Generate as many DETAIL placeholders 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.


Report Formatter
Exit! Edit Controls Alignment Bands View Populate Option Band View!
Preview page HEADERs and group HEADERs.
Preview DETAILs and page breaking behavior.
Preview typeface, size, style and color.
Prev/Next Page: PgUp/PgDn
-- Press Esc to exit --
Preview
1/1000 Inches

4. Choose **Options ➤ Zoom in** for a magnified view.

5. To exit **Preview!** mode, press ESC, or press **Band View!**

   The **Report Formatter** Band View reappears.

SETTING REPORT CONTROL PROPERTIES

Adding Text Labels and Fields

The most important part of the report creation process is laying out the fields—placing the data on the page. Clarion allows you to place strings (constants) and variable string controls on the page for 'basics.' You may also place other controls—graphics and specialized controls such as list boxes—in your report.

In most cases, setting control properties for a report is identical to setting control properties for a window. See the *Setting Control Properties* chapter for more information on each of the following controls.

Strings

Among the first items you place on the report will probably be titles or labels. STRING controls are useful for these elements.

❏ *To create column titles for a tabular report,* place them at the bottom of the HEADER, so that they appear at the top of each DETAIL section, above the data.

Tip: Be sure to line up the left handles of the data controls with the left handles of the title controls. Use the alignment tools, the grid snap feature, or both.

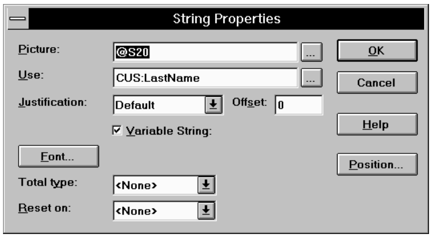
❏ *To create labels for a 'form' report,* place them alongside each field, within the DETAIL.

❏ To place a string control in your report:

1. Select the String tool from the Controls toolbox and CLICK in the detail, or choose **Controls ➤ String**. The cursor changes into a crosshair.

2. CLICK in the band which should contain the control. The center of the crosshair positions the upper left corner of the control.

   The **Report Formatter** places a STRING control in the report structure.



Picture: @S20
Use: CUS:LastName
Justification: Default
Offset: 0
☑ Variable String:
Total type: <None>
Reset on: <None>

3. DOUBLE-CLICK the control, or RIGHT-CLICK the control and choose **Properties** from the popup menu.

   When the **String Properties** dialog appears, fill it much as you would for window string controls (within the **Window Formatter**).

❏ *To specify "static" text for the string,* type the text you wish to display directly in the **Parameter** field.

❏ *To instruct the report to print a "dynamic" string,* place a picture token in the **Parameter** field, then specify a variable in the **Use** field, and check the **Variable string** box.

❏ *To center a title or label appearing over a column of data within a table,* choose **Centered** from the **Justification** drop down list.

❏ *To right align a title or label appearing over a column of numeric data within a table,* choose **Right Justified** from the **Justification** drop down list.

Multi-Line Text

A multi-line text control can print a long string (such as a memo) in the report for you, automatically word-wrapping and printing as many lines as the control's physical size allows.

For each variable string control to place in your report:

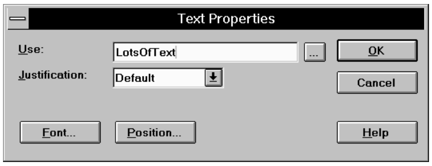
1. Select the Text tool from the Controls toolbox, or choose **Controls ➤ Text Field**.
2. CLICK in the band which should hold the control.

   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 displayed by this control.
3. Press the **Select** button.

   The **Report Formatter** places a TEXT control in the report structure.
4. DOUBLE-CLICK the control, or RIGHT-CLICK the control and choose **Properties** from the popup menu.

   When the **Text Properties** dialog appears, fill it much as you would for window text controls (within the **Window Formatter**).

   Probably the most important pieces of information to fill in for the text control are position and size. The simplest way to create a multi-line text box at the size you wish is to accept the defaults, press the **OK** button in the dialog, then resize the box by using the mouse to drag the handles. The onscreen rulers help verify the correct sizing. Alternatively, you may specify position and size on the **Position** tab.



Use: LotsOfText

Justification: Default

Tip: To specify an approximate number of lines to print, specify the Text control size in points, (the POINTS attribute). If, for example, the font size for the control is ten points, allow 20% extra for leading (the space between each line), Multiply the result (12 points) by the number of lines you wish. For example, for a five line control using 10 point type, type '60 POINTS' in the Fixed Height field. You must specify points as the measurement unit for the entire report. One point equals 1/72 inch.

Adding Graphic Controls

Graphic controls embellish the report and guide the reader's eye to the data. These controls allow you to add pictures and simple graphic lines to your report design.

In most cases, setting control properties for a report is identical to setting control properties for a window. See the *Setting Control Properties* chapter for more information on each of the following controls.

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 a bitmap is its size—Clarion automatically resizes the bitmap to fit the control size. This may introduce distortion if it shrinks or stretches the bitmap. The simplest way to prevent distortion is to keep the same ratio between the pixel resolution and the printed resolution.

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 one and a half inch box on the page.

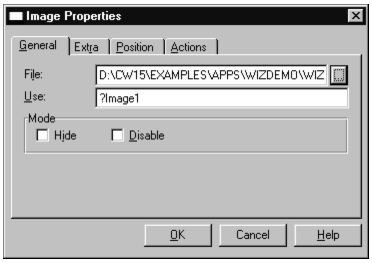
The Band View previews the image. You may even shrink and stretch the image by selecting it and dragging the handles.

To place an IMAGE control in a band:

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.

   The **Image Properties** dialog appears.



File: D:\CW15\EXAMPLES\APPS\WIZDEMO\WIZ

Use: ?Image1

Mode
☐ Hide ☐ Disable

4. In the **File** field, type the fully qualified image file name, or press the ellipsis (**...**) button to select the file from the standard open file dialog.

   Clarion automatically links the image file into the executable when the file is explicitly named in the control.

5. In the **Use** field, type a field equate label to refer to the image control in source code.
6. Press the **Position** tab.
7. Type the correct image size in the fixed **Width** and **Height** fields.
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.

   The **Line Properties** dialog appears.

4. In the **Use** field, type a field equate label to refer to the line control in source code.
5. Press the **Position** tab if you wish 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 **Height** 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 should be zero. In the **Width** 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 rule, or divider line useful for splitting a HEADER from the DETAIL section, for example, check Full Width, and set the Fixed Height to zero. To create a very thick rule, you can use a BOX control, also checking Full Width to place it across the page.

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. 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.

   The **Box Properties** dialog appears.
4. In the **Use** field, type a field equate label to refer to the control in source code.
5. Press the **Extra** tab.
6. Set the FILL attributes.

   If you want a solid box filled with color, check the **Fill** box, then press the **Fill Color** button and choose a color from the **Fill Color** dialog.
7. To give the box a border and choose its color, set the COLOR attribute.

   If you want a colored border, check the **Border** box, then press the **Border Color** button and choose a color from the **Border Color** dialog.

8. If you want rounded corners for the box, check the **Round** box to set the ROUND attribute.
9. To set the size of the box by typing in coordinates, press 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.

10. 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.

Remember, however, that the four position coordinates (AT attribute) specify the outer bounds of the ellipse. These define a box whose perimeter exactly contains the ellipse.

Adding Specialized Controls

The **Report Formatter** gives you the ability to print on the page virtually anything you can put on the screen. Just as a specialized control performs a function on screen—such as graphically portraying a "choice," like radio buttons—these same controls may perform the same function in your printed report.

In most cases, setting control properties for a report is identical to setting control properties for a window. See the *Setting Control Properties* chapter for more information on each of the following controls.

List Box

When the data you require for the report exists in a QUEUE, you may place a list box 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 Clarion provides so many list box formatting options, adding a LIST to the report allows you to create fancy "pigeonholes," suitable for columns and rows of items, for example.

At print time, the first time the code cycles through the QUEUE, it prints the LIST's header, and the first item in the queue. Each additional cycle prints the next member of the QUEUE, without repeating the header. The LIST's footer prints at the end of the QUEUE.

Printing a list box, rather than individual variable strings, is probably most useful when you've already formatted a LIST for the screen, such as a browse window.

To place a list in your 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 *Using the List Box Formatter* chapter. When you are finished, press the **OK** button.

   The **Report Formatter** places a LIST control in the report structure. Resize and relocate the box by dragging its handles or its interior.
3. DOUBLE-CLICK the list box you just placed.

   The **List Properties** dialog appears.
4. In the **Use** field, type a field equate label to refer to the control in source code.
5. Press the **Extra** tab.
6. In the **From** field, type the origin of the list box data—the QUEUE.
7. Press the **Position** tab to further specify the location and size of the list box.

Tip: To create a "quick and dirty" report from a browse list, choose Full for the Width and Height of the List box control within the DETAIL.

8. Press the **OK** button to close the **List Properties** dialog.

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 structure so that it does not 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.

   The **Option Properties** dialog appears.
4. In the **Parameter** 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. Press the **Extra** tab.
7. Uncheck 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 all the possible values for a single field in a record, and marking the one which is chosen.

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.

The **Radio Button Properties** dialog appears.

5. In the **Parameter** 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 where the control will belong.

   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.

   The **Check Box Properties** dialog appears.

5. In the **Parameter** 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 where the control will belong.

   The center of the crosshair positions the upper left corner of the check 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.

   The **Group Properties** dialog appears.
4. In the **Parameter** 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. Press the **Extra** tab.
7. Uncheck the **Boxed** box to hide the box, but not the internal controls.
8. Press **OK**.
9. Add additional controls to the group.

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.

   The **Custom Control Properties** dialog appears.
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.

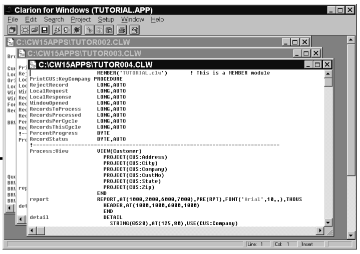
   The variable type depends on the .VBX control. The variable value is passed to the .VBX control. Please see the *Setting Control Properties* chapter for additional details.

7. Optionally, check the **Meta** box to print the control as a metafile.

USING THE TEXT EDITOR

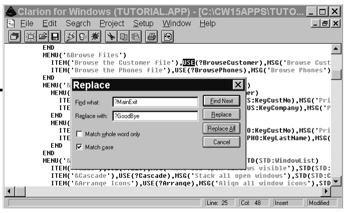
Contents

Whether using it to add Embedded Source Code in a procedure, or writing code "from scratch," the text editor is a full strength easy-to-use programmer's editor.



```
                    MEMBER('TUTORIAL.clw')       ! This is a MEMBER module
PrintCUS:KeyCompany PROCEDURE
RejectRecord         LONG,AUTO
LocalRequest         LONG,AUTO
LocalResponse        LONG,AUTO
WindowOpened         LONG,AUTO
RecordsToProcess     LONG,AUTO
RecordsProcessed     LONG,AUTO
RecordsPerCycle      LONG,AUTO
RecordsThisCycle     LONG,AUTO
PercentProgress      BYTE
RecordStatus         BYTE,AUTO
!---------------------------------------------------------------------------
Process:View         VIEW(Customer)
                       PROJECT(CUS:Address)
                       PROJECT(CUS:City)
                       PROJECT(CUS:Company)
                       PROJECT(CUS:CustNo)
                       PROJECT(CUS:State)
                       PROJECT(CUS:Zip)
                     END
report               REPORT,AT(1000,2000,6000,7000),PRE(RPT),FONT('Arial',10,,),THOUS
                       HEADER,AT(1000,1000,6000,1000)
                       END
detail                 DETAIL
                         STRING(@S20),AT(125,80),USE(CUS:Company)
```

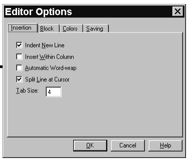
The text editor features color coded syntax highlighting to help make reading your code easier.


Clarion for Windows (TUTORIAL.APP) - [C:\CW15APPS\TUTO...
File Edit Search Project Setup Window Help
Replace
Find Next
Replace
Replace All
Cancel
Line: 25
Col: 48
Insert
Modified

```
        END
        MENU('&Browse Files')
          ITEM('Browse the Customer file'),USE(?BrowseCustomer),MSG('Browse Cust
          ITEM('Browse the Phones file'),USE(?BrowsePhones),MSG('Browse Phones')
        END
        MENU('&                                           )
          MENU(                                          er)
            ITE                                          S:KeyCustNo),MSG('Pri
            ITE                                          US:KeyCompany),MSG('P
          END
          MENU(
            ITE                                          O:KeyCustNo),MSG('Pri
            ITE                                          PHO:KeyLastName),MSG(
          END
        END
        MENU('&                                          TD(STD:WindowList)
          ITEM(                                          ws visible'),STD(STD:
          ITEM('&Cascade'),USE(?Cascade),MSG('Stack all open windows'),STD(STD:C
          ITEM('&Arrange Icons'),USE(?Arrange),MSG('Align all window icons'),STD
```

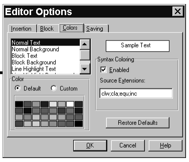
Find what: ?MainExit
Replace with: ?GoodBye
☐ Match whole word only
☑ Match case

The text editor includes full search and replace capabilities.



☑ Indent New Line
☐ Insert Within Column
☐ Automatic Word-wrap
☑ Split Line at Cursor
Tab Size: 4

Set editing options to your preferences.



Color: (● Default) ○ Custom
Syntax Coloring: ☑ Enabled
Source Extensions: clw;cla;equ;inc

Customize the color coded syntax highlighting to suit your preferences.

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. 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:

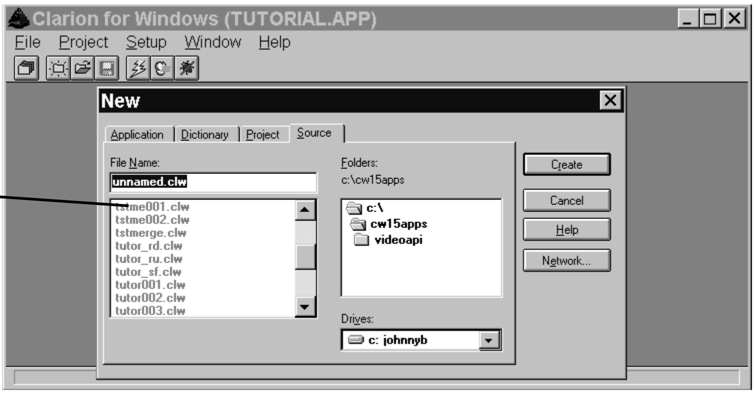
- Multiple Document Windows, in which you may edit as many documents as your system allows.
- Color coded syntax highlighting, which makes reading individual code lines easier. The color coding is fully customizable.
- Always available Search and Replace for any strings.
- Auto-indent, to make reading code easier.
- Next Error and Previous Error locator.
- Current cursor position (row and column), displays on the status bar.

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:

❑ Use the **File ➤ New** command, then select the **Source** tab in the **New** dialog. Navigate to your source directory and fill in the name of your new file in this standard dialog. Then press the **Create** button. This opens a blank source code document.

The New dialog allows you to create a new Clarion Source code document.

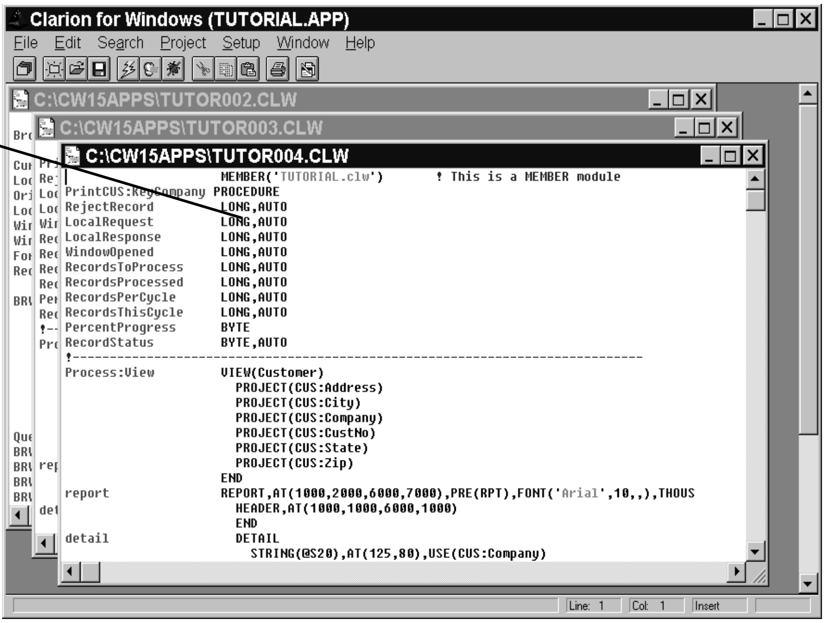


- ❏ Use the **File ➤ Open** command, select the **Source** tab, then DOUBLE-CLICK a source code file in the standard **Open File** dialog.
- ❏ Use the **File ➤ Pick** command to view your most recently edited files. Select the **Source** tab, highlight a source code file, then press the **Select** button.
- ❏ Within 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.

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:

The Editor's document window features color coded syntax highlighting.



```
                    MEMBER('TUTORIAL.clw')        ! This is a MEMBER module
PrintCUS:KeyCompany PROCEDURE
RejectRecord        LONG,AUTO
LocalRequest        LONG,AUTO
LocalResponse       LONG,AUTO
WindowOpened        LONG,AUTO
RecordsToProcess    LONG,AUTO
RecordsProcessed    LONG,AUTO
RecordsPerCycle     LONG,AUTO
RecordsThisCycle    LONG,AUTO
PercentProgress     BYTE
RecordStatus        BYTE,AUTO
!-------------------------------------------------------------------------------
Process:View        VIEW(Customer)
                      PROJECT(CUS:Address)
                      PROJECT(CUS:City)
                      PROJECT(CUS:Company)
                      PROJECT(CUS:CustNo)
                      PROJECT(CUS:State)
                      PROJECT(CUS:Zip)
                    END
report              REPORT,AT(1000,2000,6000,7000),PRE(RPT),FONT('Arial',10,,),THOUS
                      HEADER,AT(1000,1000,6000,1000)
                      END
detail                DETAIL
                        STRING(@S20),AT(125,80),USE(CUS:Company)
```

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.

Using the Edit Menu

To use an editing command, such as **Cut** or **Copy**, highlight the text you wish the command to act upon, 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.

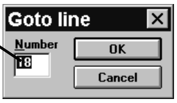
Select All

To highlight all the text in the document so that the next editing command affects the entire document, choose the **Select All** command.

Goto Line

To jump to a specific source code line to edit, choose **Goto Line**. The keyboard accelerator is CTRL+G.

Type a line number in the Goto Line dialog.



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. 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. 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

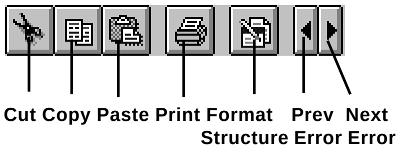
Think of this as 'visually editing' a window or report. Just place the insertion point on any line within the structure, and choose this command, or press CTRL+F. The toolbar button with the pencil and paper icon also activates this command. The Window Formatter (or Report Formatter) displays a visual 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.

Using The 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.



Using the 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.

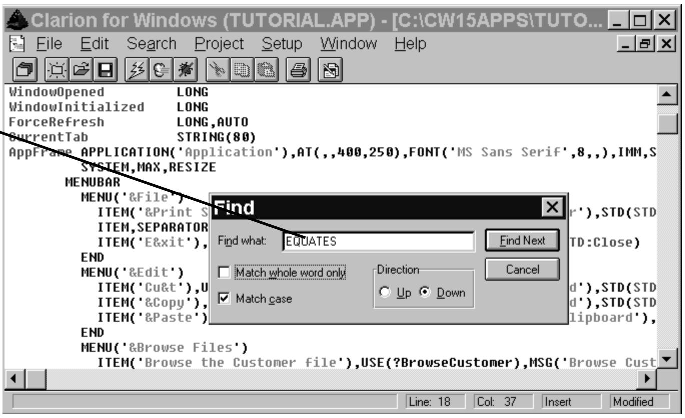
The commands on the **Search** menu are:

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.

The Find dialog, searching for “EQUATES.”



```
WindowOpened          LONG
WindowInitialized     LONG
ForceRefresh          LONG,AUTO
CurrentTab            STRING(80)
AppFrame APPLICATION('Application'),AT(,,400,250),FONT('MS Sans Serif',8,,),IMM,S
         SYSTEM,MAX,RESIZE
       MENUBAR
         MENU('&File')
           ITEM('&Print S                                          r'),STD(STD
           ITEM,SEPARATOR
           ITEM('E&xit'),                                          TD:Close)
         END
         MENU('&Edit')
           ITEM('Cu&t'),U                                          d'),STD(STD
           ITEM('&Copy'),                                          d'),STD(STD
           ITEM('&Paste')                                          lipboard'),
         END
         MENU('&Browse Files')
           ITEM('Browse the Customer file'),USE(?BrowseCustomer),MSG('Browse Cust
```

Find

Find what: EQUATES

[ ] Match whole word only

[✓] Match case

Direction: Up [✓ Down]

Find Next | Cancel

Line: 18 | Col: 37 | Insert | Modified

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.

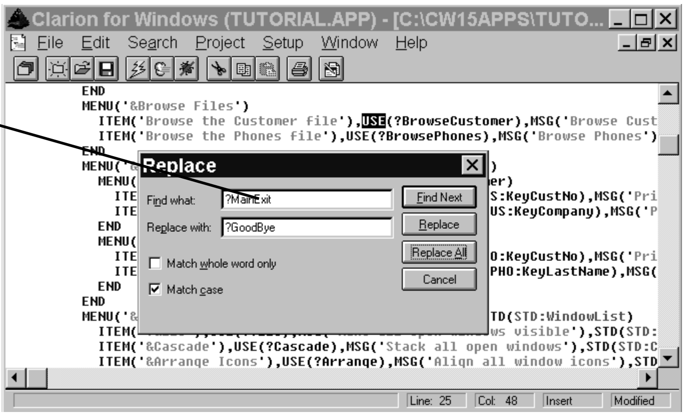
1. In the **Find What** field, type the original text to search for.

   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.

The Replace dialog — changing the name of an equate from ?MainExit to ?GoodBye.



```
    END
    MENU('&Browse Files')
      ITEM('Browse the Customer file'),USE(?BrowseCustomer),MSG('Browse Cust
      ITEM('Browse the Phones file'),USE(?BrowsePhones),MSG('Browse Phones')
    END
    MENU('&                                                )
      MENU(                                              er)
        ITE                                              S:KeyCustNo),MSG('Pri
        ITE                                              US:KeyCompany),MSG('P
      END
      MENU(
        ITE                                              O:KeyCustNo),MSG('Pri
        ITE                                              PHO:KeyLastName),MSG(
      END
    END
    MENU('&                                              TD(STD:WindowList)
      ITEM(                                              ws visible'),STD(STD:
      ITEM('&Cascade'),USE(?Cascade),MSG('Stack all open windows'),STD(STD:C
      ITEM('&Arrange Icons'),USE(?Arrange),MSG('Align all window icons'),STD
```

Find what: ?MainExit

Replace with: ?GoodBye

[ ] Match whole word only

[✓] Match case

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.

Using the File Menu

The Text Editor **File** menu has the following special file oriented commands.

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 currently active source code document, at the insertion point.

Export Block

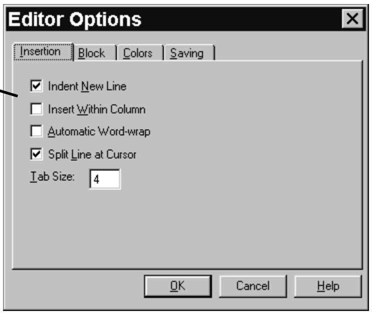
Saves the currently selected text in a new source code document under a new name which you specify.

CUSTOMIZING THE TEXT EDITOR

To personalize your editing environment, customize appearance and cursor behavior with the *Editor Options* dialog. To view the dialog, choose **Setup ➤ Editor Options**. Select the corresponding tab to set specific Text Editor options.

Insertion Options

The Editor Options dialog, with custom preferences set.



☑ Indent New Line
☐ Insert Within Column
☐ Automatic Word-wrap
☑ Split Line at Cursor
Tab Size: 4

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.

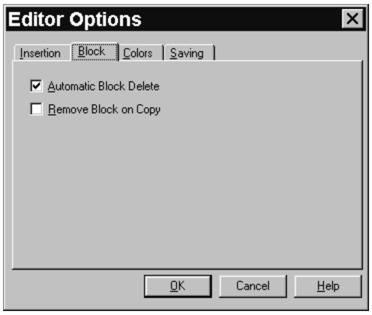
Insert Within Column
When the insertion point is in the middle of a line, ENTER adds a new line after the current line.

Automatic Word-wrap
To cause automatic line breaks at column 70, check this box.

Split Line at Cursor When this box is checked, ENTER will split the current line at the insertion point (cursor). The second part of the line will appear on 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 Options



[✓] Automatic Block Delete
[ ] Remove Block on Copy

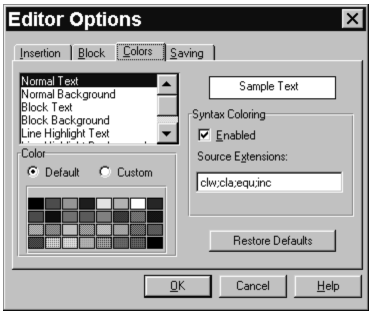
Automatic Block Delete
To delete the selected text when pasting, check this box. To insert before a selected block, *uncheck* the box.

Remove Block On Copy
To delete the selected text when copying, check this box.

Color Options

These options allow you to 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.



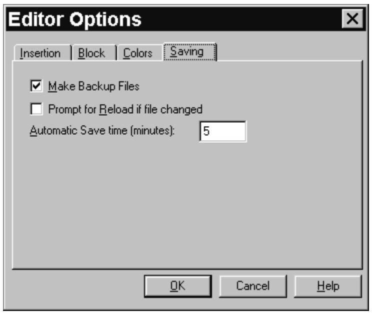
Normal Text
Normal Background
Block Text
Block Background
Line Highlight Text

Color: [✓ Default] Custom

Syntax Coloring
[✓] Enabled
Source Extensions: clw;cla;equ;inc

Color Groups	*Highlight the language or text element to receive a color assignment.*
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.

Save Options



☑ Make Backup Files
☐ Prompt for Reload if file changed
Automatic Save time (minutes): 5

Make Backup Files

To cause the Text Editor 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.

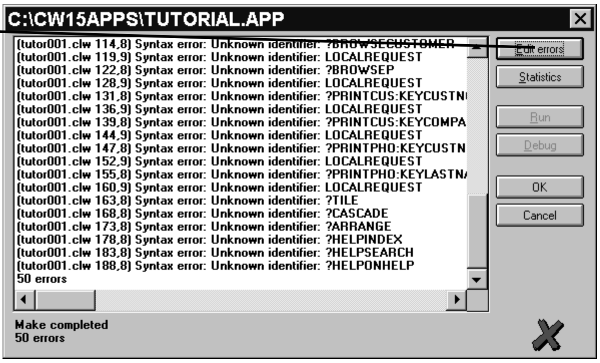
Automatic Save time (minutes)

To specify the time interval between automatic saves, type a number in this box.

EDITING ERRORS

One of the chief entry points into the Text Editor is through the compilation results dialog—when an error aborts the make.

The Edit Errors button in the compile results dialog opens the text editor.



```
(tutor001.clw 114,8) Syntax error: Unknown identifier: ?BROWSECUSTOMER
(tutor001.clw 119,9) Syntax error: Unknown identifier: LOCALREQUEST
(tutor001.clw 122,8) Syntax error: Unknown identifier: ?BROWSEP
(tutor001.clw 128,9) Syntax error: Unknown identifier: LOCALREQUEST
(tutor001.clw 131,8) Syntax error: Unknown identifier: ?PRINTCUS:KEYCUSTN
(tutor001.clw 136,9) Syntax error: Unknown identifier: LOCALREQUEST
(tutor001.clw 139,8) Syntax error: Unknown identifier: ?PRINTCUS:KEYCOMPA
(tutor001.clw 144,9) Syntax error: Unknown identifier: LOCALREQUEST
(tutor001.clw 147,8) Syntax error: Unknown identifier: ?PRINTPHO:KEYCUSTN
(tutor001.clw 152,9) Syntax error: Unknown identifier: LOCALREQUEST
(tutor001.clw 155,8) Syntax error: Unknown identifier: ?PRINTPHO:KEYLASTN
(tutor001.clw 160,9) Syntax error: Unknown identifier: LOCALREQUEST
(tutor001.clw 163,8) Syntax error: Unknown identifier: ?TILE
(tutor001.clw 168,8) Syntax error: Unknown identifier: ?CASCADE
(tutor001.clw 173,8) Syntax error: Unknown identifier: ?ARRANGE
(tutor001.clw 178,8) Syntax error: Unknown identifier: ?HELPINDEX
(tutor001.clw 183,8) Syntax error: Unknown identifier: ?HELPSEARCH
(tutor001.clw 188,8) Syntax error: Unknown identifier: ?HELPONHELP
50 errors

Make completed
50 errors
```

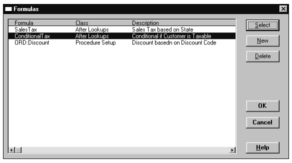
The **Edit errors** button automatically calls the Text Editor, and places the insertion point at the position where the compiler detected the error. You may then edit the source code to correct the mistake. The **Edit ➤ Goto Next Error** command is available to jump to the next compilation error once you correct the first error.

Note: When entering the Text Editor through the compilation results dialog, any changes made to generated code will be overwritten by the next project generate or make.

USING THE FORMULA EDITOR

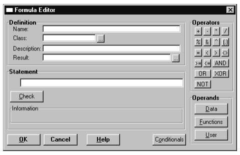
Contents

The Formula Editor helps you to quickly generate a statement assigning an expression to a variable. You can use the Formula Editor to create computed fields or conditional fields.

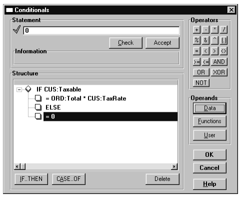


Formula	Class	Description
SalesTax	After Lookups	Sales Tax based on State
ConditionalTax	After Lookups	Conditional if Customer is Taxable
ORD:Discount	Procedure Setup	Discount basedn on Discount Code

The Formulas dialog organizes and manages all the formulas for your procedure.



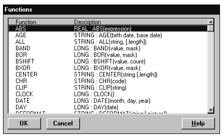
The Formula Editor creates valid expressions at the push of a button.



Statement: 0

```
IF CUS:Taxable
  = ORD:Total * CUS:TaxRate
  ELSE
  = 0
```

The Conditionals dialog creates control structures for conditional formulas.



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)

The Functions dialog provides quick access to all Clarion functions.

The Formula Editor helps you to quickly generate a statement assigning an expression to a variable. You can use the Formula Editor to create **computed fields** or **conditional fields**.

- A **computed field** receives the evaluation of an expression. In other words, a computed field is the receiving end of a simple assignment statement: variable = expression. For example, a computed field called GrossPrice might receive the result of adding two fields called BasePrice and Tax.

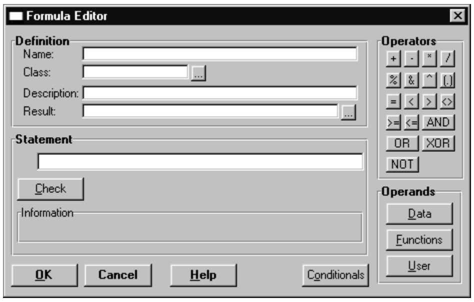
  You can use a computed field wherever the program must perform a calculation.

- A **conditional field** is a computed field with multiple possible assignments. There are two types of conditional fields—IF structures and CASE structures. 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.

  You can use a conditional field wherever the program must perform different calculations based on a condition.

The **Formula Editor** dialog provides access to data dictionary fields, as well as global and local memory variables, and facilitates creating 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, and check the syntax upon completion.



EXPRESSION COMPONENTS

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 components of an 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 a combination of more than one component, such as a function and its parameters.

The **Formula Editor** allows you to choose operators and operands, then insert them into the **Statement** line.

The table below lists all the components used in 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).
&	Ampersand: Appends one text string to another.
^	Caret: Raises one operand to the power of the other.
( )	Parentheses: Groups components together within an expression.

Logical 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.

Operands :

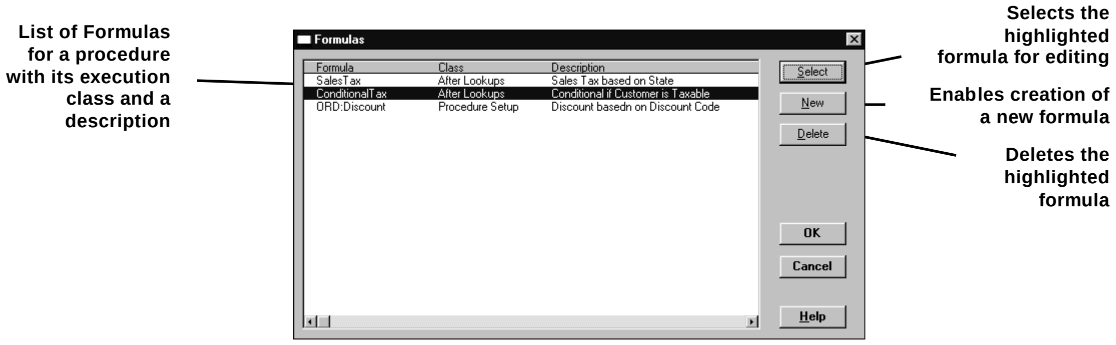
Data	Includes data dictionary fields, global and local memory variables.
Functions	All of the built-in functions of the Clarion programming language. These functions all perform some operation on parameters (other operands) and return a value to the expression.
User	Any FUNCTION in your application. These functions perform some operation on parameters (other operands) and return a value to the expression.
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).

FORMULA EDITOR TOOLS

The Formula Editor consists of three dialog boxes:

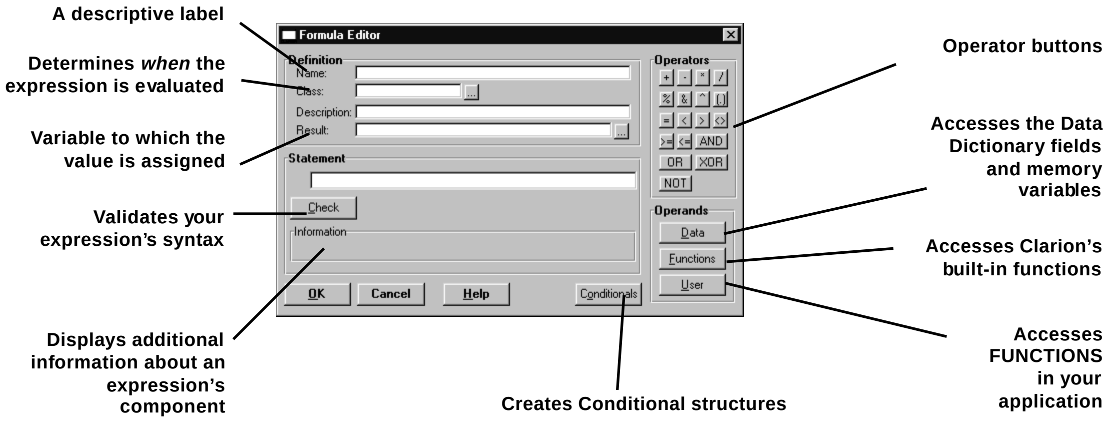
Formulas	Manages all the formulas you have created for the procedure.
Formula Editor	Creates simple assignment statements.
Conditionals	Creates conditional structures (IF..THEN or CASE..OF).

Formulas Dialog

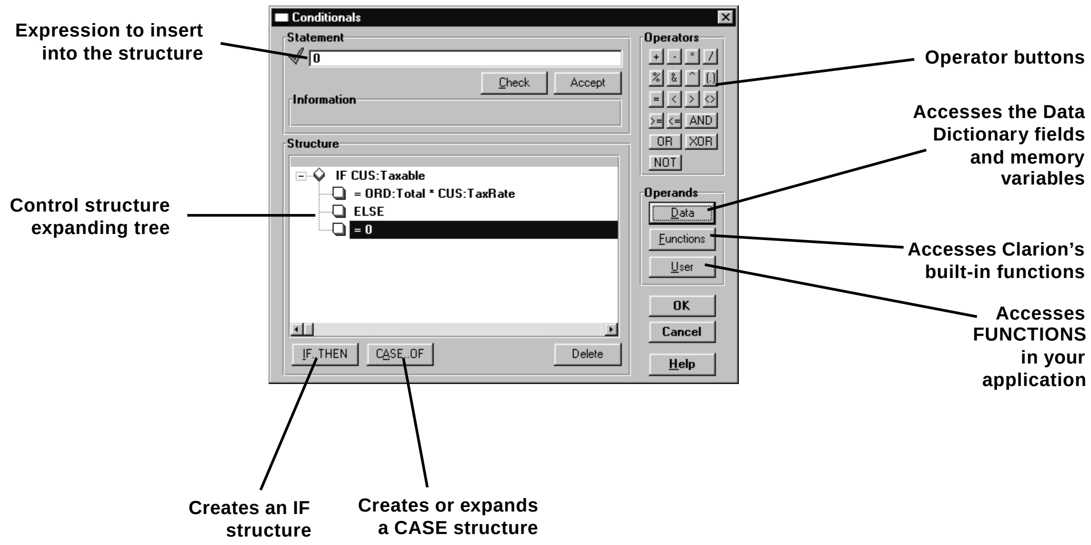

List of Formulas for a procedure with its execution class and a description
Formulas
Select
New
Delete
OK
Cancel
Help
Selects the highlighted formula for editing
Enables creation of a new formula
Deletes the highlighted formula

Formula	Class	Description
SalesTax	After Lookups	Sales Tax based on State
ConditionalTax	After Lookups	Conditional if Customer is Taxable
ORD:Discount	Procedure Setup	Discount basedn on Discount Code

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
Formula Editor
Definition
Name:
Class:
Description:
Result:
Statement
Check
Information
OK
Cancel
Help
Conditionals
Operators
+ - * /
% & ^ ( )
= < > <>
>= <= AND
OR XOR
NOT
Operands
Data
Functions
User
Operator buttons
Accesses the Data Dictionary fields and memory variables
Accesses Clarion's built-in functions
Accesses FUNCTIONS in your application
Creates Conditional structures

Conditionals Dialog


Conditionals
Statement
0
Check
Accept
Information
Structure
IF CUS:Taxable
= ORD:Total * CUS:TaxRate
ELSE
= 0
IF..THEN
CASE..OF
Delete
Operators
+ - * /
% & ^ (.)
= < > <>
>= <= AND
OR XOR
NOT
Operands
Data
Functions
User
OK
Cancel
Help
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

CREATING AN ASSIGNMENT EXPRESSION

❑ From the **Procedure Properties** dialog:

1. Press the **Formulas** button.

   If you already have formulas in the procedure, the **Formulas** dialog appears.

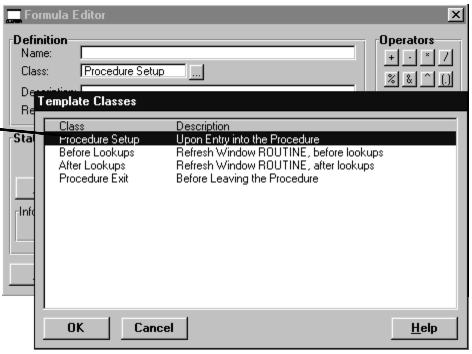
   If this is the first formula in this procedure, the **Formulas** dialog will not appear. The **Formula Editor** dialog appears, so skip step 2.

2. Press the **New** button.

   The **Formula Editor** dialog appears.

3. In the **Name** field, type a name 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

Choosing a formula class.

A formula class determines *where* in the generated source code, its 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.

Note: Do not confuse formula classes with template classes. A *template* class is simply a group of templates, Clarion or third-party, within the template registry. A *formula* class is a point within a procedure template where the formula is evaluated.

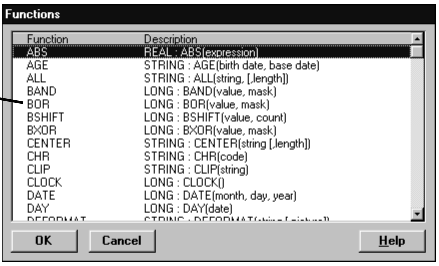
5. In the **Description** field, type a description of the formula.

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 name appears in the **Formulas** dialog list.

7. In the **Statement** field, create your formula.

   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.



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.

8. 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.

9. Press the **OK** button.

CONDITIONAL EXPRESSIONS

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.

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.

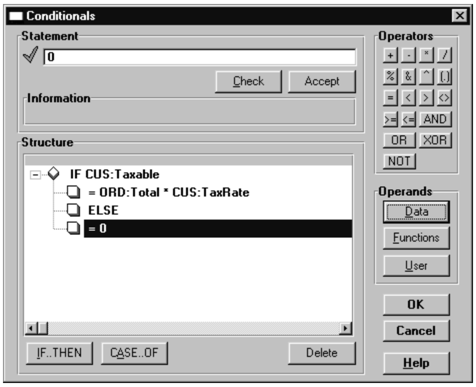
A CASE structure selectively assigns a value to the Result variable based on the evaluation of multiple OF expressions against the CASE expression. Practically speaking, there are unlimited alternative assignments because any number of expressions may be evaluated. See the *Language Reference* for more information.

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 conditional formula (from the **Procedure Properties** dialog):

1. Press the **Formulas** button.

   If you already have formulas in the procedure, the **Formulas** dialog appears.

   If this is the first formula in this procedure, the **Formulas** dialog will not appear. The **Formula Editor** dialog appears, so skip step 2.

2. Press the **New** button.

   The **Formula Editor** dialog appears.

3. In the **Name** field, type a name for the formula.

4. Press the ellipsis (...) button next to the **Class** field to choose a Formula Class.

   A formula class determines *where* in the generated source code its 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.

Note: Do not confuse formula classes with template classes. A *template* class is simply a group of templates, Clarion or third-party, within the template registry. A *formula* class is a point within a procedure template where the formula is evaluated.

5. In the **Description** field, type a description of the formula.

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 name appears in the **Formulas** dialog list.

7. Press the **Conditionals** button.
8. Press the **IF..THEN** button.

   The structure appears in the **Structure** window.
9. 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.
10. Press the **Check** button to check your syntax.
11. Press the **Accept** button to insert your expression into the structure.
12. Highlight the line below the IF line in the **Structure** window.

    This is where the “True” assignment expression goes.
13. On the **Statement** line, enter the “True” assignment expression.

    Again, 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 Result variable.

    A “true” assignment expression is *not* required. If no assignment is entered, then no assignment is made.
14. Press the **Check** button to check your syntax.
15. Press the **Accept** button to enter your expression into the structure.
16. Highlight the line below the ELSE line in the **Structure** window

    This is where the “False” assignment expression goes.
17. On the **Statement** line, insert the “False” assignment expression.

    Again, 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 Result variable.

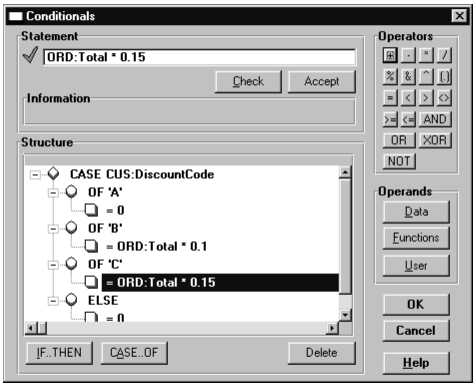
    A “false” assignment expression is *not* required. If no assignment is entered, then no assignment is made.
18. Press the **Check** button to check your syntax.
19. Press the **Accept** button to insert your expression into the structure.
20. When your control structure is complete, press the **OK** buttons in the **Conditionals, Formula Editor, and Formulas** dialogs.

Creating a CASE Structure

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 customers 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 in the illustration 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 conditional formula (from the **Procedure Properties** dialog):

1. Press the **Formulas** button.

   If you already have formulas in the procedure, the **Formulas** dialog appears.

   If this is the first formula in this procedure, the **Formulas** dialog will not appear. The **Formula Editor** dialog appears, so skip step 2.

2. Press the **New** button.

   The **Formula Editor** dialog appears.

3. In the **Name** field, type a name for the formula.

4. Press the ellipsis (...) button next to the **Class** field to choose a Formula Class.

   A formula class determines *where* in the generated source code its 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.

   Note: Do not confuse formula classes with template classes. A *template* class is simply a group of templates, Clarion or third-party, within the template registry. A *formula* class is a point within a procedure template where the formula is evaluated.

5. In the **Description** field, type a description of the formula.
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 name appears in the **Formulas** dialog list.

7. Press the **Conditionals** button.
8. Press the **CASE..OF** button.

   The CASE structure appears in the **Structure** window.

9. On the **Statement** line, enter the CASE expression that 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.

10. Press the **Check** button to check your syntax.
11. Press the **Accept** button to insert your expression into the structure.
12. Highlight the OF line below the CASE line in the **Structure** window.

    This is where the first OF *comparison* expression goes.

13. On the **Statement** line, enter the OF *comparison* expression.

    Again, 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 assignment expression is evaluated and the resulting value is assigned to the Result variable.

14. Press the **Check** button to check your syntax.
15. Press the **Accept** button to insert your expression into the structure.

16. Highlight the line below the OF line in the **Structure** window.

    This is where the first OF *assignment* expression goes.

17. On the **Statement** line, insert the OF *assignment* expression.

    Again, 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 Result variable.

18. Press the **Check** button to check your syntax.

19. 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.

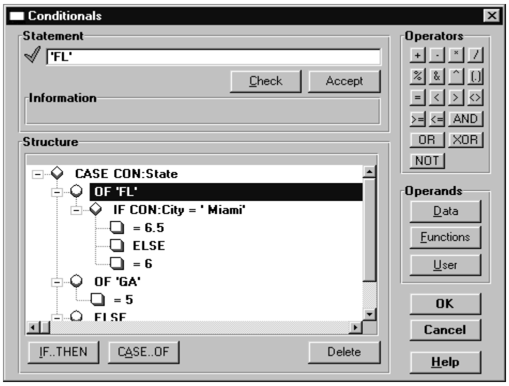
Nesting Control Structures

Either of the available control structures can be nested inside another. This enables you to easily create very complex structures.

Lets say you wanted to determine the TaxRate based on the following logic:

> If the state is Florida, the TaxRate is 6 %, unless the City is Miami, which charges 6.5 %.
>
> If the State is Georgia, the Tax Rate is 5%.

You would create a CASE structure, with a nested IF structure as displayed below.



```
CASE CON:State
  OF 'FL'
    IF CON:City = ' Miami'
      = 6.5
      ELSE
      = 6
  OF 'GA'
    = 5
  ELSE
```

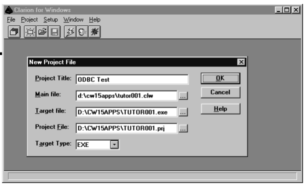
❑ To add a nested control structure:

1. Highlight an assignment line in the **Structure** window.
2. Press either the **CASE..OF** or **IF..THEN** button
3. Insert expressions on the appropriate lines following the instructions in the previous sections.

USING THE PROJECT SYSTEM

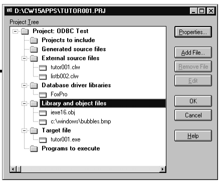
Contents

This chapter shows you how to use the Project System. The Project System controls compile and link options. This chapter also contains information on distributing your applications.



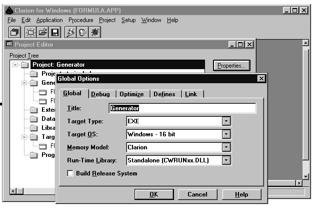
Project Title: ODBC Test
Main file: d:\cw15apps\tutor001.clw
Target file: D:\CW15APPS\TUTOR001.exe
Project File: D:\CW15APPS\TUTOR001.prj
Target Type: EXE

Start by giving the project a name. When you select your main source code file from a directory list, Clarion automatically fills in most of the dialog box items.



Project: ODBC Test
- Projects to include
- Generated source files
- External source files
  - tutor001.clw
  - listb002.clw
- Database driver libraries
  - FoxPro
- Library and object files
  - iexe16.obj
  - c:\windows\bubbles.bmp
- Target file
  - tutor001.exe
- Programs to execute

Add, delete and edit properties for source code, external libraries, and object files.



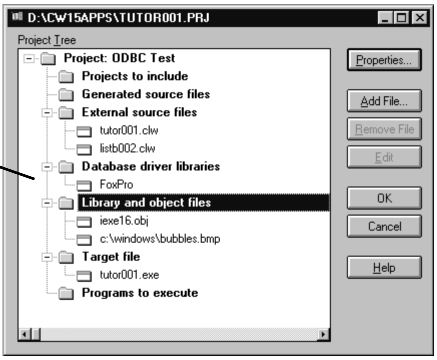
Title: Generator
Target Type: EXE
Target OS: Windows - 16 bit
Memory Model: Clarion
Run-Time Library: Standalone (CWRUNxx.DLL)
Build Release System

Set global compiler options and optimizations.

The project file (.PRJ) 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 the project file to tell them how, and what, to compile and link. The project file is functionally equivalent to a MAKE file for other language compilers.

The Clarion Project System visually manages the project file. It maintains tree diagrams of the source files, external libraries, resources, and other project components.

The Project Editor dialog contains the Project Tree list. The tree controls expand and contracts when you click them. When expanded, they list the component files. When contracted, the box contains a cross to show that you can expand the control.



Project: ODBC Test
- Projects to include
- Generated source files
- External source files
  - tutor001.clw
  - listb002.clw
- Database driver libraries
  - FoxPro
- Library and object files
  - iexe16.obj
  - c:\windows\bubbles.bmp
- Target file
  - tutor001.exe
- Programs to execute

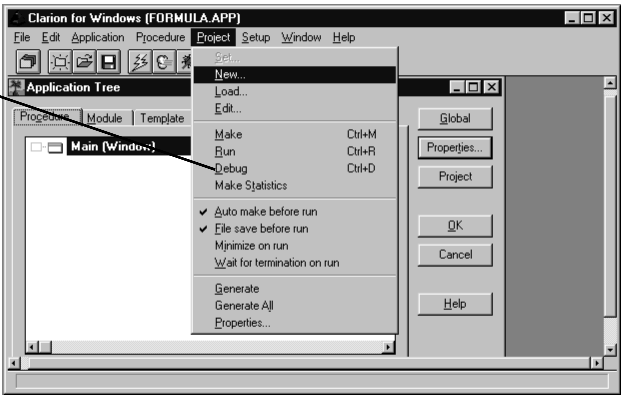
This chapter discusses the Project System and related topics. It will:

- Describe the various menu commands that set Project System options.
- Show you how to add your source code files to the Project Tree.
- Show you how to add external libraries to the Project Tree, and how to access their functions and procedures in your source code.
- Show you how to specify the target file and set other compiler options. The target file is the ultimate executable created for your application.

THE PROJECT MENU

Clarion's development environment menu provides several commands which affect or access the Project System. This section provides a list of the commands and what they do. To access these commands, choose **Project** from the action bar.

The Project Menu.



Set	Makes a project current, so that subsequent project commands such as make and run operate on the selected project. The **Select Project** dialog appears; select a .PRJ or .APP file from the list box.
New	Creates a new project. Fill in the **Project Title** and **Main file** fields in the **New Project File** dialog.
Load	Makes a project current, so that subsequent project commands such as make and run operate on the selected project. If no project is current, the **Select Project** dialog appears; select a .PRJ or .APP file from the list box. Displays either the **Application Tree** dialog or the **Project Editor** dialog depending on whether a .PRJ or .APP file was selected.
Edit	Allows you to edit the current project file with the **Project Editor** dialog.
Make	Allows you to compile and link the current project. You can also press the **Make** button on the toolbar.
Run	Allows you to compile, link, and run the current project.
Debug	Allows you to compile and load the current project into the debugger.

Make Statistics	Allows you to view a statistical profile of the most recent make. Information on the size of each module, including code and data size, will appear in the **Make Statistics** dialog.
Auto make before run	Toggles the Project System setting which forces a recompile each time you choose the **Run** command.
File save before run	Toggles the Project System setting which saves the source code file each time you choose the **Run** command.
Minimize on Run	Toggles the Project System setting which minimizes the Development Environment before displaying the application each time you choose the **Run** command.
Wait for termination on run	Toggles the Project System setting which suspends the development environment until after you terminate the application upon executing it with the **Run** command.
Generate	Generates *source code* for any procedures in the project that have *changed*.
Generate All	Generates source code for *all* procedures in the project.
Properties	Edits the current project file with the **Project Editor** dialog.

EDITING THE REDIRECTION FILE

Clarion's development environment sets the working directory to the one in which the current .APP or .PRJ file resides. Additionally, Clarion for Windows uses the redirection file (CW15.RED) to keep track of directories for the development environment's components. This file tells the development environment where to find files and where to create new files. Each line of the redirection file is in the format:

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

The *filepattern* can be a file name or a file pattern using the standard DOS wild card characters: * and ?.

For example:

```
*.dbd = c:\cw15\obj
*.dll = .;c:\cw15\bin
*.lib = c:\cw15\obj;c:\cw15\lib
*.res = c:\cw15\obj;c:\cw15\lib
*.obj = c:\cw15\obj;c:\cw15\lib
*.rsc = c:\cw15\obj
*.ico = .;c:\cw15\template
*.bmp = .;c:\cw15\template
*.tpl = c:\cw15\template
*.tpw = c:\cw15\template
*.trf = c:\cw15\template

*.* = .; c:\cw15\examples; c:\cw15\libsrc

QCKSTART.TXA = c:\cw15\TEMPLATE
QCKSTART.TXD = c:\cw15\TEMPLATE
```

The first *directory* is the directory in which any new file of the type specified by the *filepattern* is created. This is only true for files created and saved by the development environment, such as .OBJ, .DBD, .LIB, .EXE, and .CLW. The subsequent directories are paths where Clarion will search for existing files.

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

To edit the redirection file, choose **Setup ➤ Edit Redirection File**. The text editor opens the CW15.RED file for editing. File patterns appear on the left; directory paths on the right.

❏ *To change the default path for a file type*, select the current path by DOUBLE-CLICKING, then type over it with the new path.

❏ *To append an additional subdirectory onto the search path for a file pattern*, add a semicolon at the end of the current path, then add the subdirectory.

CREATING A PROJECT FILE FOR A HAND CODED APPLICATION

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 final executable for your application. It also sets the compiler options ranging from whether to include debug code or not, to setting a preferred optimization method.

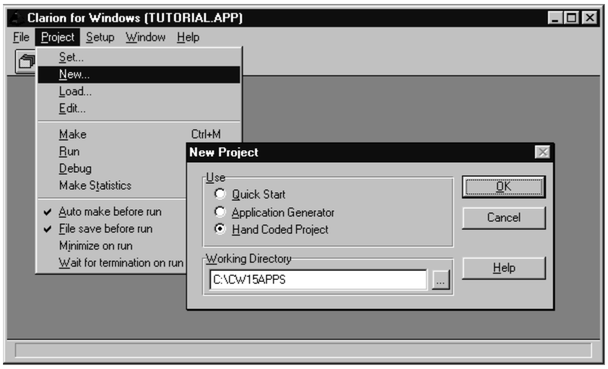
If you use the Application Generator to create your source code, no separate .PRJ file is created, and the only thing you will probably use the Project System for is to set debugging options. The Application Generator takes care of maintaining most everything else for you. Therefore, this chapter describes how to use the Project System for hand-coded applications.

The **Project Tree** dialog organizes all the components, and provides access to other dialogs that manage your project file.

❏ *To create a project file*, take these steps:

1. Choose **Project ➤ New**.

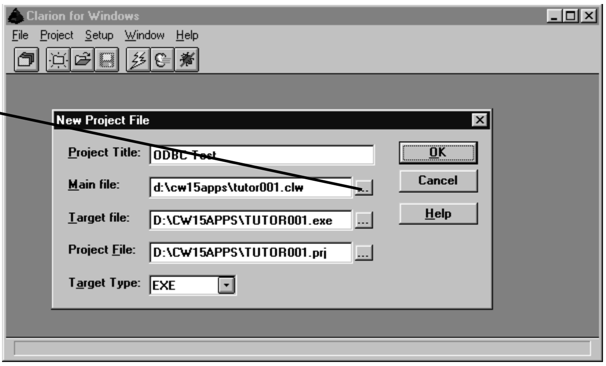
   The **New Project** dialog appears.



2. In the **Use** box, CLICK on **Hand Coded Project**.
3. In the **Working Directory** combo box, type a directory path, or press the ellipsis (...) button to choose a directory with the standard Open File dialog.
4. Press the **OK** button.

   The **New Project File** dialog appears.
5. In the **Project Title** field, type a descriptive name for your project.

Once you press the ellipsis button and select a Main File, the last three fields fill in automatically.



Project Title: ODBC Test
Main file: d:\cw15apps\tutor001.clw
Target file: D:\CW15APPS\TUTOR001.exe
Project File: D:\CW15APPS\TUTOR001.prj
Target Type: EXE

6. 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 standard Open File dialog.

   This step automatically fills in the **Target file** and **Project file** names as well. If you wish to set a different name for either, type new names, or press the corresponding ellipsis (...) button to choose a file with the standard Open File dialog.

7. Press the **OK** button.

   The **ProjectTree** dialog appears.

8. CLICK on **Database driver libraries**, then press the **Add File** button.

   The **Select Driver** dialog appears.

   The **Add File** button has a different effect, depending on which Project Tree folder is highlighted. In this case, it 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 a standard Open File dialog to help you locate the file to include in your project's compile and link process. If you choose the wrong *type* of file, the Project System adds the file to the appropriate Project Tree folder.

9. Select a database driver, and press the **OK** button.

10. CLICK on the **Project:** (first) line, and press the **Properties** button.

    The **Global Options** dialog appears. 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.

11. Press the **OK** button, then CLICK on Yes when asked if you want to save the project file.

MAINTAINING A PROJECT

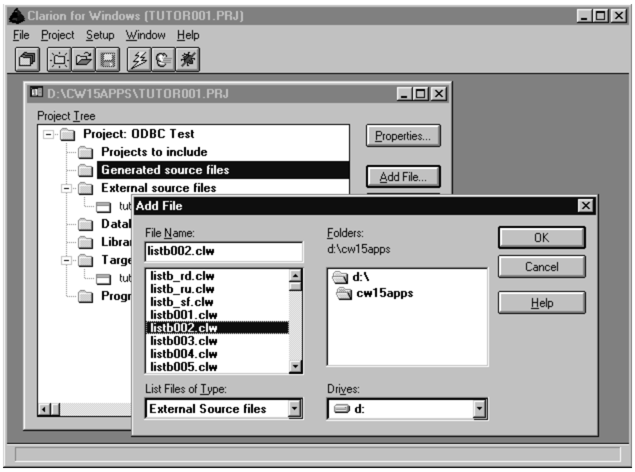
This section provides an overview of the steps necessary to maintain a project file. Maintaining the project file includes adding and removing source files, object files, or libraries from the compile and link process.

Adding 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.

❑ *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 standard **Open File** dialog.

If you choose the wrong *type* of file, the Project System adds the file to the appropriate Project Tree folder.



Generated source modules cannot be added or deleted from the Project System. They can only be added or deleted from the Application Generator.

Any attempt to add source modules to the **Generated source files** will add the source file to the **External source files**.

Adding Object Files and Libraries

An object (.OBJ) file is a file that contains an object (routines, functions, or procedures) that can be linked into or added to 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. You can create .LIB files for use with your Clarion applications. See *Setting the Target File* below.

Objects used in this manner are called external, because the source code for the objects is external to your project. That is, you don't need the object's source code to use the object. Nor do the objects need to be written in the same programming language. Your Clarion programs can call objects compiled from C, C++, Pascal, etc.

Your application may call on the TopSpeed database driver routines to access your TopSpeed files. These routines are in libraries supplied with Clarion for Windows, and are placed in the CW15\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.

❏ *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 you wish to add from the **Select Driver** dialog.

If you choose the wrong *type* of file, the Project System adds the file to the appropriate Project Tree folder.

❑ *To link another library or object file*, highlight **Library and object files** in the Project Tree list. Then press the **Add File** button and select the file you wish to add with the standard **Open File** dialog.

The .LIB or .OBJ 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.

Adding External Resources

One of the most likely things you'll wish to do with the Project System is to specify *resources* to link into the executable. These include graphics (.BMP, .ICO and .WMF files). By linking them into the executable, you can avoid having to ship them as separate, external files.

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 runtime property assignment statement, the linker will only include the new file in your executable if you add the file to your Project Tree.

❑ *To add graphic files to the executable*:

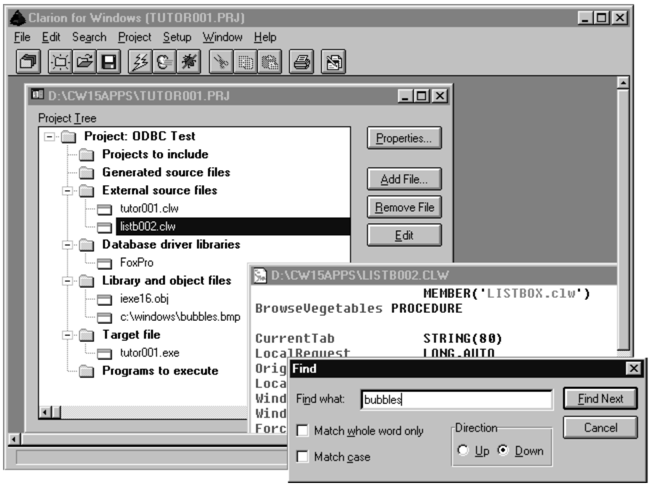
1. Highlight **Library and object files** and CLICK on the **Add File** button.

   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, and CLICK on the **Edit** button.

   The source code file is opened by the Text Editor.



Project: ODBC Test
- Projects to include
- Generated source files
- External source files
  - tutor001.clw
  - listb002.clw
- Database driver libraries
  - FoxPro
- Library and object files
  - iexe16.obj
  - c:\windows\bubbles.bmp
- Target file
  - tutor001.exe
- Programs to execute

```
                  MEMBER('LISTBOX.clw')
BrowseVegetables PROCEDURE

CurrentTab            STRING(80)
LocalRequest          LONG,AUTO
Orig
Loca
Wind
Wind
Forc
```

Find what: bubbles

Match whole word only

Match case

Direction: Up [✓ Down]

4. Place a tilde (~) in front of the graphic file name in the source code assignment statement (not in 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.

Adding Other Projects

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 allows you to 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.

Adding Programs to Execute

Programs to execute allows you to customize the compile and link process by executing the program(s) of your choice. 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 standard **Open File** dialog.

DISTRIBUTING FILES

This section is included to help you decide what kind of target file to specify for your project. See *Setting the Target File* below.

Clarion for Windows produces true 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 xx is “16” or “32” depending on whether the application is targeted to run on a 16-bit operating system (Windows 3.10, 3.11) or a 32-bit operating system (Windows 95 or Windows NT):

◆ *.EXE

A stand alone .EXE will usually be larger than an .EXE distributed with .DLL(s). However, the stand alone .EXE will probably be smaller than the combined sizes of an .EXE and its associated .DLL(s).

The stand alone .EXE is made as small as possible by Clarion’s smart linking process that only links in functions actually called by the application program (whereas the .DLL contains a fixed set of functions, whether or not they are actually called by your program).

A stand alone .EXE cannot have conflicts or problems that arise from linking with the wrong .DLL(s) at run time.

Make (compile and link) time for a stand alone .EXE is greater than for an .EXE combined with a .DLL.

- *.EXE + CWRUNxx.DLL

  Splitting the executables between .EXEs and .DLLs allows for more efficient use of disk space. Many Clarion applications (.EXEs) can share a single CWRUNxx.DLL. Or, a single application suite with several .EXEs can share a single CWRUNxx.DLL. However, as a developer, you must ensure that your application accesses the correct version of CWRUNxx.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.

Splitting executables between .EXEs and .DLLs allows for more efficient use of disk space, but less efficient use of RAM. This is because Windows loads an additional CWRUNxx.DLL into memory for each active Clarion for Windows executable, and because the CWRUNxx.DLL contains some functions your .EXE will never call.

- *.EXE + CWRUNxx.DLL + *.DLL$_1$ + ... + *.DLL$_n$

  This configuration offers the same advantages and disadvantages as the .EXE + CWRUNxx.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 CWRUNxx.DLL. See the *Database Drivers Appendix* for more information on database drivers and their files.

- *.EXE + *.DLL$_1$ + ... + *.DLL$_n$

  This configuration offers most of the same advantages and disadvantages as the .EXE + CWRUNxx.DLL configuration. It is listed here to illustrate that the CWRUNxx.DLL may be linked into another .DLL. This technique “hides” the CWRUNxx.DLL and ensures that your application will never get the wrong version of CWRUNxx.DLL, because, technically, it isn’t looking for CWRUNxx.DLL.

If CWRUNxx.DLL is distributed, it must reside in the same directory as the application, in the Windows\System subdirectory, or in any directory referenced in the DOS PATH. TopSpeed recommends that you install CWRUNxx.DLL to the application directory when you create a setup program for distributing your applications.

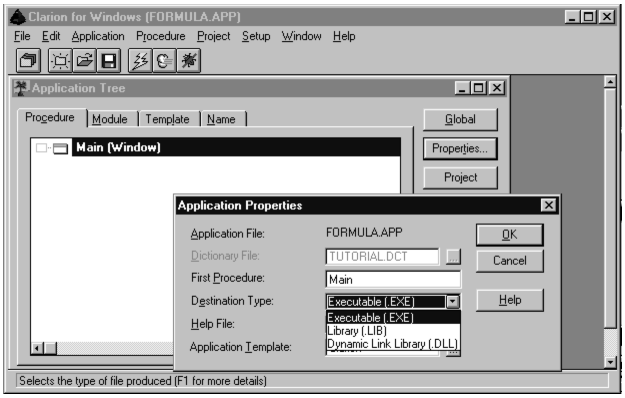
Remember, multiple Clarion for Windows applications may use the same CWRUNxx.DLL file, avoiding the need to duplicate space on the users' hard drive. On the other hand, using only one CWRUNxx.DLL raises the possibility of conflicts among applications developed under different versions of Clarion for Windows. To avoid possible conflicts, install a separate CWRUNxx.DLL to each application directory, or distribute the application as a single *.EXE file, or link the CWRUNxx.DLL into another .DLL that is unique to your application.

SETTING THE TARGET FILE

Using the Project System, you may easily specify creation of .EXE, .LIB, or .DLL files. This section provides you with information on creating these target file types, as well as brief explanations of the file types. See *Adding Object Files and Libraries* and *Distributing Files* above for more information.

❏ The project system assumes by default that you wish to create a standard executable (.EXE) file. When you name the project file in the **New Project File** dialog, it automatically sets the target file extension to .EXE.

❏ *To create a library (.LIB file*), simply change the target file extension in the Project Tree. Highlight **Target file** in the Project Tree and 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.

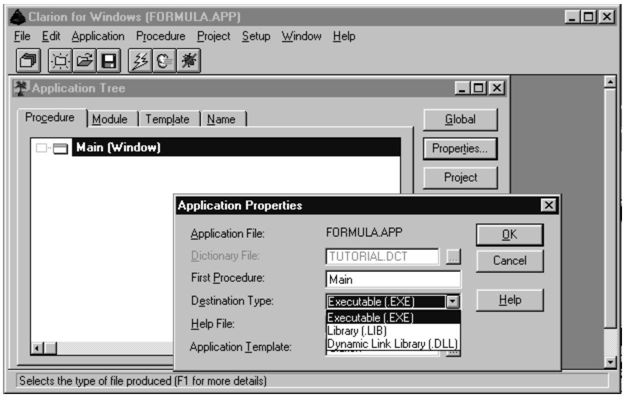

Clarion for Windows (FORMULA.APP)
File Edit Application Procedure Project Setup Window Help
Application Tree
Procedure Module Template Name
Global
Properties...
Project
Main (Window)
Application Properties
OK
Cancel
Help

Application File: FORMULA.APP
Dictionary File: TUTORIAL.DCT
First Procedure: Main
Destination Type: Executable (.EXE)
Help File:
Application Template:

Executable (.EXE)
Library (.LIB)
Dynamic Link Library (.DLL)

Selects the type of file produced (F1 for more details)

❏ *To create a dynamic link library (.DLL)*, change the target extension in the Project Tree. Highlight **Target file** in the Project Tree and 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.


Clarion for Windows (FORMULA.APP)
File Edit Application Procedure Project Setup Window Help
Application Tree
Procedure Module Template Name
Main (Window)
Global
Properties...
Project
Application Properties
OK
Cancel
Help

Application File: FORMULA.APP
Dictionary File: TUTORIAL.DCT
First Procedure: Main
Destination Type: Executable (.EXE)
Help File:
Application Template:

Executable (.EXE)
Library (.LIB)
Dynamic Link Library (.DLL)

Selects the type of file produced (F1 for more details)

.LIB Files

Library files contain (external) procedures and functions which are linked to your application at *compile* time. To create library files which may be accessed by Clarion for Windows, 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 external 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. See the *Language Reference*.

2. Add the function or procedure prototypes:

```
MAP
  MODULE('EXTERNAL.LIB')
    ExtProc(*CSTRING),RAW            !procedure prototype
    ExtFunc(USHORT, *BYTE[]),USHORT  !function prototype
  END
END
```

   Each prototype specifies the name of the procedure or function, the data types of any parameters (in parentheses), and the return data type (if a function). See the *Language Reference*.

   In the example above, the procedure (here named ExtProc) expects the address (without the length, hence the RAW attribute) of a CSTRING to be passed to it as a parameter.

The function (here named ExtFunc) 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 the procedure as expecting the C calling convention. The second line identifies the function as 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')
```

.DLL Files

Dynamic Link Libraries contain external procedures and functions which are linked to your application only at runtime. To create dynamic link libraries, just specify .DLL as the target file extension.

To call an external .DLL procedure follow the steps outlined for calling a .LIB procedure, above.

SETTING PROJECT FILE OPTIONS

This section describes the individual components of the project file, and shows you how to modify their properties.

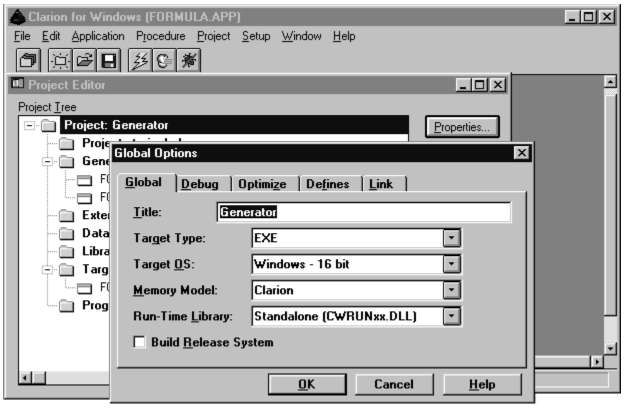
Global Compile and Link Options

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.


Clarion for Windows (FORMULA.APP)
File Edit Application Procedure Project Setup Window Help
Project Editor
Project Tree
Project: Generator
Properties...
Global Options
Global Debug Optimize Defines Link
OK Cancel Help

Title: Generator
Target Type: EXE
Target OS: Windows - 16 bit
Memory Model: Clarion
Run-Time Library: Standalone (CWRUNxx.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 — Not implemented in this release, accept the default.

Run-Time Library — *To specify how the runtime library is called by the target file,* choose ***Standalone***, ***Local***, or ***External*** from the **Run-Time Library** drop list.

- **Standalone** — Creates the target file so it calls the runtime libraries as CWRUNxx.DLL.
- **Local** — Creates the target file with the runtime library linked internally (a "one-piece" executable).

External Links the application so it calls the runtime library from a .DLL which you have created with the runtime library linked internally and exported.

Build Release System

To create an executable for release, check the **Build Release System** box. *To create an executable for use with the Debugger*, uncheck the **Build Release System** box.

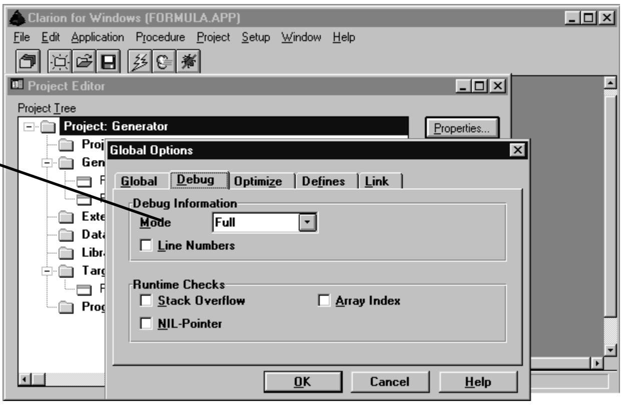
Debug Tab

Debug Mode *To specify the level of debug capability*, choose ***Off***, ***Min***, or ***Full*** from the ***Mode*** drop list.

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 runtime*, check the **Stack Overflow** box.

Setting global debug options.


Clarion for Windows (FORMULA.APP)
File Edit Application Procedure Project Setup Window Help
Project Editor
Project Tree
Project: Generator
Properties...
Global Options
Global Debug Optimize Defines Link
OK Cancel Help

Debug Information
Mode: Full
☐ Line Numbers

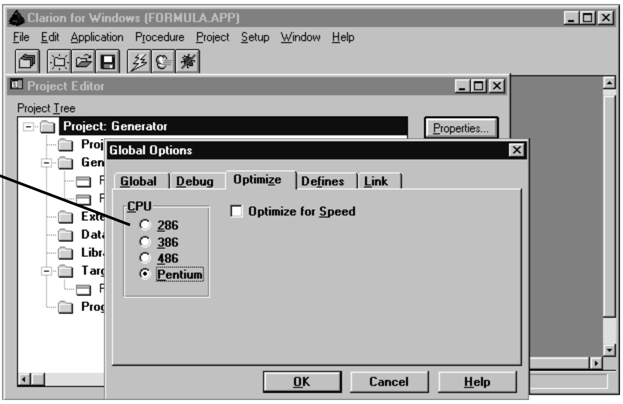
Runtime Checks
☐ Stack Overflow ☐ Array Index
☐ NIL-Pointer

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 runtime*, check the **Array Index** box.

Optimize Tab

Setting global optimizer options.


Clarion for Windows (FORMULA.APP)
File Edit Application Procedure Project Setup Window Help
Project Editor
Project Tree
Project: Generator
Properties...
Global Options
Global Debug Optimize Defines Link
CPU
286
386
486
Pentium
Optimize for Speed
OK
Cancel
Help

CPU: ( ) 286 ( ) 386 ( ) 486 [✓ Pentium]

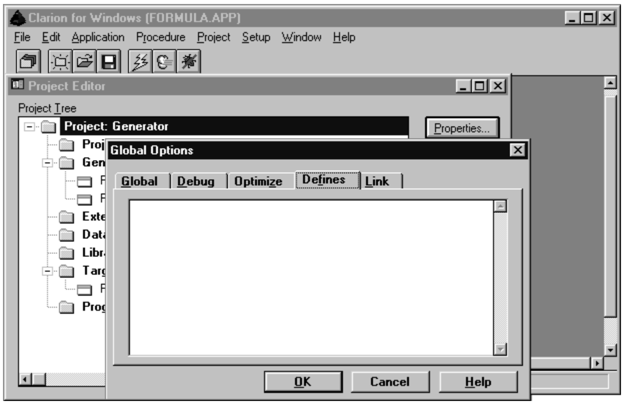
☐ Optimize for Speed

CPU *To specify optimization by microprocessor type*, choose from **286**, **386**, **486,** or **Pentium**.

Optimize for Speed

To favor program speed over creating a smaller executable file, check the **Optimize for Speed** box.

Defines Tab


Clarion for Windows (FORMULA.APP)
File Edit Application Procedure Project Setup Window Help
Project Editor
Project Tree
Project: Generator
Properties...
Global Options
Global Debug Optimize Defines Link
OK
Cancel
Help

Defines *To define a switch, or switches for use with the* COMPILE *and* OMIT *compiler directives*, type a list of valid Clarion labels separated by commas. Each label defines a separate switch.

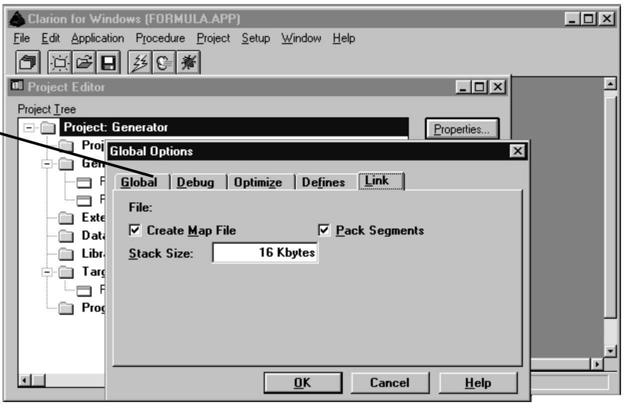
Defines refers to the Project System language statement #PRAGMA DEFINE(). The #PRAGMA DEFINE() statement creates a switch that can be toggled on and off. The switch can then be referenced by the COMPILE and OMIT compiler directives. See the *Language Reference* for more information on COMPILE and OMIT.

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
```

Link Tab

Setting global link optoins.


Clarion for Windows (FORMULA.APP)
File Edit Application Procedure Project Setup Window Help
Project Editor
Project Tree
Project: Generator
Properties...
Global Options
Global Debug Optimize Defines Link
OK Cancel Help

File:
☑ Create Map File ☑ Pack Segments
Stack Size: 16 Kbytes

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.

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.

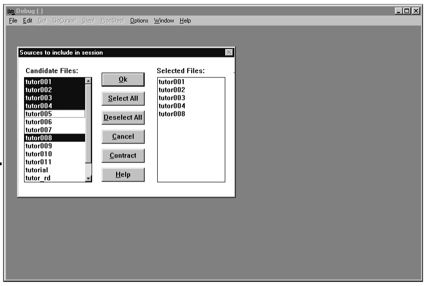
Highlight a source code file in the Project Tree dialog, then press the **Properties** button. The **Compile Options** dialog appears, showing most of the same controls as the **Global Options** dialog. This dialog sets compile options for the individual source module highlighted.

See *Debug Tab*, *Optimize Tab*, and *Defines Tab* above for information on using this dialog.

USING THE DEBUGGER

Contents

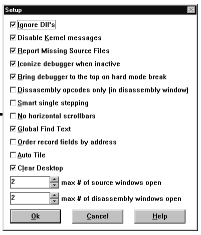
Debugging a program usually requires running the program and repeatedly stopping it to examine the value of different variables. The Clarion Debugger provides all the tools necessary for tracking down your application's bugs.


Debug ( )
File Edit Go! GoCursor! Step! ProcStep! Options Window Help
Sources to include in session
Ok
Select All
Deselect All
Cancel
Contract
Help

Candidate Files:
tutor001
tutor002
tutor003
tutor004
tutor005
tutor006
tutor007
tutor008
tutor009
tutor010
tutor011
tutorial
tutor_rd

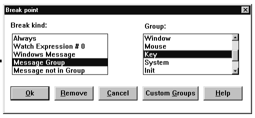
Selected Files:
tutor001
tutor002
tutor003
tutor004
tutor008

Start the Debugger from within the Development Environment, or from Windows. See your source code in one of the Debugger windows as it executes.



- ☑ Ignore Dll's
- ☑ Disable Kernel messages
- ☑ Report Missing Source Files
- ☑ Iconize debugger when inactive
- ☑ Bring debugger to the top on hard mode break
- ☐ Dissasembly 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

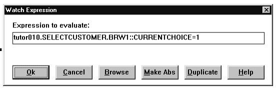
Customize the Debugger to fit your work environment.



Break kind:
Always
Watch Expression # 0
Windows Message
Message Group
Message not in Group

Group:
Window
Mouse
Key
System
Init

Once you suspect the cause of the bug, set a breakpoint. You can then execute the program, and automatically stop it at that point to examine and change variable values.



Expression to evaluate:
tutor010.SELECTCUSTOMER.BRW1::CURRENTCHOICE=1

Set breakpoints on simple or complex expressions.

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 will:

- Tell you how to prepare your projects for debugging.
- Tell you how to start the debugger.
- Tell you how to customize the debugger's operation to your work environment.
- Tell you how to monitor your program's execution and check its state at specific points by setting break points and watch expressions.

OVERVIEW: 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 about the Setup 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: Clarion Soft Mode is recommended 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.

4. Set a break point.
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 break point. 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 break points, 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 debugger. Exiting the debugger while the debuggee is still active can cause system crashes.
11. Exit the debugger.

PREPARING YOUR PROJECTS FOR DEBUGGING

The Project System allows you to set the debug options for *all* the programs in your application in the **Global Options** dialog. To make your executable (.EXE or .DLL) suitable for debugging:

1. Create your project file, and make it the current project (the *Using 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 holds the name of the project, and press the **Properties** button.
4. When the **Global Options** dialog appears, select the **Debug** tab, then choose **Full** from the **Debug Mode** drop list.
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.

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, and 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.

THE 16-BIT DEBUGGER

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 will be available for your application and the debugger.

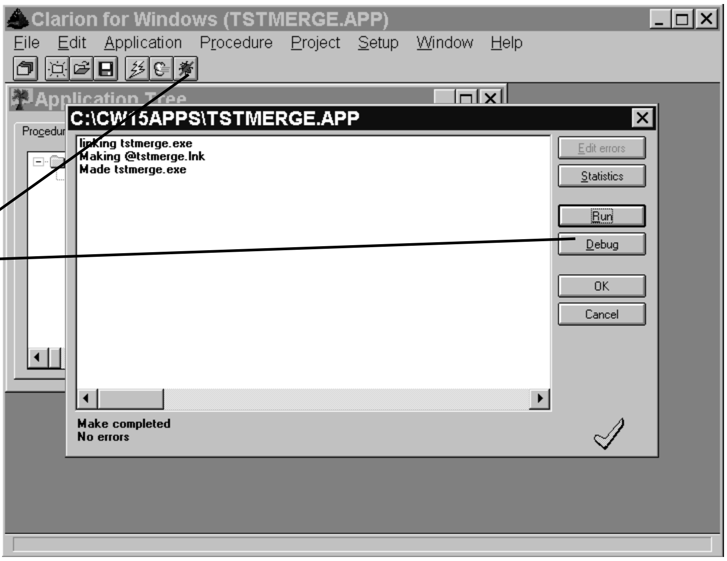
❑ To start the debugger from the development environment, *either*:

1. Choose **Project ➤ Debug** *or* press the *Debug* 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 *Make* button, then, with the compile results dialog still open, press the **Debug** button.


Two ways to start the debugger.
Clarion for Windows (TSTMERGE.APP)
File Edit Application Procedure Project Setup Window Help
Application Tree
C:\CW15APPS\TSTMERGE.APP
Edit errors
Statistics
Run
Debug
OK
Cancel

```
linking tstmerge.exe
Making @tstmerge.lnk
Made tstmerge.exe
```

```
Make completed
No errors
```

❑ To start the debugger from the windows environment:

1. Switch to Program Manager (Windows 3.x), or press the Start button on the taskbar (Windows 95) and open the Clarion program group.
2. DOUBLE-CLICK the Clarion Debugger 16-bit icon (Windows 3.x), or choose Clarion Debugger 16-bit from the programs menu (Windows 95).

   If you wish to start the debugger from another program launcher, the application path name is C:\CW15\BIN\CLWDB.EXE.
3. With the debugger launched, choose **File ➤ File to Debug**, then choose an .EXE file in the **Open 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, and choose **File ➤ File to Debug**. Choose the .EXE file for the running program from the **Open File** dialog. The debugger will ask 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 appears:

1. Select the source code files in the **Sources to include in session** dialog by CLICKING on them.

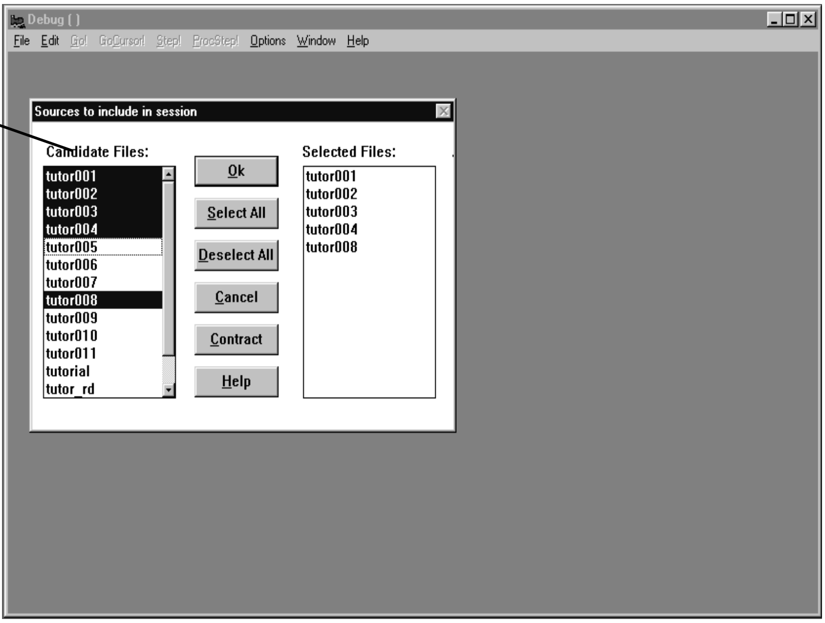
   The debugger stores the files you select between debug sessions.

2. Press the **Select All** button to include all project source files.
3. Press the **Expand/Contract** button for a list of only those source files selected.
4. Press the **OK** button.

   The debugger windows appear.

If the application does not include debug information, the debugger skips this step and opens a disassembly window (explained below).

The Debugger prompts you to select the source code files for the project.



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 in which you debug your project.

Soft Mode	Toggles hard and soft mode debugging. In *soft mode*, when the program being debugged is suspended in the debugger, part of the debugger will attempt 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 will be 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 runtime library to simulate the behavior of the program being debugged. *This is the recommended mode for most projects.*
Extended StackTrace	Debugger shows information about procedures when no debug information is available. A disassembly window opens, containing the relevant segment.
Disassembly On	The Disassembly Window “shadows” the active source window. When you select a line of source, the cursor in the Disassembly Window moves to the line corresponding to it.

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 **Break Point** dialog.

Assembly Single Step Toggles step mode for assembler break points. When execution reaches an assembler break point, step mode is set on. When execution reaches a source break point, it turns off.

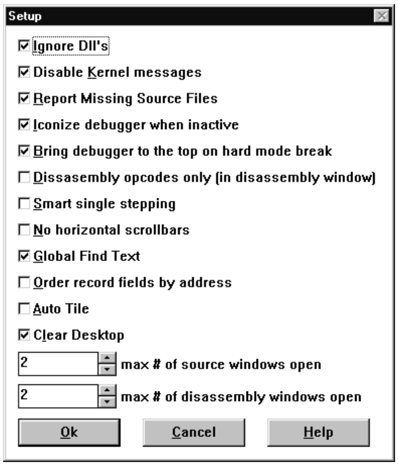
Control Panel Displays a toolbox window with buttons corresponding to the four **Go!** commands. The next time the program being debugged is suspended, the control panel receives focus.

Tip: When debugging in hard mode, when you activate the main Debugger window, you cannot access the control panel.

Setup Opens the **Setup** dialog. See below.

Debugger Setup Options

Access the debugger **Setup** dialog using the **Options ➤ Setup** command. The dialog provides the following options.


Setup

[✓] Ignore Dll's
[✓] Disable Kernel messages
[✓] Report Missing Source Files
[✓] Iconize debugger when inactive
[✓] Bring debugger to the top on hard mode break
[ ] Dissasembly 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

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 utilized 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.

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.

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 CW15.RED with the debugger, choose **File ➤ Load Redirection**. The Redirection file helps the debugger locate files such as *.DBD, and *.CLW. See the *Using the Project System* chapter, *Editing the Redirection File*.

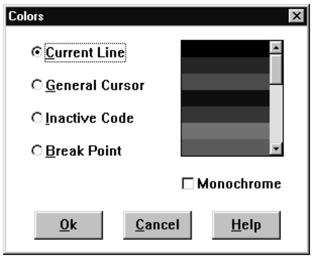
Active DLL's To add a related dynamic link library (*.DLL) to the debug session, choose **File ➤ Debug Active DLL**. Choose a file from the **Active Module** dialog. This option is available for hard mode debugging only.

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 command line 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 break points from a previously terminated session, choose **File ➤ Restart**. This option is available, *provided* the source code has not changed.
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, and choose the Windows messages from the list box in the **Selective Break Point 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 **Break Points** in the **Color** dialog.



The Debugger Windows

The debugger consists of a collection of child windows, which track different information about the program for you. These windows are:

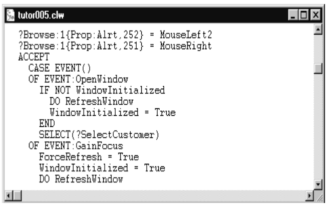
- The **source code** window
- The **Watch Expressions** window
- The **Global Variables** window
- The **Active Procedures** window
- The **Disassembly** window
- The **Machine Registers** window
- The **Library States** window

- 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.

Default Windows

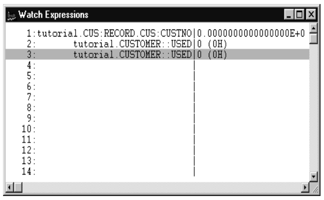
At first, the debugger opens four windows:

- 

```
?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
```

  The **source code** windows display the source code documents. 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.

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. Check for .DBD files, in C:\CW15\OBJ.

- 

```
 1:tutorial.CUS:RECORD.CUS:CUSTNO|0.0000000000000000E+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 **Watch Expressions** window shows the current value of variables *and* expressions. Set a watch expression to see how a variable or expression changes as your program executes.

  The title bar is **Watch Expressions**. See the *Editing Watch Expressions* section (below) to learn the syntax for watch expressions. To add a variable to the watch list:

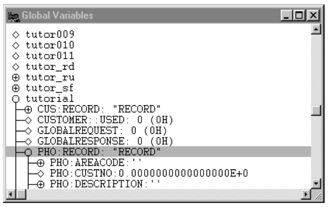
1. DOUBLE-CLICK on an empty line in the **Watch Expressions** window.
2. When the **Watch Expression** dialog appears, type a variable name (as it appears in the global variables list) and press **OK**.
3. Alternatively, press the **Browse** button, select a variable from the list, then press **OK** 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 the *Editing Watch Expressions* section, below.

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.

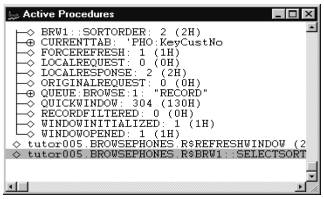
◆ 

```
◇ tutor009
◇ tutor010
◇ tutor011
◇ tutor_rd
⊕ tutor_ru
⊕ tutor_sf
○ tutorial
├⊕ CUS:RECORD: "RECORD"
├◇ CUSTOMER::USED: 0 (0H)
├◇ GLOBALREQUEST: 0 (0H)
├◇ GLOBALRESPONSE: 0 (0H)
├○ PHO:RECORD: "RECORD"
│ ├⊕ PHO:AREACODE:''
│ ├◇ PHO:CUSTNO:0.0000000000000000E+0
│ ├⊕ PHO:DESCRIPTION:''
```

The **Global Variables** window shows you 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.

The **Global Variables** window contains 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 top level is the source code module which contains the variable. The next level is the variable name.

◆ 

```
├◇ BRW1::SORTORDER: 2 (2H)
├⊕ CURRENTTAB: 'PHO:KeyCustNo
├◇ FORCEREFRESH: 1 (1H)
├◇ LOCALREQUEST: 0 (0H)
├◇ LOCALRESPONSE: 2 (2H)
├◇ ORIGINALREQUEST: 0 (0H)
├⊕ QUEUE:BROWSE:1: "RECORD"
├◇ QUICKWINDOW: 304 (130H)
├◇ RECORDFILTERED: 0 (0H)
├◇ WINDOWINITIALIZED: 1 (1H)
└◇ WINDOWOPENED: 1 (1H)
◇ tutor005.BROWSEPHONES.R$REFRESHWINDOW (2
◇ tutor005.BROWSEPHONES.R$BRW1::SELECTSORT
```

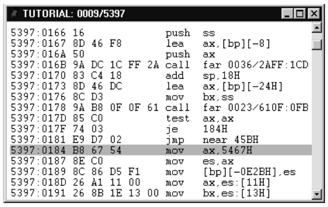
The **Active Procedures** window lists the procedure currently executing, which allows you to monitor nested procedure calls. The window appears in tree format. The upper levels represent the names of procedures and the lower levels represent the variables.

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.

Other Windows

Other debug windows provide other types of information:

◆ 

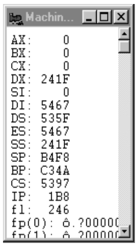
```
TUTORIAL: 0009/5397
5397:0166 16             push  ss
5397:0167 8D 46 F8       lea   ax,[bp][-8]
5397:016A 50             push  ax
5397:016B 9A DC 1C FF 2A call  far 0036/2AFF:1CD
5397:0170 83 C4 18       add   sp,18H
5397:0173 8D 46 DC       lea   ax,[bp][-24H]
5397:0176 8C D3          mov   bx,ss
5397:0178 9A B8 0F 0F 61 call  far 0023/610F:0FB
5397:017D 85 C0          test  ax,ax
5397:017F 74 03          je    184H
5397:0181 E9 D7 02       jmp   near 45BH
5397:0184 B8 67 54       mov   ax,5467H
5397:0187 8E C0          mov   es,ax
5397:0189 8C 86 D5 F1    mov   [bp][-0E2BH],es
5397:018D 26 A1 11 00    mov   ax,es:[11H]
5397:0191 26 8B 1E 13 00 mov   bx,es:[13H]
```

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.

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.

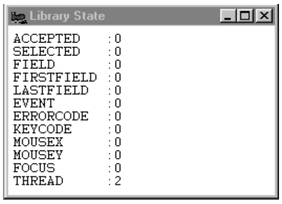
INS inserts an unconditional break point at the cursor. DEL removes one. Pressing the SPACE BAR displays the **Break Point** dialog.

◆ 

```
Machin...
AX:     0
BX:     0
CX:     0
DX:  241F
SI:     0
DI:  5467
DS:  535F
ES:  5467
SS:  241F
SP:  B4F8
BP:  C34A
CS:  5397
IP:   1B8
fl:   246
fp(0): ô.?00000
fp(1): ô.?00000
```

The **Machine Registers** window shows the current register values; choose **Window ➤ Registers** to display it.

The **Machine Registers** window shows the register in the left column, and its value to the right.

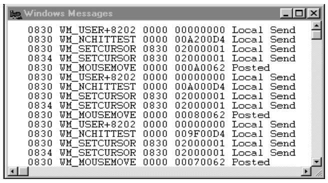
◆ 

```
Library State
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 **Library States** window displays return values for 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.

The names listed in EQUATES.CLW and KEYCODES.CLW appear next to the return values.

◆ 

```
Windows Messages
  0830 WM_USER+8202 0000 00000000 Local Send
  0830 WM_NCHITTEST 0000 00A200D4 Local Send
  0830 WM_SETCURSOR 0830 02000001 Local Send
  0834 WM_SETCURSOR 0830 02000001 Local Send
  0830 WM_MOUSEMOVE 0000 000A0062 Posted
  0830 WM_USER+8202 0000 00000000 Local Send
  0830 WM_NCHITTEST 0000 00A000D4 Local Send
  0830 WM_SETCURSOR 0830 02000001 Local Send
  0834 WM_SETCURSOR 0830 02000001 Local Send
  0830 WM_MOUSEMOVE 0000 00080062 Posted
  0830 WM_USER+8202 0000 00000000 Local Send
  0830 WM_NCHITTEST 0000 009F00D4 Local Send
  0830 WM_SETCURSOR 0830 02000001 Local Send
  0834 WM_SETCURSOR 0830 02000001 Local Send
  0830 WM_MOUSEMOVE 0000 00070062 Posted
```

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 break point 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 via 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.

Tip: If you include DDE services in your application, we recommend testing your application with another DDE application and monitoring the DDE messages. For further information, see the *Microsoft Windows 3.1 Programmers Reference, Volume 3*, available from Microsoft Press.

Setting Break Points

Normally, when debugging an application, you'll 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.

Break points 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 break point, 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.

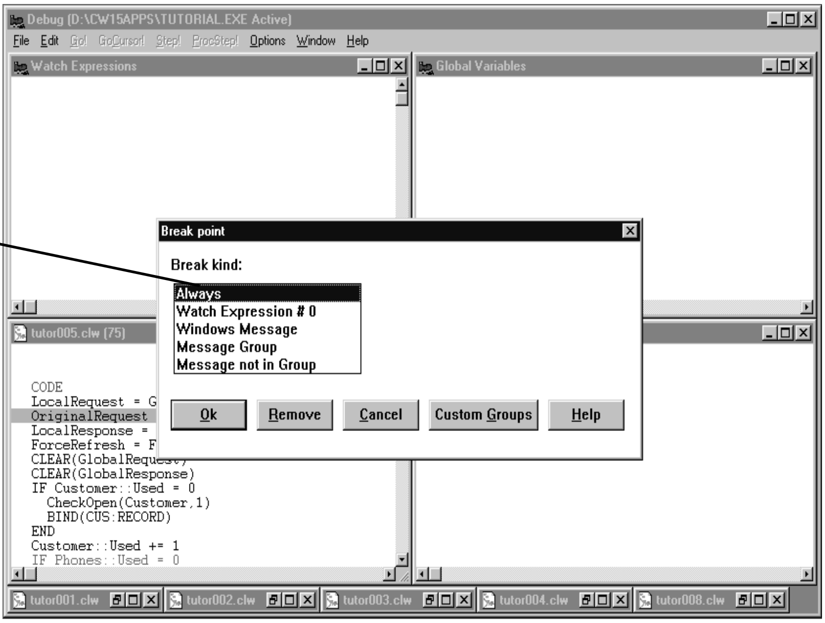
You can also set conditions on the break point, telling the program to continue executing if the condition is false, or turning control over to the debugger if true.

When you set a break point, the source code line where the break point occurs appears in magenta in the source window.

Unconditional Break Points

An unconditional or "sticky" break point is placed on a source code line, and stops execution whenever the program encounters that statement:

Setting a break point. The debugger will always stop at this break point.



```
CODE
LocalRequest = G
OriginalRequest
LocalResponse =
ForceRefresh = F
CLEAR(GlobalRequest)
CLEAR(GlobalResponse)
IF Customer::Used = 0
  CheckOpen(Customer,1)
  BIND(CUS:RECORD)
END
Customer::Used += 1
IF Phones::Used = 0
```

1. Open the source code or disassembly window.
2. Locate the line of code to break on and DOUBLE-CLICK on it.

   The **Break Point** dialog appears.
3. Select **Always** and press the **OK** button.

When you execute the **Go!** command, the program will run until it reaches the break point, then stop.

Tip: When a source code or disassembly window is the active window, press INSERT to add an unconditional break point, or DELETE to remove one.

Conditional Break Points

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 message or messages from Windows to the application.

❑ To set a conditional break point on a change in a watch expression's value:

1. Establish a watch expression as described in the *Editing Watch Expressions* section, below.
2. Locate the line of code to break on and DOUBLE-CLICK on it.

3. When the **Break Point** dialog appears, select **Watch Expression #0**.
4. Type the number of the watch expression (from the watch expressions window) in the **Watch#0** field.
5. Press the **OK** button.

   When you execute the **Go!** command, the program runs until it reaches the break point, 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.

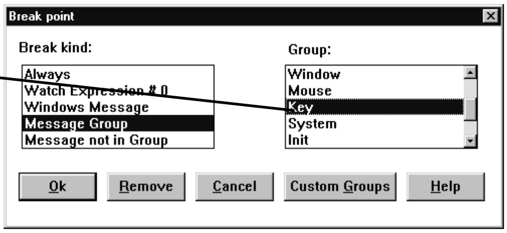
❑ To set a break point conditional on a specific Windows message:

1. Locate a line of code to break on and DOUBLE-CLICK on it.
2. When the **Break Point** 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 break point 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 break point condition would be true.

❑ To set a break point conditional (or not) on receipt of one of several Windows messages:

1. Locate the line of code to break on and DOUBLE-CLICK on it.
2. When the **Break Point** 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 break point will occur only on one of these messages.
4. Press the **OK** button.

   For example, you could place a break point 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 break point condition would be true.

❑ To set a break point which breaks when you receive an unexpected message, that is, one that doesn't belong to a group you specify:

1. Locate the line of code at which you want to establish the break point 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.

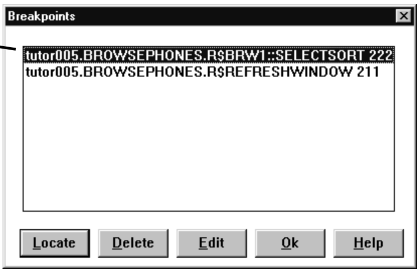
Tip: DOUBLE-CLICKING on a source code line containing a call to a procedure takes you to the first line of that procedure. esc returns you.

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.
Break Points	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	Calls the **Breakpoint** dialog, See *Setting Breakpoints* above.

The Breakpoints list.

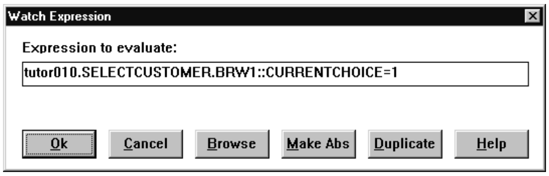


```
tutor005.BROWSEPHONES.R$BRW1::SELECTSORT 222
tutor005.BROWSEPHONES.R$REFRESHWINDOW 211
```

Editing Watch Expressions

The debugger contains an expression editor dialog, which allows you to edit a watch expression. Sometimes you want to take an action depending on the value of an expression that uses variables from your program. For example, you may want to stop the application and look at a variable if it's a negative value, or continue on to the next break point if it's positive.

To edit a watch expression, select a line in the **Watch Expressions** dialog and choose **Edit ➤ Edit**. The **Watch Expression** dialog appears.



Expression to evaluate: tutor010.SELECTCUSTOMER.BRW1::CURRENTCHOICE=1

Type an expression in the **Expression to Evaluate** field, then press the **OK** button. When the debugger runs the program, it will test the expression upon reaching the breakpoint, and halt if the expression evaluates true.

The **Edit Expression** dialog also contains a **Browse** button, to help create your expression quickly and accurately. Press the **Browse** 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.

The **Duplicate** button creates a duplicate watch expression that appears in the **Watch Expressions** dialog.

You can prefix the variable name with a procedure name and/or a module name. This allows you to 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 will guess the right 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`

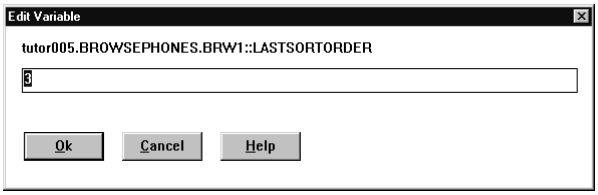
Editing Variables at Run Time

Using the debugger, you can change the value contained in a memory 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.
2. Press F2, or choose **Edit ➤ Edit**.

   The **Edit Variable** dialog appears.



tutor005.BROWSEPHONES.BRW1::LASTSORTORDER

3

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 memory variable changed to the new value.

THE 32-BIT DEBUGGER

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 will be available for your application and the debugger. The 32-bit debugger can debug multiple programs at the same time.

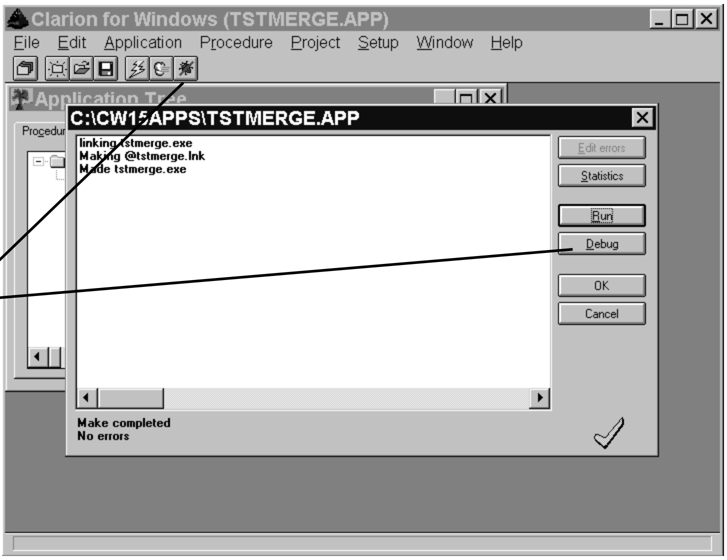
❏ To start the debugger from the development environment, *either*:

1. Choose **Project ➤ Debug** *or* press the *Debug* 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 *Make* button, then, with the compile results dialog still open, press the **Debug** button.


Two ways to start the debugger.
Clarion for Windows (TSTMERGE.APP)
File Edit Application Procedure Project Setup Window Help
Application Tree
C:\CW15APPS\TSTMERGE.APP
Edit errors
Statistics
Run
Debug
OK
Cancel

linking tstmerge.exe
Making @tstmerge.lnk
Made tstmerge.exe

Make completed
No errors

❏ To start the debugger from the windows environment:

1. Switch to Program Manager (Windows 3.x), or press the Start button on the taskbar (Windows 95) and open the Clarion program group.

2. DOUBLE-CLICK the Clarion Debugger 16-bit icon (Windows 3.x), or choose Clarion Debugger 32-bit from the programs menu (Windows 95).

   If you wish to start the debugger from another program launcher the application path name is C:\CW15\BIN\CWDB32.EXE.

3. With the debugger launched, choose **File ➤ File to Debug**, then choose an .EXE file in the **Open 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 display,

❏ To specify additional source files:

1. Choose **Window ➤ Source.**

   The **Select Source** dialog appears.

2. Highlight a source file and press the **OK** button.

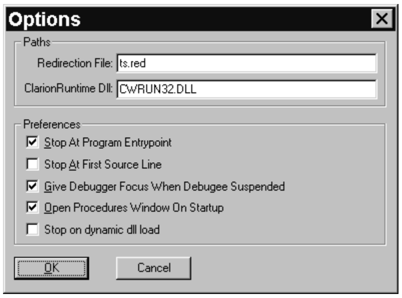
Repeat for each source file you want to debug.

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:


Options

Paths

Redirection File: ts.red

ClarionRuntime Dll: CWRUN32.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

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 the *Using the Project System* chapter, *Editing the Redirection File*.

Clarion Runtime Dll
Specifies the Clarion dynamic link library (DLL) linked into the .EXE being debugged.

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 **Open File** dialog.

Checking this option allows you to 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 **Open 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 *The Debugger Windows* below.

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.

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.

The Debugger Windows

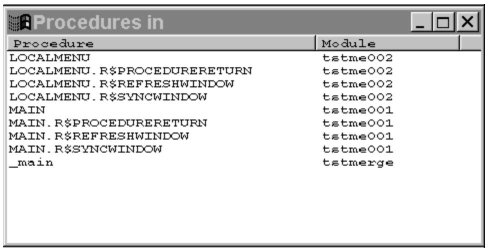
The debugger consists of a collection of child windows which track different information about the debuggee program for you. These windows are:

- The **Procedures In** window
- The **Globals** window
- The **Stack Trace** 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.

At first, the debugger 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

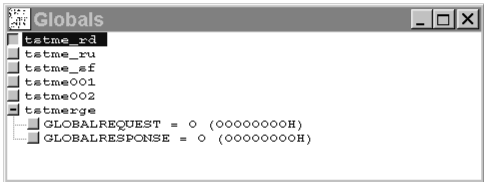


```
Procedure                              Module
LOCALMENU                              tstme002
LOCALMENU.R$PROCEDURERETURN            tstme002
LOCALMENU.R$REFRESHWINDOW              tstme002
LOCALMENU.R$SYNCWINDOW                 tstme002
MAIN                                   tstme001
MAIN.R$PROCEDURERETURN                 tstme001
MAIN.R$REFRESHWINDOW                   tstme001
MAIN.R$SYNCWINDOW                      tstme001
_main                                  tstmerge
```

Lists the procedures in the debuggee and their associated source modules. CLICK on a procedure name to display its associated source or assembler code.

Tip: Use the Procedures In window to navigate through your source code.

The Globals window



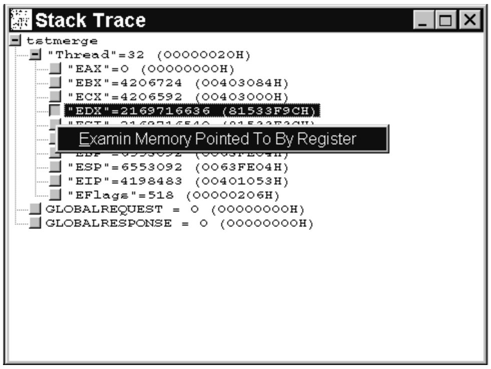
```
tstme_rd
tstme_ru
tstme_sf
tstme001
tstme002
tstmerge
   GLOBALREQUEST = 0 (00000000H)
   GLOBALRESPONSE = 0 (00000000H)
```

Displays the current value of each component of each global variable, as well as the library state. RIGHT-CLICK on a variable to change its value.

The **Globals** window contains expandable tree controls, so that 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



```
tstmerge
  "Thread"=32 (00000020H)
    "EAX"=0 (00000000H)
    "EBX"=4206724 (00403084H)
    "ECX"=4206592 (00403000H)
    "EDX"=2169716636 (81533F9CH)
    "ESP"=6553092 (0063FE04H)
    "EIP"=4198483 (00401053H)
    "EFlags"=518 (00000206H)
  GLOBALREQUEST = 0 (00000000H)
  GLOBALRESPONSE = 0 (00000000H)
```

Shows the current register values 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.

The **Stack Trace** window contains expandable tree controls, so that 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.

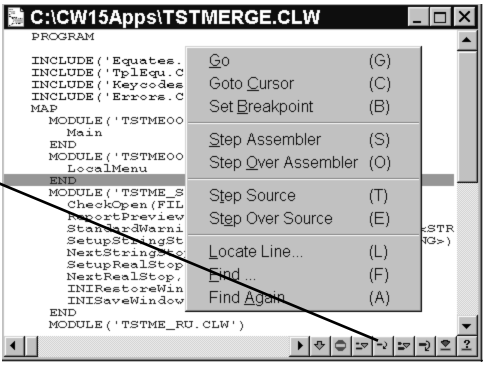
Tip: The Stack Trace window has the following special functionality:

RIGHT-CLICK on a variable to change 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 Source window



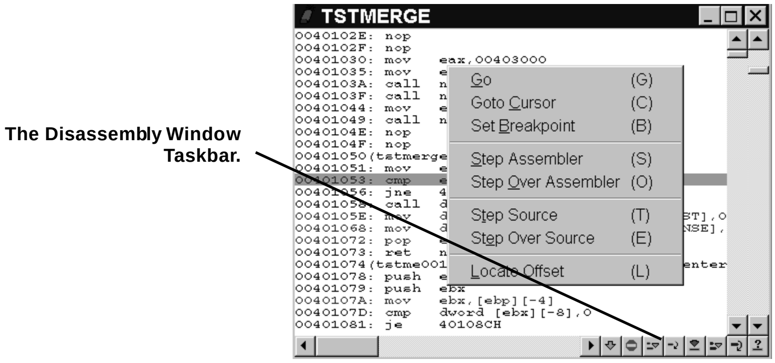
```
PROGRAM

INCLUDE('Equates.
INCLUDE('TplEqu.C
INCLUDE('Keycodes
INCLUDE('Errors.C
MAP
  MODULE('TSTMEOO
    Main
  END
  MODULE('TSTMEOO
    LocalMenu
  END
  MODULE('TSTME_S
    CheckOpen(FIL
    ReportPreview
    StandardWarni                          <STR
    SetupStringSt                          NG>)
    NextStringSto
    SetupRealStop
    NextRealStop,
    INIRestoreWin
    INISaveWindow
  END
  MODULE('TSTME_RU.CLW')
```

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.

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



```
0040102E: nop
0040102F: nop
00401030: mov    eax,00403000
00401035: mov    e
0040103A: call   n
0040103F: call   n
00401044: mov    e
00401049: call   n
0040104E: nop
0040104F: nop
00401050(tstmerge
00401051: mov    e
00401053: cmp    e
00401056: jne    4
00401058: call   d
0040105E: mov    d                              ST],0
00401068: mov    d                              NSE],
00401072: pop    e
00401073: ret    n
00401074(tstme001                               enter
00401078: push   e
00401079: push   ebx
0040107A: mov    ebx,[ebp][-4]
0040107D: cmp    dword [ebx][-8],0
00401081: je     40108CH
```

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.

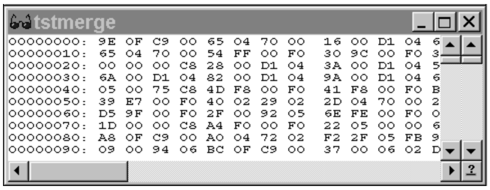
Blue text has a corresponding source statement associated with it. Moving the cursor to a line with blue text moves the cursor in the **source** window to the corresponding source line.

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.

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



```
00000000:  9E  0F  C9  00  65  04  70  00   16  00  D1  04  6
00000010:  65  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  82  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  06  BC  0F  C9  00   37  00  06  02  D
```

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.

Setting Breakpoints

Normally, when debugging an application, you'll 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 break point:

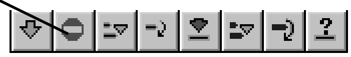
1. Navigate to the source or assembler code where you want the debugger to break.

   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 the breakpoint button.

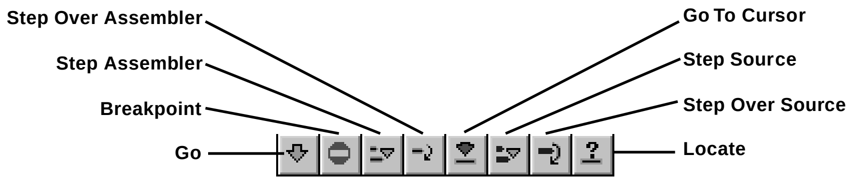
   The breakpoint button appears on the **source** and **disassembly** window taskbars with a round red icon. The breakpoint button acts as a toggle. Pressing it a second time removes the breakpoint.

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 runtime 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.

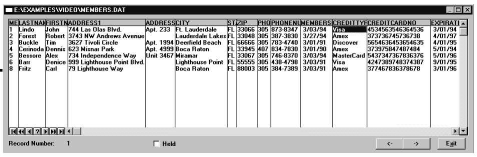
Changing Variable Values

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.

USING THE DATABASE MANAGER

Contents

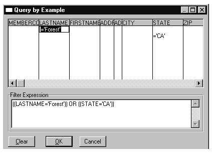


E:\EXAMPLES\VIDEO\MEMBERS.DAT

ME	LASTNAM	FIRSTN	ADDRESS1	ADDRESS	CITY	ST	ZIP	PHO	PHONENU	MEMBERS	CREDITTYP	CREDITCARDNO	EXPIRATI
1	Lindo	John	744 Las Olas Blvd.	Apt. 233	Ft. Lauderdale	FL	33066	305	873-8347	3/03/94	Visa	4534563546364536	3/01/94
2	Forest	Robert	3743 NW Andrews Avenue		Lauderdale Lakes	FL	33048	305	387-3830	3/27/94	Amex	373736745736738	4/01/97
3	Buckle	Tim	3627 Tivoli Circle	Apt. 1994	Deerfield Beach	FL	66666	305	783-4740	3/01/91	Discover	5654636453654635	4/01/95
4	Cerinoda	Dennis	623 Misnar Park	Apt. 4999	Boca Raton	FL	33945	407	834-7830	3/01/90	Amex	373975847487484	5/01/94
5	Bessore	Alex	734 Independence Way	Unit 3467	Miramar	FL	33067	305	746-8370	3/03/94	MasterCard	5437347367836376	5/01/96
6	Barr	Denice	999 Lighthouse Point Blvd.		Lighthouse Point	FL	55555	305	438-4798	3/03/91	Visa	4247389748374387	9/01/95
8	Fritz	Carl	79 Lighthouse Way		Boca Raton	FL	88003	305	384-7389	3/03/91	Amex	377467836378678	3/01/96

Record Number: 1

Use the Database Manager to interactively browse through your data files. You can add, modify, or delete Records.



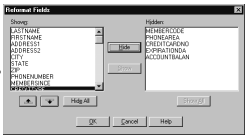
Query by Example

MEMBERCO	LASTNAME	FIRSTNAME	ADDR	AD	CITY	STATE	ZIP
	='Forest'					='CA'	

Filter Expression

((LASTNAME='Forest')) OR ((STATE='CA'))

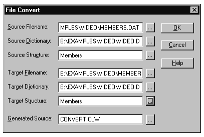
Use Query-by-Example (QBE) to find records that meet specific criteria.



Reformat Fields

Shown:	Hidden:
LASTNAME	MEMBERCODE
FIRSTNAME	PHONEAREA
ADDRESS1	CREDITCARDNO
ADDRESS2	EXPIRATIONDA
CITY	ACCOUNTBALAN
STATE	
ZIP	
PHONENUMBER	
MEMBERSINCE	

Format the list to display only those fields you choose.



File Convert

Source Filename:	MPLES\VIDEO\MEMBERS.DAT
Source Dictionary:	E:\EXAMPLES\VIDEO\VIDEO.D
Source Structure:	Members
Target Filename:	E:\EXAMPLES\VIDEO\MEMBER
Target Dictionary:	E:\EXAMPLES\VIDEO\VIDEO.D
Target Structure:	Members
Generated Source:	CONVERT.CLW

Use the File Conversion Utility to convert existing dtat files to a new format. You can also generate source code to produce an executable to convert data files for your end users.

THE DATABASE MANAGER—AN OVERVIEW

The Database Manager is tool is provided to allow 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

Database Manager was 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 ensured 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 *Using the Dictionary Editor.*

BROWSING DATA FILES

There are three ways to browse data files with the Database Manager.

- Through 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...** .

Which 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—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 file is displayed, and ready for any Database Manager operation.

From the Open File Dialog

To open an existing data file:

1. Choose **File ➤ Open** (or press the **Open** button on the **Pick List**).

The **Open** dialog appears.

2. Select the **Database** tab.
3. Highlight the file you want to open, and press the **Open** button.

   A dialog appears prompting for the File Driver and file information.
4. Select the Database Driver from the 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 appendix for more information on valid driver strings for specific file systems.
6. Press the **OK** button.

   The file is displayed, and ready for any Database Manager operation.

From the Browse Database Menu Command

The Browse Database Menu command displays a specialized pick list displaying recently opened data files or logical tables in database files which contain multiple files (ODBC and TopSpeed).

1. Choose **File ➤ Browse Database.**

   The Database Manager's Pick List appears, displaying recently opened files.
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 desired driver, then press the **OK button.**
4. If you are browsing an ODBC Data Source or a TopSpeed database with multiple tables in a single file, a dialog appears to allow you to select the table to browse.

The file is displayed, and ready for any Database Manager operation.

CLOSING A DATA FILE

Database Manager asks if you want to creates a backup copy before modifying any data in a file. Creating a backup file allows you to 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.

CHANGING THE 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).

   The **Select File Order** dialog appears, listing the available Keys and “Record Order”.

2. Highlight the key which matches the desired sort order (or Record Order), then press the **Select** button.

The file is displayed in the selected sort order, and ready for any Database Manager operation.

VIEWING FILE STATISTICS

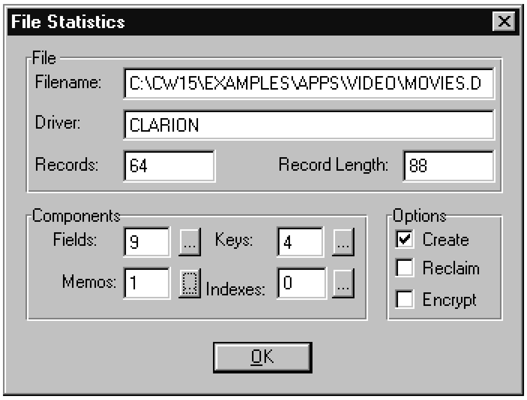
The File Statistics command allows you to examine file information including:

Filename	The DOS file name and PATH for the data file
Driver	The File System the file uses
Records	The total number of records in the file (including deleted records).
Record Length	The size of each record.
Fields	The number of fields in the file. Pressing the ellipsis (...) button displays the field layout.
Keys	The number of keys in the file. Pressing the ellipsis (...) button displays the key components.
Memos	The number of memos (and BLOBs) in the file. Pressing the ellipsis (...) button displays the memo field layout.
Indexes	The number of indexes in the file. Pressing the ellipsis (...) button displays the index components.
Options	Create, Reclaim, and Encrypt attributes.

To view File Statistics:

1. Choose **File ➤ File Statistics**.

   The **File Statistics** dialog appears.



File Statistics

File
Filename: C:\CW15\EXAMPLES\APPS\VIDEO\MOVIES.D
Driver: CLARION
Records: 64
Record Length: 88

Components
Fields: 9
Keys: 4
Memos: 1
Indexes: 0

Options
☑ Create
☐ Reclaim
☐ Encrypt

2. To view additional information about Fields, Keys, Memos, or Indexes, press the ellipsis (...) button next to the appropriate control.

WORKING WITH COLUMNS

Database Manager allows you to 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**.

   The **Select Columns to Show** dialog appears, with scrolling list of the hidden columns.
2. Highlight the desired field and press the **OK** button, or press the **All** button to show all fields.

   The **Show Fields** dialog reappears, allowing you to select other fields to restore to view. 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.

Using Reformat

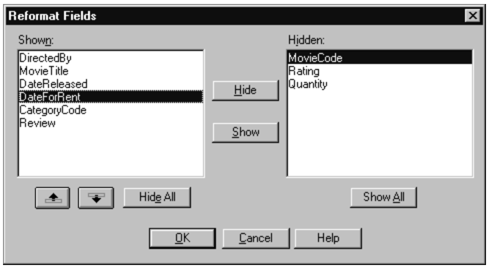
The Reformat command allows you to quickly define the desired view. You can specify the fields to hide or show and set the order of the columns all at once.

1. Choose **Column ➤ Reformat**.

   The **Reformat Fields** dialog appears.

❏ *To show all fields,* press the **Show All** button.

- ❑ *To hide all fields*, press the **Hide All** button.
- ❑ *To hide individual fields*, highlight the desired field in the **Shown Fields** list box, 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.



SETTING 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 desired 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.

SETTING COLUMN WIDTH

You can adjust the column display width for individual fields.

To adjust the display width for a single field:

1. CLICK-AND-DRAG the grid line to the right of the column.

CHANGING A COLUMN'S DISPLAY PICTURE

You can change any column's display picture. This enables you to view data in any supported format.

To change column's display:

1. Highlight the desired column.
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.

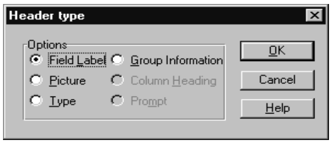
CHANGING THE COLUMN HEADER

Database Manager allows you to specify what displays as a column header.

To specify the column header:

1. Choose **Column ➤ Header.**

   The **Header** sub-menu appears, displaying the valid options. A check mark next to an option indicates it is enabled.



The available Header Options are:

Field Label	Displays the field's label from the Database Dictionary.
Picture	Displays the field's display picture from the Database Dictionary.
Type	Displays the field's data type from the Database Dictionary.
Group Information	Displays the field's GROUP information from the Database Dictionary.
Column Heading	Displays the Default Column Heading from the Database Dictionary.
Prompt	Displays the default prompt from the Database 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 cursor keys move to first and last columns, respectively.
- PAGE UP and PAGE DOWN scroll up and down, respectively, between record screens.
- CTRL+PAGE UP and CTRL+PAGE DOWN moves to the first or last record.
- INSERT allows you to add a record
- DELETE allows you to delete a record
- IN Browse Mode, ENTER allows you to edit the currently highlighted field on the current record. IN Edit Mode, ENTER accepts your entry in the current field.

Database Manager also provides VCR controls to navigate through files:

Using the Locate Command

This 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.

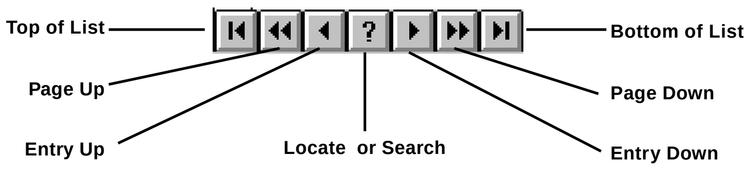
1. Choose **Edit ➤ Locate.**
2. Type the desired value(s) in the Key field(s).
3. Press the **OK** Button.

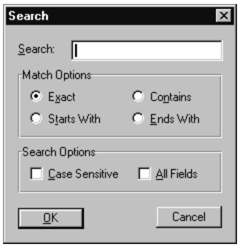
   The highlight bar is positioned on the first occurrence of the specified value(s) in the key field(s). If the value entered does not exist, the next highest match is highlighted

Search and Find Next

This 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 desired value in the **Search For** field.
3. Select the appropriate radio buttons for the desired type of search.

   Valid options are:

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 appears to report the search's progress by displaying the number of records searched and to provide the opportunity to abort the search. When the search is complete, the highlight bar is positioned on the first record found 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. To continue searching repeat the last step.

USING QUERY-BY-EXAMPLE

Query-by-Example (QBE) is a powerful tool to find information in a data file. This allows you to 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 runtime 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.

Warning: 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 creates a backup copy before modifying any data in a file. Creating a backup file allows you to 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 easily edit any field of any record in Database Manager.

1. Highlight the desired field in the desired record.
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 go to the next field, or SHIFT+TAB to go to the previous field.

Adding Records

Database Manager allows you to 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 Edit Mode in the first field.

2. Type the information you want to enter in a field, then press TAB to go to the next field (or SHIFT+TAB to go to the previous field). Repeat for all fields in which you want to enter data.
3. When you have completed all desired data entry for the record, press ENTER.

The record is added to the file and keys are updated.

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

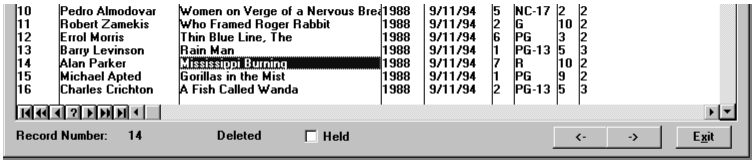
By default only active records are shown; however, you can display deleted records. You can use this feature to browse recently deleted records or undelete deleted records.

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.



10	Pedro Almodovar	Women on Verge of a Nervous Brea	1988	9/11/94	5	NC-17	2	2
11	Robert Zamekis	Who Framed Roger Rabbit	1988	9/11/94	2	G	10	2
12	Errol Morris	Thin Blue Line, The	1988	9/11/94	6	PG	3	2
13	Barry Levinson	Rain Man	1988	9/11/94	1	PG-13	5	3
14	Alan Parker	Mississippi Burning	1988	9/11/94	7	R	10	2
15	Michael Apted	Gorillas in the Mist	1988	9/11/94	1	PG	9	2
16	Charles Crichton	A Fish Called Wanda	1988	9/11/94	2	PG-13	5	3

Record Number: 14 Deleted ☐ Held

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.

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.

Sending a Driver String

Database Manager enables you to communicate with the file driver for the open file. This is equivalent to issuing a SEND command.

Refer to the *Database Drivers* Appendix for valid driver strings for each specific file driver.

Saving a File Definition as Source Code

Database Manager allows you to 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 file name for the source code.
3. Press the **OK** button.

   The source code for the file definition is created.

CONVERTING A DATA FILE

The file conversion utility allows you to convert the records in an existing data file to a new file format. When you modify a data dictionary and application, you can use the conversion utility to convert your existing data to the modified format.

The method you use to call the file conversion utility affects its behavior. If you open the converter through the Dictionary Editor (with the appropriate .DCT file open) the converter 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. The information stored in a file varies with different file systems.

There are two methods of converting a file—immediate conversion and Generating Source for File Conversion.

Automatic conversion runs the conversion process on a data file one time. This is useful to convert your sample data during the development process. Generating Source for File Conversion creates a source code file and Project, allowing you to make any desired modifications before compiling. Generating and compiling source also creates an executable file that you can ship to end users to convert their data files to the new format.

The conversion utility creates backup files for both conversion methods (automatic or generate source). If you specify a target file name that differs from the original, then the original files are not renamed and are left in place.

Immediate Conversion

If a file's definition needs to be changed and meaningful data exists, follow these steps to convert the file. This method does not create an executable file. It converts the data file on your system to a new format. If you want to create a file conversion program that you can ship with your application, see *Generating Source for File Conversion*.

Tip: It is always a good idea to make backup copies of your files before running any conversion process.

Note: If you change the name of a field, you must generate source code, and edit the source code to make the field assignments. Otherwise, your data will be lost.

1. Open (load) the dictionary that contains the file to be modified.
2. Modify the data file definition as desired (add fields or keys, change the file driver type, etc.).
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 conversion process is now complete and the file is displayed.

Generating Source for File Conversion

If a file's definition needs to be changed and meaningful data exists, follow these steps to convert the file. This method creates an executable file that you can ship to end users to convert their data files. If you want to convert a file without creating a file conversion program see How to Convert a File (without generating source) .

1. Open (load) the dictionary that contains the file to be modified.
2. Copy the data file definition to a new name. To copy a file definition, highlight the file to be copied in the Files List and press CTRL+C, then press CTRL+V to paste it. You will be prompted to supply a new name and prefix. (Example - copy Customer to OldCustomer)

   An alternative would be to copy the entire dictionary to a new name. You might use this method if there are multiple files to be converted in one session. Clarion for Windows allows files to be converted from one dictionary to another.

3. After the file definition has been copied, make any necessary changes (add fields, change the file driver type, etc.) to the definition with the original name. In our example above, the Customer file is the file to be modified.
4. Save the Dictionary after file modification and close it.

   The Dictionary file ***must*** be closed in order to use it for file conversion.
5. Load the file in the Database Manager File utility (**File ➤ Open , Database Tab**)
6. Next, you will be prompted to pick a file to load. For this example, you would select the Customer file.

   The Customer file displays.
7. Choose **File ➤ File Convert** (or press CTRL+V).

   The File Convert dialog appears, prompting for the information below:

Source Filename	Specifies the file to convert. This defaults to the file opened by the Database Manager.
Source Dictionary	Specifies the dictionary which contains the file definition for the source data file. A Source Dictionary is not required.
Source Structure	Specifies the structure (within the dictionary) which defines the source file. A Source Structure is not required.
Target Filename	Specifies the name of the new file. This defaults to the current file name.
Target Dictionary	Specifies the dictionary which contains the file definition to which to convert. A Target Dictionary is required.
Target Structure	Specifies the structure (within the dictionary) of the target file. The Target Structure is required.

8. Specify the file name for the generated source code of accept the default of CONVERT.CLW.
9. Press the OK button.

   This generates a source file. This file can now be compiled and linked to an executable program which will perform the file conversion.

NOTE: Prior to executing the source conversion program, the current data file loaded into the Database Manager must be closed.

10. Press the Exit button to close the data file in the Database Manager.
11. Load the conversion program by choosing **File ➤ Open** and selecting the **Source** tab.
12. Select *CONVERT.CLW* (or the file name you specified) in the **File Open** dialog. The conversion source code is displayed in the Text Editor.

Tip: If you changed the name of a field, edit the source code to make the field assignments. Otherwise, your data will be lost. See *Editing Source Code to Make Field Assignments.*

13. The project file must now be loaded. Choose Project > Set. Select the project file. This defaults to CONVERT.PRJ.
14. Finally, you may now Make and/or Run the conversion program.

    After the conversion program runs:

15. Check the file that has just been converted by loading the new (target) file back into the Database Manager.

    After viewing the file converted, some clean up steps are all that's left to do:

16. If the file converted was located in a different directory, you may now copy it into the working program directory. If you had originally renamed the file and placed it in the same directory, you may rename it to the original file name at this time.
17. The "old" file definition may now be deleted from the active dictionary, or archived into a backup dictionary file.

    The conversion process is now complete. This example creates CONVERT.EXE which may be shipped 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 handled automatically except in two cases:

- If a field's label is changed
- If a field is split into two separate fields.

In these cases you must modify the source code to handle the field assignments. The portion of the source code you'll 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
```

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. However, if the field labels do not match, no assignment is made. For example, if you change the LastName field to Surname, an comment statement is generated to alert you of an assignment that may need to be made:

```
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
```

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
```

Writing the assignment statements to split the contents of a field into two fields involves a little more work, but using string slicing minimizes the effort. For this example, let's assume that 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 a another. Assuming that these fields are numeric data types, you will 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 will look 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
```

Notice that there is an assignment from the original PhoneNumber field to the new PhoneNumber field. However, since the new field only stores seven digits, you must edit this. To handle the field assignments, you will 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]
```

For more information on String Slicing, see *Implicit Arrays and String Slicing* in Chapter 4 of the *Language Reference*.

PRINTING DATA

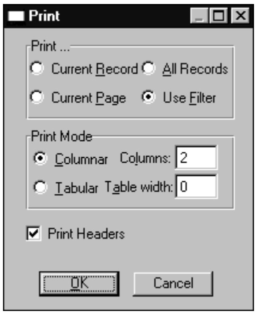
You can print data from Database Manager. You can use these simple reports to look at the data in your files.

To print data:

1. Choose **File ➤ Print.**

   The **Print** dialog appears, allowing you to select the report's format.

2. Select the appropriate radio buttons from the **Print...** group to specify the records to print.



Print ...: ○ Current Record ○ All Records ○ Current Page ● Use Filter

Print Mode: ● 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.

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.

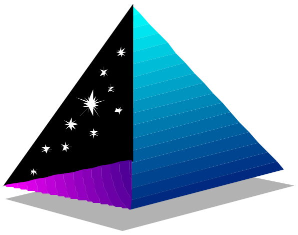
5. If you selected **Tabular,** specify the total number of characters to print for one record in the **Table width** field.

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).

WINDOWS DESIGN ISSUES

Contents



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.

OVERVIEW

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.

Microsoft makes specific recommendations on what a Windows application should look like, and how the controls in a window should look and act. There is even a set of standard menu and command recommendations.

This chapter also discusses the use of color in your application.

DESIGN PRINCIPLES

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

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, utilizes 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 is very important. As much as possible, an application should utilize 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 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 allows you to 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 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.

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.

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 paradigm shift 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 via messages just what it is the user wants the program to do. This is the opposite of DOS programming.

Windows messages inform 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 utilizing your main application window as the backbone for your program. Generate another window—dialog box, MDI window, etc.—only upon some user action in the main window.

WINDOWS AND WINDOW ELEMENTS

This section describes common Windows elements, including window types and controls. Choosing the right element for the right procedure will help your users get the most out of your program.

There are three types of windows: 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 re-sizes 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 sizable frames, a Control-menu box, and Minimize and Restore buttons. In Clarion, you add these attributes to the window via the Window Properties dialog.

MDI

Clarion enables the Multiple Document Interface (MDI). 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 re-sizes the application window to make it smaller than the document window, the document window will appear clipped.

In database programming, the document "file" is often the database. Use document windows to display browse windows, views, and queries. Since Clarion provides the tools to do virtually anything in a document window, be conservative in your use of document windows. *Do* use a document window to display the contents of a text file, for example, as a graphics viewer, or to view a report. *Don't* use a document window for procedures which belong in a dialog box, such as record entry.

Microsoft recommends each document window contain a title bar with the name of the document, a Control-menu box, and a Maximize button. Optionally, you can include a minimize button.

Dialog Boxes

Use a dialog box when you require additional information from a user to complete a task. Adding a record to the database is an example of a task—data entry for the new record would be the process of getting additional information.

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. Since the guidelines allow for wide variation, this section will only present a few pointers which will help you design dialogs that are easier to use.

- ❏ Using either the dialog box caption bar or static text in the box, briefly explain the function of the dialog, 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 text edit (entry) box. This allows the user to simply type a word or words at the keyboard without having to position the cursor.
- ❏ Place dialog box 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 using the MESSAGE function. The Stop, Information, and Question icons are familiar to users from other applications.

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 allows you to use either 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 **<u>O</u>K**—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 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 allows you to select the font for text boxes. We suggest using the default system font (which is a helvetica font, 10 pts., bold style). 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 allows you to select the font for static text. We suggest using the default system font (which is a helvetica font, 10 pts., bold style). 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 allows you to control the user's movement through the tab "pages." This allows you to 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 allow the user to 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 "**E<u>x</u>it**". If the first word in the control label is less important than another, use the other (e.g., "**by <u>A</u>scending 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 utilize 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 utilize 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 utilize 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.

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.

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.

Accelerator Keys

A number of commands have gained standard accelerator (or alert, or hot) keys. When creating your application, should you utilize 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

COLOR

Color will 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:

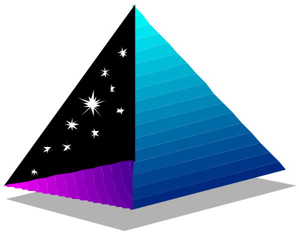
❏ Windows allows users to select default colors for window text and background. It’s best to accept these default colors for the parts of the program which require the most data entry: the user has expressed a preference, so you should respect it!

❏ 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.

DATABASE DRIVERS



Clarion for Windows achieves database independence with its built-in driver technology, enabling you to access data from virtually any file system. Many file drivers are available and more are being added. All of the file drivers read and write in the file system's native format without temporary files or import/export routines.

Often, your application's main purpose is accessing existing data in its original format. For those times, you just plug in the appropriate file driver. For the times when you're not "locked into" a particular file system, this appendix provides tips on the file drivers best suited for different jobs. You can choose the right tool depending on the type, size, and nature of the data files necessary for your application.

The commands for accessing data from different systems are the same; simply choose the correct file driver from the drop down list within your Data Dictionary, and don't worry about it.

There are settings you can specify, with driver strings (the second parameter of the DRIVER attribute), to optimize the way your application creates, reads, and writes data files for a specific driver. To specify a driver string with the Data Dictionary, type it in the **Options** field in the *New File Properties* dialog, as described in the *Using the Dictionary* chapter. To send a driver string in executable code (after the application initializes the driver), use the SEND() function, described in the *Language Reference.*

Driver strings are all preceded by a forward slash character (/). SEND function commands can take two formats—one with an equal sign to modify a switch setting, the other without an equal sign to return the value of the switch.

Driver strings are sent to the file driver when the file is opened. The SEND function sends a command to modify a setting. Some driver strings have no effect after the file is open, so the SEND function syntax to modify the setting is not listed. However, the SEND function to return the value of the switch is valid for all driver strings.

ASCII FILES

The ASCII driver reads and writes standard ASCII files without field delimiters. This is often used for mainframe data import/export via an ASCII flat-file. A carriage-return/line-feed delimits records. The ASCII driver does not support keys.

Files:	**CWASC16.LIB**	Windows Export Library (16-bit)
	CWASC32.LIB	Windows Export Library (32-bit)
	CLASC16.LIB	Windows Static Link Library (16-bit)
	CLASC32.LIB	Windows Static Link Library (32-bit)
	CWASC16.DLL	Windows Dynamic Link Library (16-bit)
	CWASC32.DLL	Windows Dynamic Link Library (32-bit)

Tip: Due to its lack of relational features and security (anyone can view and change an ASCII file using Notepad), it's unlikely you'll use the ASCII driver to store large data files. But it can help you create a text file viewer—use it to open a file, and read it in to a multi-line edit or listbox control!

Supported Data Types

```
STRING
GROUP
```

File Specifications/Maximums

```
File Size:               4,294,967,295 bytes
Records per File:        4,294,967,295 bytes
Record Size:             65,520 bytes
Field Size:              65,520 bytes
Fields per Record:       65,520 bytes
Keys/Indexes per File:   n/a
Key Size:                n/a
Memo fields per File:    n/a
Memo Field Size:         n/a
Open Data Files:         Operating system dependent
```

Driver Strings and SEND functions

Driver strings (the second parameter of the DRIVER attribute) are all preceded by a forward slash character (/). SEND function commands can take two formats—one with an equal sign modifies a switch setting and return the value of the previous switch setting; the other format (without an equal sign) returns the value of the switch.

Driver strings are sent to the file driver when the file is opened. The SEND function sends a command to modify a setting after the file is open. Some driver strings have no effect after the file is open, so the SEND function syntax to modify the setting is not listed. However, the SEND function syntax to return the value of the switch is listed for all driver strings.

`/FILEBUFFERS=n`

Specifies a value for the number of buffers used to read and write to the file. The ASCII driver allocates internal buffers of 512 bytes, or the size of the record, whichever is larger. The default number of buffers is 2 for files opened denying write access to other users, and 1 for all other open modes. Use the optional driver string to increase the buffers should you find access to records is slow.

`SEND(file, ‘FILEBUFFERS’)`

Returns a STRING containing the number of bytes in the buffers in STRING format

`/TAB= n`

Specifies TAB/SPACE expansion. The ASCII driver expands TABs (ASCII character 9) to spaces when reading. The value indicates the number of spaces with which to replace the tab, subject to the guidelines below. The default value is 8.

If n > 0 , spaces replace each tab until the character pointer moves to the next multiple of *n*. For example, with the default of 8, if the TAB character is the third character in the record, 6 spaces replace the TAB.

If n = 0, the driver removes tabs without replacement.

If n < 0, the driver removes tabs with the positive value of *n* spaces. For example, “TAB=-4” causes 4 spaces to replace every tab, regardless of the position of the tab in the record.

If n = -100, tabs remain as tabs; the driver *does not* replace them with spaces.

`SEND(file, ‘TAB’)` Returns the number of spaces which replace the tab character in the form of a STRING.

`/ENDOFRECORD=n,<m>` Specifies the end of record delimiter.

n represents the number of characters that make up the end-of-record separator.

m represents the ASCII code(s) for the end-of-record characters, separated by commas. The default is 2,13,10, indicating 2 characters mark the end-of-record, namely, carriage return (13) and line feed (10).

`SEND(file,’ENDOFRECORD’)`
Returns the end of record delimiter in the form of a STRING.

Tip: Mainframes frequently use a carriage return to delimit records. You can use /ENDOFRECORD to read these files.

`/QUICKSCAN=on|off`
`SEND(file,’QUICKSCAN=on|off’)`
Specifies buffered access behavior. The ASCII driver reads a buffer at a time (not a record), allowing for fast access. In a multi-user environment these buffers are not 100% trustworthy for subsequent access, because another user may change the file between accesses. As a safeguard, the driver rereads the buffers before each record access. To *disable* the reread, set QUICKSCAN to ON. The default is ON for files opened denying write access to other users, and OFF for all other open modes.

`SEND(file,’QUICKSCAN’)`
Returns the Quickscan setting (ON or OFF) in the form of a STRING(3).

`/CLIP=on|off`	The driver automatically removes trailing spaces from a record before writing it to file. To disable this feature, set CLIP to OFF. The default is ON.
`SEND(file,'CLIP')`	Returns the CLIP setting (ON or OFF) in the form of a STRING(3).

Unsupported Functions and Attributes

Memos:	`NOMEMO()`
Transaction Processing:	`COMMIT(), LOGOUT(), ROLLBACK()`

Key Processing:	`BUILD(key), BUILD(index)`
	`GET(file,key), GET(key,keypointer)`
	`RESET(key,string)`
	`SET(file,key), SET(key), SET(key,key),`
	`SET(key,keypointer), SET(key,key,filepointer)`
	`DUPLICATE()`
	`POINTER(key)`
	`POSITION(key)`
	`RECORDS(key)`
	`REGET(key,string)`

Record Locking:	`HOLD(), RELEASE()`
File Buffering:	`STREAM()`
File Information:	`RECORDS(file)`
Sequential Processing:	`PREVIOUS(), BOF(), SKIP()`
File Manipulation:	`BUILD(), DELETE(), PACK(), WATCH(), REGET()`

Miscellaneous

- POSITION(file) returns a STRING(4).

BASIC FILES

The BASIC file driver reads and writes comma delimited ASCII files. Quotes (“ “) surround strings, commas delimit fields, and a carriage-return/line-feed delimits records. The original BASIC programming language defined this file format. The Basic driver does not support keys.

Tip: The Basic file format provides a good choice for a common file format for sharing data with spreadsheet programs. A common file extension used for these files is *.CSV, which stands for “comma separated values.”

Files:	**CWBAS16.LIB**	Windows Export Library (16-bit)
	CWBAS32.LIB	Windows Export Library (32-bit)
	CLBAS16.LIB	Windows Static Link Library (16-bit)
	CLBAS32.LIB	Windows Static Link Library (32-bit)
	CWBAS16.DLL	Windows Dynamic Link Library (16-bit)
	CWBAS32.DLL	Windows Dynamic Link Library (32-bit)

Supported Data Types

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

File Specifications/Maximums

File Size:	4,294,967,295 bytes
Records per File:	4,294,967,295 bytes
Record Size:	65,520 bytes
Field Size:	65,520 bytes
Fields per Record:	65,520 bytes
Keys/Indexes per File:	n/a
Key Size:	n/a
Memo fields per File:	0
Memo Field Size:	n/a
Open Data Files:	Operating system dependent

Driver Strings and SEND functions

Driver strings (the second parameter of the DRIVER attribute) are all preceded by a forward slash character (/). SEND function commands can take two formats—one with an equal sign modifies a switch setting and return the value of the previous switch setting; the other format (without an equal sign) returns the value of the switch.

Driver strings are sent to the file driver when the file is opened. The SEND function sends a command to modify a setting after the file is open. Some driver strings have no effect after the file is open, so the SEND function syntax to modify the setting is not listed. However, the SEND function syntax to return the value of the switch is listed for all driver strings.

`/FILEBUFFERS=n`	Specifies a value for the number of buffers used to read and write to the file. The Basic driver allocates internal buffers of 512 bytes, or the size of your record, whichever is larger, to store the retrieved data. The default number of buffers is 2 for files opened denying write access to other users, and 1 for all other open modes. Use the optional driver string to increase the buffers should you find access to records is slow.
`SEND(file, 'FILEBUFFERS')`	Returns the number of buffers in the form of a STRING.
`/ENDOFRECORD=n,<m>`	Specifies the end of record delimiter. *n* represents the number of characters of the end-of-record separator. *m* represents the ASCII code(s) for the end-of-record characters, separated by commas. The default is 2,13,10 indicating 2 characters mark the end-of-record, namely, carriage return (13) and line feed (10).
`SEND(file,'ENDOFRECORD')`	Returns the end of record delimiter in the form of a STRING.

Tip: Mainframes frequently use a carriage return to delimit records. You can use /ENDOFRECORD to read these files.

`/QUICKSCAN=on|off`

`SEND(file,'QUICKSCAN=ON|OFF')`

Specifies buffered access behavior.

The Basic driver reads a buffer at a time (not a record), allowing for fast access. In a multi-user environment these buffers are not 100% trustworthy for subsequent access, because another user may change the database between accesses. As a safeguard, the driver rereads the buffers before each record access. To *disable* the reread, set QUICKSCAN to ON. The default is ON for files opened denying write access to other users, and OFF for all other open modes.

`SEND(file,'QUICKSCAN')`

Returns the Quickscan setting (ON or OFF) in the form of a STRING(3).

`/FIELDDELIMITER=n,<m>`

Specifies the end-of-field separator.

n represents the number of characters that make up the end-of-field separator.

m represents the ASCII code(s) for the end-of-field characters, separated by commas. The default is 1,44, which indicates the “comma” character.

`SEND(file,'FIELDDELIMITER')`

Returns the value of the field delimiter in the form of a STRING.

`/COMMA=n`

Specifies a single character end-of-field separator. *n* represents the ASCII code for the end-of-field character. The default is 44, which is equivalent to “/FIELDDELIMITER=1,44.”

`SEND(file,'COMMA')`

Returns the ASCII code for the single character end-of-field delimiter in the form of a STRING.

Tip: TAB-delimited values are a common format compatible with the Windows clipboard. Using the BASIC file driver string /COMMA=9 allows you to read Windows clipboard files

`/QUOTE=n`	Specifies a single character string delimiter. *n* represents the ASCII code. The default is 34, the ASCII value for the quotation mark character.
`SEND(file,'QUOTE')`	Returns the ASCII code value of the single character string delimiter in the form of a STRING.
`/ALWAYSQUOTE=on|off`	For compatibility with Basic format data files created by products which do *not* place string values in quotes, set ALWAYSQUOTE to off. When the contents of a string field includes the comma or quote character(s), and ALWAYSQUOTE is off, the Basic driver automatically places quotes around the string when writing to file. This also applies to delimiter characters specified with FIELDDELIMITER, or COMMA. For example, with the defaults in use and ALWAYSQUOTE off, a STRING field containing the value *1313 Mockingbird Lane, Apt. 33* is automatically stored as: "1313 Mockingbird Lane, Apt. 33"
`SEND(file,'ALWAYSQUOTE')`	Returns the ALWAYSQUOTE setting (ON or OFF) in the form of a STRING(3).

Unsupported Functions and Attributes

Memos:	`NOMEMO()`
Transaction Processing:	`COMMIT(), LOGOUT(), ROLLBACK()`

Key Processing:	`BUILD(key), BUILD(index)`
	`GET(file,key), GET(key,keypointer)`
	`RESET(key,string)`
	`SET(file,key), SET(key), SET(key,key),`
	`SET(key,keypointer), SET(key,key,filepointer)`
	`DUPLICATE()`
	`POINTER(key)`
	`POSITION(key)`
	`RECORDS(key)`
	`REGET(key,string)`

Record Locking:	`HOLD(), RELEASE()`
File Buffering:	`STREAM()`
File Information:	`RECORDS(file)`
Sequential Processing:	`PREVIOUS(), BOF(), SKIP()`
File Manipulation:	`BUILD(), DELETE(), PACK(), WATCH(), REGET()`

Miscellaneous

The following demonstrates how to use the driver strings to create two popular file formats:

- Microsoft Word for Windows Mail Merge:

```
/ALWAYSQUOTE=OFF
/FIELDDELIMITER=2,13,7
/ENDOFRECORD=4,13,7,13,7
```

- TAB delimited format:

```
/COMMA=9
```

❖ POSITION(file) returns a STRING(4).

BTRIEVE FILES

This file driver reads and writes Btrieve files, using low-level direct access.

Under Clarion for Windows, the Btrieve file driver is implemented by using .DLLs and an .EXE supplied by Btrieve Technologies, Inc. (BTI). For an application to use a Btrieve file driver, the following BTI files must accompany the executable:

<u>**16-bit**</u>

WBTR32.EXE

WBTRLOCL.DLL

WBTRCALL.DLL

WBTRVRES.DLL

<u>**32-bit**</u>

Filenames were not available at press time.
Contact BTI for more information.

LICENSE WARNING: A registered Clarion for Windows owner cannot redistribute the above BTI files outside of his/her organization without a license from BTI. In order to obtain a license, please contact:

Btrieve Technologies, Inc.
5918 West Courtyard Drive, Suite 400
Austin, Texas 78730
Phone: (512)794-1719

For Client/Server-based Btrieve, Netware Btrieve is a server-based version of Btrieve that runs on a Novell server. The Btrieve requester program— BREQUEST.EXE must be loaded at each workstation before Windows is started.

A single file normally holds the data and all keys. Data filenames default to a *.DAT file extension. By default, the driver stores memos in a separate file, or optionally in the data file itself, given the appropriate driver string.

KEYs are dynamic, and automatically update when the data file changes.

INDEXes are stored separately from data files. INDEX files receive a temporary file name, and are deleted when the program terminates normally. INDEXes are static—they are not automatically updated when the data file changes. The BUILD statement creates or updates index files.

The Btrieve file format stores minimal file structure information in the file. The driver validates your description against the information in the file. It *is* possible to successfully open a Btrieve file that has key definitions that do not exactly match your definition. You must make certain that your file and key definitions accurately match the Btrieve file.

Files:	**CWBTRV16.LIB**	Windows Export Library (16-bit)
	CWBTRV32.LIB	Windows Export Library (32-bit)
	CLBTRV16.LIB	Windows Static Link Library (16-bit)
	CLBTRV32.LIB	Windows Static Link Library (32-bit)
	CWBTRV16.DLL	Windows Dynamic Link Library (16-bit)
	CWBTRV32.DLL	Windows Dynamic Link Library (32-bit)

An Owner Name is similar to a password. An encrypted Btrieve file uses the owner name as the encryption key.

Data Types

Clarion data type	Btrieve data type
BYTE	STRING (1 byte)
SHORT	INTEGER (2 bytes)
LONG	INTEGER (4 bytes)
SREAL	FLOAT (4 bytes)
REAL	FLOAT (8 bytes)
BFLOAT4	BFLOAT (4 bytes)
BFLOAT8	BFLOAT (8 bytes)
PDECIMAL	DECIMAL
STRING	STRING
CSTRING	ZSTRING
PSTRING	LSTRING
DATE	DATE
TIME	TIME
USHORT	UNSIGNED BINARY (2 bytes)
ULONG	UNSIGNED BINARY (4 bytes)
MEMO	STRING,LVAR or NOTE (see below)
BYTE,NAME('LOGICAL')	LOGICAL*
USHORT,NAME('LOGICAL')	LOGICAL*
PDECIMAL,NAME('MONEY')	MONEY*
STRING(@N0n-),NAME('STS')	SIGNED TRAILING SEPERATE*
DECIMAL	

Notes:

- You can store Clarion DECIMAL types in a Btrieve file. However, you cannot build a key or index using the field.

- If you want to create a file with LOGICAL or MONEY field types, you must specify an external name of LOGICAL or MONEY, respectively. If you are accessing an existing file, the NAME attribute is not required.

  LOGICAL may be declared as a BYTE or USHORT, depending on whether it is a one or two byte LOGICAL:

  ```
  LogicalField1  BYTE     !One byte LOGICAL
  LogicalField2  USHORT   !Two byte LOGICAL
  ```

  MONEY may be declared as a PDECIMAL(x,2), where *x* is the total number of digits to be stored:

  ```
  MoneyField    PDECIMAL(7,2) !Store up to 99999.99
  ```

- Btrieve NUMERIC fields are not fully supported by the driver. Btrieve NUMERIC is stored as a string with the last character holding a digit and an implied sign.. The possible values for this last character are:

  ```
             1 2 3 4 5 6 7 8 9 0
  Positive:  A B C D E F G H I {
  Negative:  J K L M N O P Q R }
  ```

To access a NUMERIC field you must define a STRING(@N0x), where x is one less than the digits in the NUMERIC, and a STRING(1) to hold the sign indicator. The Btrieve driver does not maintain this sign field, the application must be written to directly handle it.

For example to access a NUMERIC(7) you would have:

```
NumericGroup    GROUP          !Store -999999 to 999999
Number            STRING(@N06)  !Numbers
Sign              STRING(1)     !Sign indicator
                END
```

File Specifications/Maximums:

```
File Size           :       4,000,000,000 bytes
Records per File    :       Limited by the size of the file
Record Size
    Client-based    :       65,520 bytes variable length
    Server based    :       54K variable length
Field Size          :       65,520 bytes
Fields per Record   :       65,520 bytes
Keys/Indexes per File:      24 with NLM5
                            256 with NLM6.
                            Client Btrieve v6.15
                            Page Size       Max Key Segments
                            512             8
                            1,024           23
                            1,536           24
                            2,048           54
                            4,096           119
                            This is the total number of
                            components. If you have a
                            multicomponent key built from three
                            fields, this counts as three
                            indexes when counting the number of
                            allowed indexes.
Key Size            :       255 bytes
Memo fields per File:       System memory dependent
Memo field size     :       65,520 bytes
Open Files          :       Operating system dependent
```

Driver Strings and SEND functions

Driver strings (the second parameter of the DRIVER attribute) are all preceded by a forward slash character (/). SEND function commands can take two formats—one with an equal sign modifies a switch setting and return the value of the previous switch setting; the other format (without an equal sign) returns the value of the switch.

Driver strings are sent to the file driver when the file is opened. The SEND function sends a command to modify a setting after the file is open. Some driver strings have no effect after the file is open, so the SEND function syntax to modify the setting is not listed. However, the SEND function to return the value of the switch is valid for all driver strings.

`/MEMO=SINGLE`	To access existing Btrieve files created with the Btrieve LEM from Clarion 2.1, or files with variable length records set MEMO to SINGLE.
`/MEMO=LVAR`	To access a file with variable length records, use a SINGLE style MEMO whose size equals the maximum size of the variable length component of the record. To add/put records to this style file with binary data stored in the variable length section, use the ADD(file,length), APPEND(file,length) and PUT(file,pos,length) functions. The driver ignores the *pos* parameter in the PUT function, but initialize it to *0 (zero)* for future compatibility. The ADD, APPEND or PUT functions will remove all trailing spaces for text memos and NULL characters for binary memos before storing the record.
`/MEMO=NOTE,<delimiter>`	To access Xtrieve data files that have data type of Note or LVar, set the driver string to NOTE and LVAR respectively. With the NOTE data type, specify the end-of-field delimiter. Specify the ASCII value for the delimiter. NOTE and LVAR memos do not require the use of the size variants of ADD, APPEND and PUT, when storing records. The end of record marker is not necessary for a NOTE style memo. The driver automatically adds the end of record marker before storing the record and removes it before putting the memo data into the memo buffer.

As an example, "/MEMO=NOTE,141" indicates a file with an Xtrieve Notes field using CR/LF as the delimiter. For more information on the Xtrieve data types refer to the documentation supplied by Novell.

`SEND(file,'MEMO')`	Returns the MEMO setting: NORMAL,NOTE,LVAR or SINGLE
`/PAGESIZE=<size>`	Optionally sets the Btrieve Page size at file creation time. The keyword must be upper case. It must always be a multiple of 512, with a maximum of 4096. Larger page sizes usually result in more efficient disk storage. *Do not add spaces before or after* the equal sign.
`SEND(file,'PAGESIZE')`	Returns the page size in the form of a STRING.
`/ALLOWREAD=[ON|OFF]`	By default, a Btrieve file created with an owner name may be accessed *only* in read-only mode when the owner name is not known. To prevent *all* access to the file without the owner name, set ALLOWREAD to OFF.
`SEND(file,'ALLOWREAD')`	Returns the ALLOWREAD setting (ON or OFF) in the form of a STRING(3).
`/COMPRESS=[ON|OFF]`	Btrieve allows you to compress the data before storage. This allows for a smaller storage requirement, but reduces performance. When COMPRESS is ON, CREATE creates a compressed Btrieve file.
`SEND(file,'COMPRESS')`	Returns the COMPRESS setting (ON or OFF) in the form of a STRING(3).
`/PREALLOCATE=n`	When creating a Btrieve file, you can preallocate *n* pages of disk space for the file. The default is zero.
`SEND(file,'PREALLOCATE')`	Returns the number of pages of allocated disk space in the form of a STRING.

`/FREESPACE=[0|10|20|30]`

Specifies the percentage of free space to maintain on variable length pages. The default is zero.

`SEND(file,'FREESPACE')` Returns the percentage of free space to maintain on variable length pages in the form of a STRING.

`/ACS=file_name` When creating a Btrieve file you can specify an alternate collating sequence that STRING keys will be sorted by. This sorting sequence is normally obtained from the sort sequence you define in the INI file for your program. However, Btrieve supplies files for doing case insensitive sorts. To create your file using these sort sequences you specify the name of the sort file in the driver string.

For example. To use the alternate collating sequence file UPPER.ALT you would specify:

```
AFile  FILE,DRIVER('BTRIEVE','/ACS=UPPER.ALT'),CREATE
```

`/APPENDBUFFER=size`

`SEND(file,'APPENDBUFFER=size')`

By default APPEND adds records to the file one at a time. To get better performance over a network you can tell the driver to build up a buffer of records then send all of them to Btrieve at once. This is done using SEND(file,'APPENDBUFFER=size') where size is the number of records you want to allocate for the buffer. The maximum value of size of the buffer.

`SEND(file,'APPENDBUFFER')`

Returns the number of records that will fit in the buffer.

`/BALANCEKEYS=[ON|OFF]')`

When creating a Btrieve file, you can use this driver string to tell Btrieve that Btrieve that all keys associated with the file must be stored in a balances btree. This saves disk space, but will slow down file adds, deletes and updates where key values change.

`SEND(file,'BALANCEKEYS')`	Returns the BALANCEKEYS setting (ON or OFF) in the form of a STRING(3).
`SEND(file,'FREEAPPENDBUFFER')`	Frees up the memory used by the append buffer allocated by a call to SEND(file,APPENDBUFFER=size). Returns the number of records that fitted in the old buffer.
`/LACS=`	With Btrieve v6.15 Btrieve added the feature of Local Alternate Collating Sequences. This allows your string key to sort based on the country code for the machine running your program. To use this feature you put '/LACS=' in your driver string.
`/LACS=country_ID,code_page`	With Btrieve v6.15 Btrieve added the feature of User-Defined Alternate Collating Sequences. This allows your string key to sort based on the DOS country code and code page for a particular country. To use this feature you put '/LACS=country_id,codepage' in your driver string. Note that there must be no spaces surrounding the comma.
`SEND(file,'LACS')`	Returns country_ID,code_page or the string ',' (if using machine-dependent LACS).
`/TRUNCATE=[ON|OFF]`	When creating a Btrieve file, you can use this driver string to tell Btrieve to truncate trailing spaces. This forces the record to be stored as a variable length records.
`SEND(file,'TRUNCATE')`	Returns the TRUNCATE setting (ON or OFF) in the form of a STRING(3).

Unsupported/Modified Functions and Attributes

- Key Attribute: `NOCASE`

  NLM 5 does not support case insensitive indexing. When necessary, you must supply an alternate collating sequence which implements case insensitive sorting.

  Btrieve supports an alternate collating sequence. However, NLM 6 does not support *both* NOCASE *and* an alternate collating sequence. If you specify both, the NOCASE attribute takes precedence. No error is returned from the SEND function.

- Buffering Control: `STREAM, FLUSH`

  There is no buffering control within the Btrieve driver.

- File Locking: `LOCK()`

  Btrieve does not support file locking. The driver does not return any error if you call it. If you require file locking, use LOGOUT.

- Record Access: `GET(file, fileptr, len)`
- File information: `BYTES()`
- File updates: `PUT(file, fileptr)`
- `SET(file, filepointer), SET(key, keypointer)`

  If a file or key pointer has a value of zero, or any other value that does not exist in the file, the driver ignores the pointer parameter. Processing is set to either file or key order, and the record pointer is set to the first element.

- `SET(key, key, filepointer)`

  If the filepointer has a value of zero, or any other value that does not exist in the file, processing starts at the first key value whose position is greater than (or less than for PREVIOUS) the filepointer. Not passing a valid pointer is inefficient.

- `EOF(file), BOF(file)`

  These functions are supported, but not recommended. They cause more disk I/O than ERRORCODE(). Btrieve returns *eof* when reading past the last record. This requires the driver must read the next record, then the *next* to see if it's at the end of file, then goes back to the record you want.

- `ADD(file), PUT(file)`

When using the LVAR and NOTE memo type, make certain that the memo has the appropriate structure. If the structure is incorrect and the driver calculates a length greater than the maximum memo size defined for that file, these functions fail and set errorcode to 57 - Invalid Memo File.***

❖ `DELETE(file)` *when stepping through in record order*

Tip: Btrieve's DELETE destroys positioning information when processing in file order. The driver attempts to reposition to the appropriate record. This is not always possible and may require the driver to read from the start of the file. Using key order processing avoids this possible slow down.

❖ `LOGOUT()`

Btrieve does not allow you to logout only certain files. When you issue a LOGOUT() call, all Btrieve files accessed during the transaction are logged out. This means the following code is illegal (as you cannot close a logged-out file:

```
LOGOUT(1,file1)
OPEN(file2)
CLOSE(file2)
```

❖ `APPEND()`

Btrieve does not support non-key updates. To emulate APPEND() behavior, the driver drops all indexes possible when APPEND() is first called. Calling BUILD() immediately after appending records rebuilds the dropped key fields.

❖ `BUILD()`

If used after an APPEND(), but before a file is closed, this adds the keys dropped by APPEND(). In all other cases BUILD() rebuilds the file and keys. If you only want to rebuild keys, doing a BUILD(key) for each key is faster than BUILD(file).

❖ `BUILD(DynamicIndex, expression,filter)`

Miscellaneous

- The driver stores records less than 4K as fixed length. It stores records greater than 4K as variable length. The minimum record length is 4 bytes. One record can be held in each open file by each user.

- The driver ignores any NAME attribute on a MEMO field. MEMO fields can reside either in a separate file, or in the data file if the driver string MEMO is set to SINGLE, LVAR or NOTE. If the driver string MEMO is not set, the separate MEMO file name is "MEM," preceded by the first five characters of the file's label, plus the file extension "DAT." Setting the driver string MEMO restricts you to one memo field per file.

- Btrieve allows you to open a file in five different formats: NORMAL, ACCELERATED, READ-ONLY, VERIFY or EXCLUSIVE. The equivalent Clarion OPEN() states are:

```
Btrieve State     Clarion OPEN/SHARE access mode
ACCELERATED   Read/Write with FCB compatibility mode (2H)
READ-ONLY     Read Only  (0H,10H,20H,30H,40H)
VERIFY        Write Only with FCB compatibility mode (1H)
EXCLUSIVE     Write Only with any Deny flag (11H,21H,31H,41H);
              Read/Write with Deny All, Read or Write (12H,22H,32H)
NORMAL        Read/Write with Deny None (42H)
```

- Btrieve allows a file to have a specified owner. See the driver string /READONLY for details on setting this flag. The file may also be encrypted. This is set with the ENCRYPT attribute. A file can only be encrypted when an owner name is supplied.

- Btrieve uses an unsigned long for its internal record pointer; negative values are stripped of their sign. We recommend the ULONG data type for your record pointer.

- Calculating Page Size:

  To determine the physical record length, add 8 bytes for each KEY that allows duplicates. Add 4 bytes if the file allows variable record lengths. Finally, allow 6 bytes for overhead per page.

  For example: If the record size is 300 bytes and the file has three KEYs that allow Duplicates, the total record size is:

	300	*record size*
x	24	*overhead for three KEYs with the DUP attribute*
=	324	*physical record length*

  A page size of 512 would only hold one such record, and 182 bytes per page would go unused (512 - 6 - 324). If the page size were

1024, three records could be stored per page and only 46 bytes would go unused (1024 - 6 - (324 * 3)).

You must load BTRIEVE.EXE with a page size equal to or greater than the largest page size of any file that you will be accessing.

When defining a file, the key definition does not need to exactly match the underlying file. For example, you can have a physical file with a single component STRING(20). You can define this as a key with two string components with a total length of 20. The rule is that the data types must match and the total size must match. However, if your Clarion definition does not exactly match the underlying file, the driver cannot optimize APPEND() or BUILD() statements.

❖ A Key's NAME attribute can add additional functionality.

`KEY,NAME('MODIFIABLE=true|false')`

Btrieve allows you to create a key that can not be changed once created. To use this feature you can use the name attribute on the key to set MODIFIABLE to FALSE. It defaults to TRUE.

`KEY,NAME('ANYNULL')`

Btrieve allows you to create a key that will not include a record if any key components are null. To create such a key you specify ANYNULL in the key name.

For example, to create a key that is non modifiable and excludes keys if any component is null:

```
  Key1   KEY(+pre:field1,-pre:field2),NAME('ANYNULL MODIFIABLE=FALSE')
```

KEY,NAME('REPEATINGDUPLICATE')

By default Btrieve version 6 stores a reference to only the first record in a series of duplicate records in a key. The other occurrences of the duplicate key value are obtained by following a link list stored at the record. To create an index where all duplicate records are stored in the key you use the NAME('REPEATINGDUPLICATE'). This produces larger keys, but random access to duplicate records is faster. (This feature is only available for version 6 files.)

❖ POSITION(file) returns a STRING(4).

❖ POSITION(key) returns a STRING the size of the key fields + 4 bytes.

CLARION FILES

The Clarion file driver is compatible with the file system used by Clarion Database Developer 3.0 and Clarion Professional Developer.

Keys and Indexes exist as separate files from the data file. Keys are dynamic—they are automatically updated as the data file changes. The default file extension for a key file is *.K##. Indexes are static—they do not automatically update, but instead require the BUILD statement for updating.

The driver stores records as fixed length. It stores memo fields in a separate file. The memo file defaults to the first eight characters of the File Label plus an extension of .MEM.

Files:

CWC2116.LIB	Windows Export Library (16-bit)
CWC2132.LIB	Windows Export Library (32-bit)
CLC2116.LIB	Windows Static Link Library (16-bit)
CLC2132.LIB	Windows Static Link Library (32-bit)
CWC2116.DLL	Windows Dynamic Link Library (16-bit)
CWC2132.DLL	Windows Dynamic Link Library (32-bit)

Tip: By avoiding the ASCII-only file formats of many other popular PC database application development systems, the Clarion file format provides a more secure means of storing data.

Data Types

BYTE	DECIMAL
SHORT	STRING (255 byte maximum)
LONG	MEMO
REAL	GROUP

Maximum File Specifications:

File Size:	limited only by disk space
Records per File:	4,294,967,295
Record Size:	65,520 bytes
Field Size:	65,520 bytes
Fields per Record:	65,520 bytes
Keys/Indexes per File:	251
Key Size:	245 bytes
Memo fields per File :	1
Memo Field Size:	65,520 bytes
Open Data Files:	Operating system dependent

Driver Strings and SEND functions

Driver strings (the second parameter of the DRIVER attribute) are all preceded by a forward slash character (/). SEND function commands can take two formats—one with an equal sign modifies a switch setting and return the value of the previous switch setting; the other format (without an equal sign) returns the value of the switch.

Driver strings are sent to the file driver when the file is opened. The SEND function sends a command to modify a setting after the file is open. Some driver strings have no effect after the file is open, so the SEND function syntax to modify the setting is not listed. However, the SEND function syntax to return the value of the switch is listed for all driver strings.

RECOVER may not be used as a DRIVER string—you may only use it with the SEND function.

`SEND(file,'RECOVER=n')`

The RECOVER string, when *n* is greater than 0, UNLOCKs a data file, or RELEASEs a held record in order to recover from a system crash.

n represents the number os seconds to wait before invoking the recovery process. When *n* is equal to 1, the recovery process is invoked immediately. When *n* is equal to 0, the recovery process is disarmed.

The SEND function returns a blank string.

To RELEASE a held record, you must read that record into memory. If there are multiple held records, loop through the entire file after SENDing the RECOVER= message to the driver.

`SEND(file,'IGNORESTATUS=on|off')`
`/IGNORESTATUS=on|off` When set *on*, the driver does *not* skip deleted records when accessing the file with GET(), NEXT(), and PREVIOUS() in file order. It also enables a PUT() on a deleted or held record. / IGNORESTATUS requires opening the file in exclusive mode.

`SEND(file,'IGNORESTATUS')`

Returns the **IGNORESTATUS** setting (ON or OFF) in the form of a STRING(3).

`SEND(file,'DELETED')`	For use only with the SEND command, when /IGNORESTATUS is on. Reports the status of the loaded record. If deleted, the return string is "ON;" if not, "OFF."
`SEND(file,'HELD')`	For use only with the SEND command, when /IGNORESTATUS is on. Reports the status of the loaded record. If held, the return string is "ON;" if not, "OFF."

Miscellaneous

- POSITION(file) returns a STRING(4).
- POSITION(key) returns a STRING the size of the key fields + 4 bytes.

Unsupported Functions and Attributes

Record Access: `GET(file, fileptr, len), ADD(file, len), APPEND(file, len)`

The driver does not support variable length records.

File updates: `PUT(file, fileptr, len)`

The driver does not support variable length records.

`EOF(file), BOF(file)`

Although the driver supports these functions, we do not recommend their use. These functions must physically access the files in order to operate, adding considerable overhead. Instead, test the value returned by ERRORCODE() after each sequential access. NEXT() or PREVIOUS() post Error 33 (Record Not Available) if an attempt is made to access a record beyond the end or beginning of the file.

`BUILD(DynamicIndex, expression,filter)`

CLIPPER FILES

The Clipper file driver is compatible with Clipper Summer ‘87 and Clipper 5.0. The default data file extension is *.DBF.

Keys and Indexes exist as separate files from the data file. Keys are dynamic—they automatically update as the data file changes. Indexes are static—they do not automatically update, but instead require the BUILD statement for updating. The default file extension for the index file is *.NTX.

The driver stores records as fixed length. It stores memo fields in a separate file. The memo file name takes the first eight characters of the File Label plus an extension of .DBT.

Files:	**CWCLIP16.LIB**	Windows Export Library (16-bit)
	CWCLIP32.LIB	Windows Export Library (32-bit)
	CLCLIP16.LIB	Windows Static Link Library (16-bit)
	CLCLIP32.LIB	Windows Static Link Library (32-bit)
	CWCLIP16.DLL	Windows Dynamic Link Library (16-bit)
	CWCLIP32.DLL	Windows Dynamic Link Library (32-bit)

Tip: As a popular xBase database application development system, Clipper provides a common file format for many installed business applications and their data files. Use the Clipper driver to access these files in their native format.

Data Types

The xBase file format stores all data as ASCII strings. You may either specify STRING types with declared pictures for each field, or specify native Clarion data types, which the driver converts automatically.

Clipper data type	Clarion data type	STRING w/ picture
Date	DATE	STRING(@D12)
*Numeric	REAL	STRING(@N-_p.d)
*Logical	BYTE	STRING(1)
Character	STRING	STRING
*Memo	MEMO	MEMO

If your application reads and writes to existing files, a pictured STRING will suffice. However, if your application *creates* a Clipper file, you may require additional information for these Clipper types:

- To create a numeric field in the Data Dictionary, choose the REAL data type. In the External Name attribute, specify '*NumericFieldName*=N(*Precision*,*DecimalPlaces*)' where *NumericFieldName* is the name of the field, *Precision* is the precision of the field and *DecimalPlaces* is the number of decimal places.

  If you're hand coding a native Clarion data type, add the NAME attribute using the same syntax. If you're hand coding a STRING with picture, STRING(@N-_9.2), NAME('*Number*'), where *Number* is the field name.

- To create a logical field, using the data dictionary, choose the BYTE data type. There are no special steps; however, see the miscellaneous section for tips on reading the data from the field.

  If you're hand coding a STRING with picture, add the NAME attribute: STRING(1), NAME('*LogFld* = L')**.**

- To create a date field, using the data dictionary, choose the DATE data type, rather than LONG, which you usually use for the TopSpeed or Clarion file formats.

- MEMO field declarations require the a pointer field in the file's record structure. Declare the pointer field as a STRING(10) or a LONG. This field will be stored in the .DBF file containing the offset of the memo in the .DBT file. The MEMO declaration must have a NAME() attribute naming the pointer field. An example file declaration follows:

```
File  FILE, DRIVER('Clipper')
Memo1   MEMO(200),NAME('Notes')
Memo2   MEMO(200),NAME('Text')
Rec     RECORD
Mem1Ptr   LONG,NAME('Notes')
Mem2Ptr   STRING(10),NAME('Text')
          END
        END
```

File Specifications/Maximums

```
File Size:              2,000,000,000 bytes
Records per File:       1,000,000,000 bytes
Record Size:            4,000 bytes (Clipper '87)
                        8,192 bytes (Clipper 5.0)
Field Size
     Character:         254 bytes (Clipper '87)
                        2048 bytes (Clipper 5.0)
     Date:              8 bytes
     Logical:           1 byte
     Numeric:           20 bytes including decimal point
     Memo:              65,520 bytes (see note)
Fields per Record:      255
Keys/Indexes per File:  No Limit
Key Sizes
     Character:         100 bytes
     Numeric, Date:     8 bytes
```

```
Memo fields per File:   Dependent on available memory
Open Files:             Operating system dependent
```

Driver Strings and SEND functions

Driver strings (the second parameter of the DRIVER attribute) are all preceded by a forward slash character (/). SEND function commands can take two formats—one with an equal sign modifies a switch setting and return the value of the previous switch setting; the other format (without an equal sign) returns the value of the switch.

Driver strings are sent to the file driver when the file is opened. The SEND function sends a command to modify a setting after the file is open. Some driver strings have no effect after the file is open, so the SEND function syntax to modify the setting is not listed. However, the SEND function syntax to return the value of the switch is listed for all driver strings.

`/BUFFERS=n`

Specify a value for the number of buffers used to read and write to the file.

The Clipper driver utilizes DOS buffering. The default is three buffers of 1024 bytes each. Increasing the number of buffers will not increase performance when a file is shared by multiple users.

`SEND(file,'BUFFERS')`

Returns the number of buffers in the form of a STRING.

`/RECOVER`
`SEND(file,'RECOVER')`

Equivalent to the Xbase RECALL command, which recovers records marked for deletion. When using the Clipper driver, the DELETE statement flags a record as “inactive.” The driver does not remove the record until the PACK or BUILD command is executed.

/RECOVER is evaluated each time you open the file if you add the driver string to the data dictionary. When the driver recovers the records previously marked for deletion, you must manually rebuild keys and indexes with the BUILD statement.

`SEND(file,'IGNORESTATUS=on|off')`

`/IGNORESTATUS=on|off`	When set *on*, the driver does *not* skip deleted records when accessing the file with GET, NEXT, and PREVIOUS in file order. It also enables a PUT on a deleted or held record. /IGNORESTATUS requires opening the file in exclusive mode.
`SEND(file,'IGNORESTATUS')`	Returns the IGNORESTATUS setting (ON or OFF) in the form of a STRING(3).
`SEND(file,'DELETED')`	For use only with the SEND command, when IGNORESTATUS is on. Reports the status of the current record. If deleted, the return string is "ON;" if not, "OFF."

Unsupported/Modified Functions & Attributes

Memos: `BINARY`

Clipper supports only text memos.

Keys: `NOCASE, OPT`

File: `ENCRYPT, OWNER, RECLAIM`

The Clipper driver cannot read encrypted Clipper files. To reclaim space from deleted records, call PACK(file).

Transaction Processing: `COMMIT(), LOGOUT(), ROLLBACK()`

The Clipper driver does not support any transaction logging.

Record Access: `GET(file, fileptr, len), ADD(file, len)`

Clipper does not support variable length records

File updates: `PUT(file, fileptr, len)`

Clipper does not support variable length records

`EOF(file), BOF(file)`

Although the driver supports these functions, we do not recommend their use. They must physically access the files and add overhead. Instead, test the value returned by ERRORCODE() after each sequential access. NEXT or PREVIOUS post Error 33 (Record Not Available) if an attempt is made to access a record beyond the end or beginning of the file.

`ADD(file)` vs. `APPEND(file)`

The ADD statement tests for duplicate keys before modifying the data file or its associated KEY files. Consequently it is slower than APPEND which performs no checks and does not update KEYs. When adding large amounts of data to a database use APPEND...BUILD in preference to ADD.

`BUILD(key, str)`

When building dynamic indexes, the *str* component may take one of two forms:

```
BUILD(DynNdx, '+Pre:FLD1, -Pre:FLD2')
```

This form specifies the names of the fields on which to build the index. The field names must appear as specified in the fields' NAME() attribute if supplied, or must be the label name. A prefix may be used for compatibility with the Clarion conventions but is ignored.

```
BUILD(DynNdx, 'T[Expression]')
```

This form specifies the type and Expression used to build an index, see the miscellaneous section, below.

```
COPY(file, newname)
```

The COPY() command copies data and memo files using *newname*, which may specify a new file name or directory. Key or index files are copied if the *newname* is a subdirectory specification. To copy an index file to a new file, use a special form of the copy command:

```
COPY(file,'<index>|<newname>')
```

This returns *File Not Found* if an invalid index is passed. The COPY command assumes a default extension of ".NTX" for both the source and the target file names if none is specified. If you require a file name without an extension, terminate the name with a period. Given the file structure:

```
Clar2   FILE,CREATE,DRIVER('Clipper')
NumKey    KEY(Num),DUP
StrKey    KEY(Str1)
StrKey2   KEY(Str2)
AMemo     MEMO(100), NAME('mem')
Record    RECORD
Num         STRING(@n-_9.2)
STR1        STRING(2)
STR2        STRING(2)
Mem         STRING(10)
        . .
```

The following commands copy this file definition to A:

```
COPY(Clar2,'A:\CLAR2')
COPY(Clar2,'StrKey|A:\STRKEY')
COPY(Clar2,'StrKey2|A:\STRKEY2')
COPY(Clar2,'NumKey|A:\NUMKEY')
```

After these calls, the following files would exist on drive A: CLAR2.DBF, CLAR2.DBT, STRKEY.NTX, STRKEY2.NTX, and NUMKEY.NTX.

`DELETE(file)`

When the driver deletes a record from a Clipper database, the record is not physically removed, instead the driver marks it inactive. Memo fields are not physically removed from the memo file, however they cannot be retrieved if they refer to an inactive record. Key values *are* removed from the index files. To remove records and memo fields permanently, execute a PACK(file).

Tip: To those programmers familiar with Clipper, this driver processes deleted records consistent with the way Clipper processes them after the SET DELETED ON command is issued. Records marked for deletion are ignored from processing by executable code statements, but remain in the data file.

`HOLD(file), HOLD(file, timeout)`

Clipper performs record locking by locking the entire record within the data file. This prevents read access to other processes. Therefore we recommend minimizing the amount of time for which a record is held.

`POINTER(file), POINTER(key)`

There is no distinction between file pointers and key pointers; they both contain the same value for any given record.

`RECORDS(file), RECORDS(key)`

Under Clipper, the RECORDS() function reports the same number of records for the data file and its keys and indexes. Usually there will be no difference in the number of records *unless* the INDEX is out of date. Because the DELETE statement does not physically remove records, *the number of records reported by the* RECORDS() *function includes inactive records*. Exercise care when using this function.

`RENAME(file, newname)`

The RENAME command copies the data and memo files using *newname*, which may specify a new file name or directory path. Key and index files must be renamed using the same syntax as the COPY command, above.

- POSITION(file) returns a STRING(12).
- POSITION(key) returns a STRING the size of the key fields + 4 bytes.
- `BUILD(DynamicIndex, expression,filter)` is not supported.

Miscellaneous

- Clipper allows a maximum of 254 characters to a character field.
- Clipper allows a logical field to accept one of nine possible values (y,Y,n,N,t,T,f,F or a space character). The space character is neither true nor false. When using a logical field from a preexisting database in a logical expression, account for all these possibilities. Remember that when a STRING field is used as an expression, it is true if it contains any data and false if it is equal to zero or blank. Therefore, to evaluate a Logical field's truth, the expression should be true if the field contains any of the "true" characters (T,t,Y, or y). For example, if a Logical field were used to specify a product as taxable or nontaxable, the expression to evaluate its truth would be:

  (If Condition):

  ```
          Taxable='T' OR Taxable='t' OR Taxable='Y' OR Taxable='y'
  ```
- Clarion for Windows supports MEMO fields up to a maximum of 64K. If you have an existing file which includes a memo greater than 64K, you can use the file but not modify the large MEMOs.
- You can determine when your application encounters a large MEMO by detecting when the memo pointer variable is non-blank, but the memo appears to be blank. Error 47 (Bad Record Declaration) is posted. If you attempt to update such a record, any modification to

the MEMO field is ignored.

- Clipper supports a maximum of 10 characters in a field name. If you require more, use an External Name with 10 characters or less.

- Clipper supports the use of expressions to define keys. Within the Dictionary Editor, you can place the expression in the external name field in the *Key Properties* dialog. The general format of the external name is :

  ```
  'FileName=T[Expression]'
  ```

  Where *FileName* represents the name of the index file (which can contain a path and file extension), and *T* represents the type of the index. Valid types are: C = character, D = date, and N = numeric. If the type is D or N then *Expression* can name only one field.

  The expression may refer to multiple fields in the record, and contain xBase functions. Square brackets must enclose the expression. The currently supported functions appear below. If the driver encounters an unsupported Xbase function in a preexisting file, it posts error 76 'Invalid Index String' when the file is opened for keys and static indexes.

  String expressions may use the '+' operator to concatenate multiple string arguments. Numeric expressions use the '+' or '-' operators with their conventional meanings. The maximum length of a Clipper expression is 250 characters.

Supported xBase commands

`ALLTRIN(string)`	Removes leading and trailing spaces.
`CTOD(string)`	Converts a string key to a date. The *string* format mm/dd/yy; the result takes the form 'yyyymmdd'. The yyyy element of the date defaults to the twentieth century. An invalid date results in a key containing blanks.
`DELETED()`	Returns TRUE if the record is deleted.
`DESCEND(string|date|numeric)`	Inverts the argument, and creates descending Clipper indexes.
`DTOC(date)`	Converts a date key to string format 'mm/dd/yy'
`DTOS(date)`	Converts a date key to string format 'yyyymmdd'
`FIXED(float)`	Converts a float key to a numeric.
`FLOAT(numeric)`	Converts a numeric key to a float.

`IIF(bool,val1,val2)`	Returns val1 if the first parameter is TRUE, otherwise returns val2.
`LEFT(string, n)`	Returns the leftmost *n* characters of the string key as a string of length *n*.
`RIGHT(string, n)`	Returns the rightmost *n* characters of the string key as a string of length *n*.
`RTRIM(string)`	Removes spaces from the right of a string.
`STR(numeric [,length[, decimal places]])`	converts a numeric to a string. The length of the string and the number of decimal places are optional. The default string length is 10, and the number of decimal places is 0.
`SUBSTR(string,offset,n)`	Returns a substring of the *string* key starting at *offset* and of *n* characters in length.
`TRIM(string)`	Removes spaces from the right of a string (identical to RTRIM).
`UPPER(string)`	Converts a string key to upper case.
`VAL(string)`	Converts a string key to a numeric.

dBASE III FILES

The dBase3 file driver is compatible with dBase III. The default data file extension is *.DBF.

Keys and Indexes exist as separate files from the data file. Keys are dynamic—they automatically update as the data file changes. Indexes are static—they do not automatically update, but instead require the BUILD statement for updating. The default file extension for the index file is *.NDX.

The driver stores records as fixed length. It stores memo fields in a separate file. The memo file name takes the first eight characters of the File Label plus an extension of .DBT.

Files:

CWDB316.LIB	Windows Export Library (16-bit)
CWDB332.LIB	Windows Export Library (32-bit)
CLDB316.LIB	Windows Static Link Library (16-bit)
CLDB332.LIB	Windows Static Link Library (32-bit)
CWDB316.DLL	Windows Dynamic Link Library (16-bit)
CWDB332.DLL	Windows Dynamic Link Library (32-bit)

Tip: dBase III is probably the most common file format for PC database applications. These days, even desktop publishing programs can import dBase III compatible .DBF files. If the main task of your application is to export data files for other applications about which you know nothing, you should consider this format.

Data Types

The xBase file format stores all data as ASCII strings. You may either specify STRING types with declared pictures for each field, or specify native Clarion types, which the driver converts automatically.

dBase data type	Clarion data type	STRING w/ picture
Date	DATE	STRING(@D12)
*Numeric	REAL	STRING(@N-_p.d)
*Logical	BYTE	STRING(1)
Character	STRING	STRING
*Memo	MEMO	MEMO

If your application reads and writes to existing files, a pictured STRING will suffice. However, if your application *creates* a dBase III file, you may require additional information for these dBase III types:

❖ To create a numeric field in the Data Dictionary, choose the REAL data type. In the External Name attribute, specify '*NumericFieldName*=N(*Precision*,*DecimalPlaces*)' where *NumericFieldName* is the name of the field, *Precision* is the precision of the field and *DecimalPlaces* is the number of decimal places.

  If you're hand coding a native Clarion data type, add the NAME attribute using the same syntax. If you're hand coding a STRING with picture, STRING(@N-_9.2), NAME('*Number*'), where *Number* is the field name.

❖ To create a logical field, using the data dictionary, choose the BYTE data type. There are no special steps; however, see the miscellaneous section for tips on reading the data from the field.

  If you're hand coding a STRING with picture, add the NAME attribute: STRING(1), NAME('*LogFld* = L')**.**

❖ To create a date field, using the data dictionary, choose the DATE data type, rather than LONG, which you usually use for the TopSpeed or Clarion file formats.

❖ MEMO field declarations require the a pointer field in the file's record structure. Declare the pointer field as a STRING(10) or a LONG. This field will be stored in the .DBF file containing the offset of the memo in the .DBT file. The MEMO declaration must have a NAME() attribute naming the pointer field. An example file declaration follows:

```
File  FILE, DRIVER('dBase3')
Memo1   MEMO(200),NAME('Notes')
Memo2   MEMO(200),NAME('Text')
Rec     RECORD
Mem1Ptr   LONG,NAME('Notes')
Mem2Ptr   STRING(10),NAME('Text')
        END
      END
```

File Specifications/Maximums

```
File Size:              2,000,000,000 bytes
Records per File:       1,000,000,000 bytes
Record Size:            4,000 bytes
Field Size
     Character:         254 bytes
     Date:              8 bytes
     Logical:           1 byte
     Numeric:           20 bytes including decimal point
     Memo:              64K (see note)
Fields per Record:      255
Keys/Indexes per File:  No Limit
Key Sizes
     Character:         100 bytes
     Numeric, Date:     8 bytes
Memo fields per File:   Dependent on available memory
Open Files:             Operating system dependent
```

Driver Strings and SEND functions

Driver strings (the second parameter of the DRIVER attribute) are all preceded by a forward slash character (/). SEND function commands can take two formats—one with an equal sign modifies a switch setting and return the value of the previous switch setting; the other format (without an equal sign) returns the value of the switch.

Driver strings are sent to the file driver when the file is opened. The SEND function sends a command to modify a setting after the file is open. Some driver strings have no effect after the file is open, so the SEND function syntax to modify the setting is not listed. However, the SEND function syntax to return the value of the switch is listed for all driver strings.

`/BUFFERS=n` Specify a value for the number of buffers used to read and write to the file.

The dBase III driver utilizes DOS buffering. The default is three buffers of 1024 bytes each. Increasing the number of buffers will not increase performance when a file is shared by multiple users.

`SEND(file,'BUFFERS')`

Returns the number of buffers in the form of a STRING.

`/RECOVER`

`SEND(file,'RECOVER')`

Equivalent to the Xbase RECALL command, which recovers records marked for deletion. When using the dBase III driver, the DELETE statement flags a record as "inactive." The driver does not remove the record until the PACK command is executed.

/RECOVER is evaluated each time you open the file if you add the driver string to the data dictionary. When the driver recovers the records previously marked for deletion, you must manually rebuild keys and indexes with the BUILD statement.

`SEND(file,'IGNORESTATUS=on|off')`
`/IGNORESTATUS=on|off`

When set *on*, the driver does *not* skip deleted records when accessing the file with GET, NEXT, and PREVIOUS in file order. It also enables a PUT on a deleted or held record. /IGNORESTATUS requires opening the file in exclusive mode.

`SEND(file,'IGNORESTATUS')`

Returns the IGNORESTATUS setting (ON or OFF) in the form of a STRING(3).

`SEND(file,'DELETED')`

For use only with the SEND command, when IGNORESTATUS is on. Reports the status of the current record. If deleted, the return string is "ON;" if not, "OFF."

`/OMNIS` Specifies OMNIS file header and file delimiter compatibility.

Unsupported/Modified Functions & Attributes

Memos: `BINARY`

dBase III supports only text memos.

Keys: `DUP, NOCASE, OPT, ascending|descending`

File: `ENCRYPT, OWNER, RECLAIM`

The dBase III driver cannot read encrypted dBase III files. To reclaim space from deleted records, call PACK(file).

Transaction Processing: `COMMIT(), LOGOUT(), ROLLBACK()`

The dBase III driver does not support any transaction logging.

Record Access: `GET(file, fileptr, len), ADD(file, len)`

dBase III does not support variable length records

File updates: `PUT(file, fileptr, len)`

dBase III does not support variable length records

`EOF(file), BOF(file)`

Although the driver supports these functions, we do not recommend their use. They must physically access the files and add overhead. Instead, test the value returned by ERRORCODE() after each sequential access. NEXT or PREVIOUS post Error 33 (Record Not Available) if an attempt is made to access a record beyond the end or beginning of the file.

`ADD(file)` vs. `APPEND(file)`

The ADD statement tests for duplicate keys before modifying the data file or its associated KEY files. Consequently it is slower than APPEND which performs no checks and does not update KEYs. When adding large amounts of data to a database use APPEND...BUILD in preference to ADD.

`BUILD(key, str)`

When building dynamic indexes, the *str* component may take one of two forms:

```
BUILD(DynNdx, '+Pre:FLD1, -Pre:FLD2')
```

This form specifies the names of the fields on which to build the index. The field names must appear as specified in the fields' NAME() attribute if supplied, or must be the label name. A prefix may be used for compatibility with the Clarion conventions but is ignored.

```
BUILD(DynNdx, 'T[Expression]')
```

This form specifies the type and Expression used to build an index, see the miscellaneous section, below.

```
COPY(file, newname)
```

The COPY() command copies data and memo files using *newname*, which may specify a new file name or directory. Key or index files are copied if the *newname* is a subdirectory specification. To copy an index file to a new file, use a special form of the copy command:

```
COPY(file,'<index>|<newname>')
```

This returns *File Not Found* if an invalid index is passed. The COPY command assumes a default extension of ".NDX" for both the source and the target file names if none is specified. If you require a file name without an extension, terminate the name with a period. Given the file structure:

```
Clar2 FILE,CREATE,DRIVER('dBase3')
NumKey  KEY(Num),DUP
StrKey  KEY(Str1)
StrKey2 KEY(Str2)
AMemo   MEMO(100), NAME('mem')
Record    RECORD
Num       STRING(@n_9.2)
STR1      STRING(2)
STR2      STRING(2)
  Mem     STRING(10)
      .   .
```

The following commands copy this file definition to A:

```
COPY(Clar2,'A:\CLAR2')
COPY(Clar2,'StrKey|A:\STRKEY')
COPY(Clar2,'StrKey2|A:\STRKEY2')
COPY(Clar2,'NumKey|A:\NUMKEY')
```

After these calls, the following files would exist on drive A: CLAR2.DBF, CLAR2.DBT, STRKEY.NDX, STRKEY2.NDX, and NUMKEY.NDX.

```
DELETE(file)
```

When the driver deletes a record from a dBase III database, the record is not physically removed, instead the driver marks it inactive. Memo fields are not physically removed from the memo file, however they cannot be retrieved if they refer to an inactive record. Key values *are* removed from the index files. To remove records and memo fields permanently, execute a PACK(file).

Tip: To those programmers familiar with dBase III, this driver processes deleted records consistent with the way dBase III processes them after the SET DELETED ON command is issued. Records marked for deletion are ignored from processing by executable code statements, but remain in the data file.

`HOLD(file), HOLD(file, timeout)`

dBase III performs record locking by locking the entire record within the data file. This prevents read access to other processes. Therefore we recommend minimizing the amount of time for which a record is held.

`POINTER(file), POINTER(key)`

There is no distinction between file pointers and key pointers; they both contain the same value for any given record.

`RECORDS(file), RECORDS(key)`

Under dBase III the RECORDS() function reports the same number of records for the data file and its keys and indexes. Usually there will be no difference in the number of records *unless* the INDEX is out of date. Because the DELETE statement does not physically remove records, *the number of records reported by the* RECORDS() *function includes inactive records*. Exercise care when using this function.

`RENAME(file, newname)`

The RENAME command copies the data and memo files using *newname*, which may specify a new file name or directory path. Key and index files must be renamed using the same syntax as the COPY command, above.

❖ `BUILD(DynamicIndex, expression,filter)` is not supported.

Miscellaneous

- dBase III allows a maximum of 254 characters to a character field.
- dBase III allows a logical field to accept one of nine possible values (y,Y,n,N,t,T,f,F or a space character). The space character is neither true nor false. When using a logical field from a preexisting database in a logical expression, account for all these possibilities. Remember that when a STRING field is used as an expression, it is true if it contains any data and false if it is equal to zero or blank. Therefore, to evaluate a Logical field's truth, the expression should be true if the field contains any of the "true" characters (T,t,Y, or y). For example, if a Logical field were used to specify a product as taxable or nontaxable, the expression to evaluate its truth would be:

  (If Condition):

  ```
  Taxable='T' OR Taxable='t' OR Taxable='Y' OR Taxable='y'
  ```
- Clarion for Windows supports MEMO fields up to a maximum of 64K. If you have an existing file which includes a memo greater than 64K, you can use the file but not modify the large MEMOs.
- You can determine when your application encounters a large MEMO by detecting when the memo pointer variable is non-blank, but the memo appears to be blank. Error 47 (Bad Record Declaration) is posted, and any modification to the MEMO field is ignored.
- dBase III supports a maximum of 10 characters in a field name. If you require more, use an External Name with 10 characters or less.
- dBase III supports the use of expressions to define keys. Within the Dictionary Editor, you can place the expression in the external name field in the *Key Properties* dialog. The general format of the external name is :

  ```
  'FileName=T[Expression]'
  ```

  Where *FileName* represents the name of the index file (which can contain a path and file extension), and *T* represents the type of the index. Valid types are: C = character, D = date, and N = numeric. If the type is D or N then *Expression* can name only one field.

  The expression may refer to multiple fields in the record, and contain xBase functions. Square brackets must enclose the expression. The currently supported functions appear below. If the driver encounters an unsupported Xbase function in a preexisting file, it posts error 76 'Invalid Index String' when the file is opened for keys and static indexes.

String expressions may use the ‘+’ operator to concatenate multiple string arguments. Numeric expressions use the ‘+’ or ‘-’ operators with their conventional meanings. The maximum length of a dBase III expression is 250 characters.

Supported xBase commands

`ALLTRIN(string)`	Removes leading and trailing spaces.
`CTOD(string)`	Converts a string key to a date. The *string* format mm/dd/yy; the result takes the form ‘yyyymmdd’. The yyyy element of the date defaults to the twentieth century. An invalid date results in a key containing blanks.
`DELETED()`	Returns TRUE if the record is deleted.
`DTOC(date)`	Converts a date key to string format ‘mm/dd/yy.’
`DTOS(date)`	Converts a date key to string format ‘yyyymmdd.’
`FIXED(float)`	Converts a float key to a numeric.
`FLOAT(numeric)`	Converts a numeric key to a float.
`IIF(bool,val1,val2)`	Returns val1 if the first parameter is TRUE, otherwise returns val2.
`LEFT(string, n)`	Returns the leftmost *n* characters of the string key as a string of length *n*.
`RIGHT(string, n)`	Returns the rightmost *n* characters of the string key as a string of length *n*.
`RTRIM(string)`	Removes spaces from the right of a string.
`STR(numeric [,length [, decimal places] ] )`	Converts a numeric to a string. The length of the string and the number of decimal places are optional. The default string length is 10, and the number of decimal places is 0.
`SUBSTR(string,offset,n)`	Returns a substring of the *string* key starting at *offset* and of *n* characters in length.
`TRIM(string)`	Removes spaces from the right of a string (identical to RTRIM).
`UPPER(string)`	Converts a string key to upper case.
`VAL(string)`	Converts a string key to a numeric.

- POSITION(file) returns a STRING(12).
- POSITION(key) returns a STRING the size of the key fields + 4 bytes.

dBASE IV FILES

The dBase4 file driver is compatible with dBase IV. The default data file extension is *.DBF.

Keys and Indexes exist as separate files from the data file. Keys are dynamic—they automatically update as the data file changes. Indexes are static—they do not automatically update, but instead require the BUILD statement for updating. The default file extension for the index file is *.NDX.

dBase IV supports multiple index files, whose extension is *.MDX. The miscellaneous section describes procedures for using .MDX files.

The driver stores records as fixed length. It stores memo fields in a separate file. The memo file name takes the first eight characters of the File Label plus an extension of .DBT.

Files:	**CWDB416.LIB**	Windows Export Library (16-bit)
	CWDB432.LIB	Windows Export Library (32-bit)
	CLDB416.LIB	Windows Static Link Library (16-bit)
	CLDB432.LIB	Windows Static Link Library (32-bit)
	CWDB416.DLL	Windows Dynamic Link Library (16-bit)
	CWDB432.DLL	Windows Dynamic Link Library (32-bit)

Tip: dBase IV was never as widely adopted as dBase III. Choose this driver only when you must share data with an end-user using dBase IV.

Data Types

The xBase file format stores all data as ASCII strings. You may either specify STRING types with declared pictures for each field, or specify native Clarion types, which the driver converts automatically.

dBase data type	Clarion data type	STRING w/ picture
Date	DATE	STRING(@D12)
*Numeric	REAL	STRING(@N-_p.d)
*Logical	BYTE	STRING(1)
Character	STRING	STRING
*Memo	MEMO	MEMO

If your application reads and writes to existing files, a pictured STRING will suffice. However, if your application *creates* a dBase IV file, you may require additional information for these dBase IV types:

- To create a numeric field in the Data Dictionary, choose the REAL data type. In the External Name attribute, specify '*NumericFieldName*=N(*Precision*,*DecimalPlaces*)' where *NumericFieldName* is the name of the field, *Precision* is the precision of the field and *DecimalPlaces* is the number of decimal places.

  If you're hand coding a native Clarion data type, add the NAME attribute using the same syntax. If you're hand coding a STRING with picture, STRING(@N-_9.2), NAME('*Number*'), where *Number* is the field name.

- To create a logical field, using the data dictionary, choose the BYTE data type. There are no special steps; however, see the miscellaneous section for tips on reading the data from the field.

  If you're hand coding a STRING with picture, add the NAME attribute: STRING(1), NAME('*LogFld* = L')**.**

- To create a date field, using the data dictionary, choose the DATE data type, rather than LONG, which you usually use for the TopSpeed or Clarion file formats.

- MEMO field declarations require the a pointer field in the file's record structure. Declare the pointer field as a STRING(10) or a LONG. This field will be stored in the .DBF file containing the offset of the memo in the .DBT file. The MEMO declaration must have a NAME() attribute naming the pointer field. An example file declaration follows:

```
File  FILE, DRIVER('dBase4')
Memo1   MEMO(200),NAME('Notes')
Memo2   MEMO(200),NAME('Text')
Rec     RECORD
Mem1Ptr   LONG,NAME('Notes')
Mem2Ptr   STRING(10),NAME('Text')
        END
      END
```

File Specifications/Maximums

File Size:	2,000,000,000 bytes
Records per File:	1,000,000,000 bytes
Record Size:	4,000 bytes
Field Size	
Character:	254 bytes
Date:	8 bytes
Logical:	1 byte
Numeric:	20 bytes including decimal point
Float:	20 bytes including decimal point
Memo:	64K (see note)
Fields per Record:	255
Keys/Indexes per File:	
.NDX:	No Limit
.MDX	47 tags per .MDX files
Key Sizes	
Character:	100 bytes
Numeric, Date:	8 bytes
Memo fields per File:	Dependent on available memory
Open Files:	Operating system dependent

Driver Strings and SEND functions

Driver strings (the second parameter of the DRIVER attribute) are all preceded by a forward slash character (/). SEND function commands can take two formats—one with an equal sign modifies a switch setting and return the value of the previous switch setting; the other format (without an equal sign) returns the value of the switch.

Driver strings are sent to the file driver when the file is opened. The SEND function sends a command to modify a setting after the file is open. Some driver strings have no effect after the file is open, so the SEND function syntax to modify the setting is not listed. However, the SEND function syntax to return the value of the switch is listed for all driver strings.

`/BUFFERS=n`	Specify a value for the number of buffers used to read and write to the file. The dBase IV driver utilizes DOS buffering. The default is three buffers of 1024 bytes each. Increasing the number of buffers will not increase performance when a file is shared by multiple users.
`SEND(file,'BUFFERS')`	Returns the number of buffers in the form of a STRING.

`/RECOVER`
`SEND(file,'RECOVER')`
	Equivalent to the Xbase RECALL command, which recovers records marked for deletion. When using the dBase IV driver, the DELETE statement flags a record as "inactive." The driver does not remove the record until the PACK command is executed.
	
	/RECOVER is evaluated each time you open the file if you add the driver string to the data dictionary. When the driver recovers the records previously marked for deletion, you must manually rebuild keys and indexes with the BUILD statement.

`SEND(file,'IGNORESTATUS=on|off')`	

`/IGNORESTATUS=on|off`	When set *on*, the driver does *not* skip deleted records when accessing the file with GET, NEXT, and PREVIOUS in file order. It also enables a PUT on a deleted or held record. /IGNORESTATUS requires opening the file in exclusive mode.
`SEND(file,'IGNORESTATUS')`	Returns the IGNORESTATUS setting (ON or OFF) in the form of a STRING(3).
`SEND(file,'DELETED')`	For use only with the SEND command, when IGNORESTATUS is on. Reports the status of the current record. If deleted, the return string is “ON;” if not, “OFF.”

Unsupported/Modified Functions & Attributes

Memos: `BINARY`

dBase IV supports only text memos.

Keys: `OPT`

File: `ENCRYPT, OWNER, RECLAIM`

The dBase IV driver cannot read encrypted dBase IV files. To reclaim space from deleted records, call PACK(file).

Transaction Processing: `COMMIT(), LOGOUT(), ROLLBACK()`

The dBase IV driver does not support any transaction logging.

Record Access: `GET(file, fileptr, len), ADD(file, len)`

dBase IV does not support variable length records

File updates:

`PUT(file, fileptr, len)`

dBase IV does not support variable length records

`EOF(file), BOF(file)`

Although the driver supports these functions, we do not recommend their use. They must physically access the files and add overhead. Instead, test the value returned by ERRORCODE() after each sequential access. NEXT or PREVIOUS post Error 33 (Record Not Available) if an attempt is made to access a record beyond the end or beginning of the file.

`ADD(file)` vs. `APPEND(file)`

The ADD statement tests for duplicate keys before modifying the data file or its associated KEY files. Consequently it is slower than APPEND which performs no checks and does not update KEYs. When adding large amounts of data to a database use APPEND...BUILD in preference to ADD.

`BUILD(key, str)`

When building dynamic indexes, the *str* component may take one of two forms:

```
BUILD(DynNdx, '+Pre:FLD1, -Pre:FLD2')
```

This form specifies the names of the fields on which to build the index. The field names must appear as specified in the fields' NAME() attribute if supplied, or must be the label name. A prefix may be used for compatibility with the Clarion conventions but is ignored.

```
BUILD(DynNdx, 'T[Expression]')
```

This form specifies the type and Expression used to build an index, see the miscellaneous section, below.

```
COPY(file, newname)
```

The COPY() command copies data and memo files using *newname*, which may specify a new file name or directory. Key or index files are copied if the *newname* is a subdirectory specification. To copy an index file to a new file, use a special form of the copy command:

```
COPY(file,'<index>|<newname>')
```

This returns *File Not Found* if an invalid index is passed. The COPY command assumes a default extension of ".NDX" for both the source and the target file names if none is specified. If you require a file name without an extension, terminate the name with a period. Given the file structure:

```
Clar2   FILE,CREATE,DRIVER('dBase3')
NumKey    KEY(Num),DUP
StrKey    KEY(Str1)
StrKey2   KEY(Str2)
AMemo     MEMO(100), NAME('mem')
Record    RECORD
Num         STRING(@n-_9.2)
STR1        STRING(2)
STR2        STRING(2)
Mem         STRING(10)
        . .
```

The following commands copy this file definition to A:

```
COPY(Clar2,'A:\CLAR2')
COPY(Clar2,'StrKey|A:\STRKEY')
COPY(Clar2,'StrKey2|A:\STRKEY2')
COPY(Clar2,'NumKey|A:\NUMKEY')
```

After these calls, the following files would exist on drive A: CLAR2.DBF, CLAR2.DBT, STRKEY.NDX, STRKEY2.NDX, and NUMKEY.NDX.

```
DELETE(file)
```

When the driver deletes a record from a dBase IV database, the record is not physically removed, instead the driver marks it inactive. Memo fields are not physically removed from the memo file, however they cannot be retrieved if they refer to an inactive record. Key values *are* removed from the index files. To remove records and memo fields permanently, execute a PACK(file).

Tip: To those programmers familiar with dBase IV, this driver processes deleted records in a consistent manner with the way dBase IV processes them after the SET DELETED ON command is issued. Records marked for deletion are ignored from processing by executable code statements, but remain in the data file.

`HOLD(file), HOLD(file, timeout)`

dBase IV performs record locking by locking the entire record within the data file. This prevents read access to other processes. Therefore we recommend minimizing the amount of time for which a record is held.

`POINTER(file), POINTER(key)`

There is no distinction between file pointers and key pointers; they both contain the same value for any given record.

`RECORDS(file), RECORDS(key)`

Under dBase IV the RECORDS() function reports the same number of records for the data file and its keys and indexes. Usually there will be no difference in the number of records *unless* the INDEX is out of date. Because the DELETE statement does not physically remove records, *the number of records reported by the* RECORDS() *function includes inactive records*. Exercise care when using this function.

`RENAME(file, newname)`

The RENAME command copies the data and memo files using *newname*, which may specify a new file name or directory path. Key and index files must be renamed using the same syntax as the COPY command, above.

- POSITION(file) returns a STRING(12).
- POSITION(key) returns a STRING containing the size of the key fields + 4 bytes.
- `BUILD(DynamicIndex, expression,filter)` is not supported.

Miscellaneous

- dBase IV allows a maximum of 254 characters to a character field.
- dBase IV allows a logical field to accept one of 11 possible values (1,0,y,Y,n,N,t,T,f,F or a space character). The space character is neither true nor false. When using a logical field from a preexisting database in a logical expression, account for all these possibilities. Remember that when a STRING field is used as an expression, it is true if it contains any data and false if it is equal to zero or blank. Therefore, to evaluate a Logical field's truth, the expression should be true if the field contains any of the "true" characters (T,t,Y, or y). For example, if a Logical field were used to specify a product as taxable or nontaxable, the expression to evaluate its truth would be:

  (If Condition):

```
Taxable='1' OR Taxable='T' OR Taxable='t' OR Taxable='Y' OR Taxable='y'
```

- Clarion for Windows supports MEMO fields up to a maximum of 64K. If you have an existing file which includes a memo greater than 64K, you can use the file but not modify the large MEMOs.
- You can determine when your application encounters a large MEMO by detecting when the memo pointer variable is non-blank, but the memo appears to be blank. Error 47 (Bad Record Declaration) is posted, and any modification to the MEMO field is ignored.
- dBase IV supports a maximum of 10 characters in a field name. If you require more, use an External Name with 10 characters or less.
- dBase IV supports the use of expressions to define keys. Within the Dictionary Editor, you can place the expression in the external name field in the *Key Properties* dialog. The general format of the external name is :

```
'FileName=T[Expression]'
```

  Where *FileName* represents the name of the index file (which can contain a path and file extension), and *T* represents the type of the index. Valid types are: C = character, D = date, and N = numeric. If the type is D or N then *Expression* can name only one field.

- Multiple-index (.MDX) files require the NAME() attribute on a KEY or INDEX to specify the storage type of the key and any expression used to generate the key values. The general format of the NAME() attribute on a KEY or INDEX is:

```
NAME('TagName|FileName[PageSize]=T[Expression],FOR[Expression]')
```

  The following documents the parameters for the NAME() attribute:

`TagName`	Specifies the name of an index tag within a multiple index file. If omitted the driver creates a dBase IV style .NDX file using the name specified in FileName.
`FileName`	Specifies the name of the index file, which may contain a path and extension.
`PageSize`	Specifies that when creating a .MDX file, (if a TagName is specified), a number in the range 2-32 specifying the number of 512-byte blocks in each index page. This value is only used when creating the file. If you specify multiple values via declarations for different tags in the same .MDX file, the largest value will be selected. The default value is 2.
`T`	Specifies the type of the index,. Legal types are C = character, D = date, N = numeric. If the type is D or N then *Expression* may name *only one* field.
`Expression`	Specifies an expression to generate the index. It may refer to multiple fields, and invoke multiple xBase functions. The functions currently supported are listed below. Square brackets must enclose the expression.

Elements of the NAME() attribute may be omitted from the right. When specifying an Expression, you must also specify the type and name. If the Expression is omitted, the driver determines the Expression from the key fields when the file is created, or from the index file when opened.

If the type is omitted, the driver determines the index type from the first key component when the file is created, or from the index file when opened.

If the NAME() attribute is omitted altogether, the index file name is determined from the key label. The path defaults to the same location as the .DBF.

Tag names are limited to 9 characters in length. If the supplied name is too long it is automatically truncated.

Specify all field names in the NAME() attribute without a prefix.

- dBase IV additionally supports the use of the Xbase FOR statement in expressions to define keys. The expressions supported in the FOR condition must be a simple condition of the form:

  ```
  expression comparison_op expression
  ```

  comparison_op may be one of the following: <, <=, =<, <>, =, =>, >= or >.

The expression may refer to multiple fields in the record, and contain xBase functions. Square brackets must enclose the expression. The currently supported functions appear below. If the driver encounters an unsupported Xbase function in a preexisting file, it posts error 76 ‘Invalid Index String’ when the file is opened for keys and static indexes.

String expressions may use the ‘+’ operator to concatenate multiple string arguments. Numeric expressions use the ‘+’ or ‘-’ operators with their conventional meanings. The maximum length of a dBase IV expression is 250 characters.

Supported xBase commands

`ALLTRIM(string)`	Removes leading and trailing spaces.
`CTOD(string)`	Converts a string key to a date. The *string* format mm/dd/yy; the result takes the form 'yyyymmdd'. The yyyy element of the date defaults to the twentieth century. An invalid date results in a key containing blanks.
`DELETED()`	Returns TRUE if the record is deleted.
`DTOC(date)`	Converts a date key to string format 'mm/dd/yy.'
`DTOS(date)`	Converts a date key to string format 'yyyymmdd.'
`FIXED(float)`	Converts a float key to a numeric.
`FLOAT(numeric)`	Converts a numeric key to a float.
`IIF(bool,val1,val2)`	Returns val1 if the first parameter is TRUE, otherwise returns val2.
`LEFT(string, n)`	Returns the leftmost *n* characters of the string key as a string of length *n*.
`RIGHT(string, n)`	Returns the rightmost *n* characters of the string key as a string of length *n*.
`RTRIM(string)`	Removes spaces from the right of a string.
`STR(numeric [,length[, decimal places]])`	converts a numeric to a string. The length of the string and the number of decimal places are optional. The default string length is 10, and the number of decimal places is 0.
`SUBSTR(string,offset,n)`	Returns a substring of the *string* key starting at *offset* and of *n* characters in length.
`TRIM(string)`	Removes spaces from the right of a string (identical to RTRIM).
`UPPER(string)`	Converts a string key to upper case.
`VAL(string)`	Converts a string key to a numeric.

DOS FILES

The DOS file driver reads and writes any binary, byte-addressable files. Neither fields nor records are delimited. When reading a record, the driver reads the number of bytes defined in the file's RECORD structure, unless a length parameter is specified in the GET statement.

The DOS driver supports the length parameter for the ADD, APPEND, GET, and PUT statements; this allows for variable length records in a DOS file.

The POINTER function returns the relative byte position within the file of the beginning of the last record accessed by an ADD, APPEND, GET, or NEXT statement.

This file driver performs forward sequential processing *only*. No key or transaction processing functions are supported, and the PREVIOUS statement is not supported.

Tip: Due to its limitations, the main function of this driver is as a disk editor for binary files.

Files:	**CWDOS16.LIB**	Windows Export Library (16-bit)
	CWDOS32.LIB	Windows Export Library (32-bit)
	CLDOS16.LIB	Windows Static Link Library (16-bit)
	CLDOS32.LIB	Windows Static Link Library (32-bit)
	CWDOS16.DLL	Windows Dynamic Link Library (16-bit)
	CWDOS32.DLL	Windows Dynamic Link Library (32-bit)

Data Types

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

File Specifications/Maximums

```
File Size           :       4,294,967,295
Records per File    :       4,294,967,295
Record Size         :       64K
Field Size          :       64K
Fields per Record   :       64K
Keys/Indexes per File:      n/a
Key Size            :       n/a
Memo fields per File:       n/a
Memo Field Size     :       n/a
Open Data Files     :       Operating system dependent
```

Driver Strings and SEND functions

Driver strings (the second parameter of the DRIVER attribute) are all preceded by a forward slash character (/). SEND function commands can take two formats—one with an equal sign modifies a switch setting and return the value of the previous switch setting; the other format (without an equal sign) returns the value of the switch.

Driver strings are sent to the file driver when the file is opened. The SEND function sends a command to modify a setting after the file is open. Some driver strings have no effect after the file is open, so the SEND function syntax to modify the setting is not listed. However, the SEND function syntax to return the value of the switch is listed for all driver strings.

`/FILEBUFFERS=n`

Specifies a value for the number of buffers used to read and write to the file.

The DOS driver allocates internal buffers of 512 bytes, or the size of your record, whichever is larger, to store the retrieved file. The default number of buffers is 2 for files opened denying write access to other users, and 1 for all other open modes. Use the optional driver string to increase the buffers should you find access to records is slow.

`SEND(file, ‘FILEBUFFERS’)`

Returns the value of the number of buffers in STRING format.

`/QUICKSCAN=on|off`

`SEND(file,'QUICKSCAN=on|off')`

The DOS driver reads a buffer at a time (not a record), allowing for fast access. In a multi-user environment these buffers are not 100% trustworthy for subsequent access, because another user may change the database between accesses. As a safeguard, the driver rereads the buffers before each record access. To *disable* the reread, set QUICKSCAN to ON. The default is ON for files opened denying write access to other users, and OFF for all other open modes.

`SEND(file,'QUICKSCAN')`

Returns the Quickscan setting (ON or OFF) in the form of a STRING(3).

Unsupported Functions and Attributes

Memos:	`NOMEMO()`
Transaction Processing:	`COMMIT(), LOGOUT(), ROLLBACK()`

Key Processing:	`BUILD(key), BUILD(index)`
	`GET(file,key), GET(key,keypointer),`
	`RESET(key,string)`
	`SET(file,key), SET(key), SET(key,key),`
	`SET(key,keypointer), SET(key,key,filepointer),`
	`DUPLICATE()`
	`POINTER(key)`
	`POSITION(key)`
	`RECORDS(key)REGET(key)`

Record Locking:	`HOLD(), RELEASE()`
File Buffering:	`STREAM()`
File Information:	`RECORDS(file)`
Sequential Processing:	`PREVIOUS(), BOF(), SKIP()`
File Manipulation:	`BUILD(), DELETE(), PACK(), WATCH(), REGET()`

Miscellaneous

❖ POSITION(file) returns a STRING(4).

FOXPRO AND FOXBASE FILES

The FoxPro file driver is compatible with FoxPro and FoxBase. The default data file extension is *.DBF.

The default index file extension is *.IDX. The default Memo file extension is .FBT. FoxPro also supports multiple index files, whose default extension is *.CDX. The miscellaneous section describes the procedures for using the .CDX files.

Files:	**CWFOX16.LIB**	Windows Export Library (16-bit)
	CWFOX32.LIB	Windows Export Library (32-bit)
	CLFOX16.LIB	Windows Static Link Library (16-bit)
	CLFOX32.LIB	Windows Static Link Library (32-bit)
	CWFOX16.DLL	Windows Dynamic Link Library (16-bit)
	CWFOX32.DLL	Windows Dynamic Link Library (32-bit)

Tip: The FoxPro index file format is the backbone of its vaunted “Rushmore” technology. The old saying “There’s no free lunch,” however, applies. Adding and appending records to a large database is a slower process than in other xBase formats, due to the time required to update the index file.

Data Types

The xBase file format stores all data as ASCII strings. You may either specify STRING types with declared pictures for each field, or specify native Clarion types, which the driver converts automatically.

FoxPro data type	Clarion data type	STRING w/ picture
Date	DATE	STRING(@D12)
*Numeric	REAL	STRING(@N-_p.d)
*Logical	BYTE	STRING(1)
Character	STRING	STRING
*Memo	MEMO	MEMO

If your application reads and writes to existing files, a pictured STRING will suffice. However, if your application *creates* a FoxPro or FoxBase file, you may require additional information for these FoxPro and FoxBase types:

❖ To create a numeric field in the Data Dictionary, choose the REAL data type. In the External Name attribute, specify ‘*NumericFieldName*=N(*Precision*,*DecimalPlaces*)’ where

NumericFieldName is the name of the field, *Precision* is the precision of the field and *DecimalPlaces* is the number of decimal places.

If you’re hand coding a native Clarion data type, add the NAME attribute using the same syntax. If you’re hand coding a STRING with picture, STRING(@N-_9.2), NAME(‘*Number*’), where *Number* is the field name.

❖ To create a logical field, using the data dictionary, choose the BYTE data type. There are no special steps; however, see the miscellaneous section for tips on reading the data from the field.

  If you’re hand coding a STRING with picture, add the NAME attribute: STRING(1), NAME(‘*LogFld* = L’)**.**

❖ To create a date field, using the data dictionary, choose the DATE data type, rather than LONG, which you usually use for the TopSpeed or Clarion file formats.

❖ MEMO field declarations require the a pointer field in the file’s record structure. Declare the pointer field as a STRING(10) or a LONG. This field will be stored in the .DBF file containing the offset of the memo in the .DBT file. The MEMO declaration must have a NAME() attribute naming the pointer field. An example file declaration follows:

```
File  FILE, DRIVER(‘FoxPro’)
Memo1   MEMO(200),NAME(‘Notes’)
Memo2   MEMO(200),NAME(‘Text’)
Rec     RECORD
Mem1Ptr   LONG,NAME(‘Notes’)
Mem2Ptr   STRING(10),NAME(‘Text’)
        END
      END
```

File Specifications/Maximums

```
File Size:              2,000,000,000 bytes
Records per File:       1,000,000,000 bytes
Record Size:            4,000 bytes
Field Size
     Character:         254 bytes
     Date:              8 bytes
     Logical:           1 byte
     Numeric:           20 bytes including decimal point
     Float:             20 bytes including decimal point
     Memo:              65,520 bytes (see note)
Fields per Record:      255
Keys/Indexes per File:  No Limit
Key Sizes
     Character:         100 bytes (.IDX)
                254 bytes (.CDX)
     Numeric, Date:     8 bytes
Memo fields per File:   Dependent on available memory
Open Files:             Operating system dependent
```

Driver Strings and SEND functions

Driver strings (the second parameter of the DRIVER attribute) are all preceded by a forward slash character (/). SEND function commands can take two formats—one with an equal sign modifies a switch setting and return the value of the previous switch setting; the other format (without an equal sign) returns the value of the switch.

Driver strings are sent to the file driver when the file is opened. The SEND function sends a command to modify a setting after the file is open. Some driver strings have no effect after the file is open, so the SEND function syntax to modify the setting is not listed. However, the SEND function syntax to return the value of the switch is listed for all driver strings.

`/BUFFERS=n`

Specify a value for the number of buffers used to read and write to the file.

The FoxPro driver utilizes DOS buffering. The default is three buffers of 1024 bytes each. Increasing the number of buffers will not increase performance when a file is shared by multiple users.

`SEND(file,'BUFFERS')`

Returns the number of buffers in the form of a STRING.

`/RECOVER`
`SEND(file,'RECOVER')`

Equivalent to the Xbase RECALL command, which recovers records marked for deletion. When using the FoxPro driver, the DELETE statement flags a record as “inactive.” The driver does not remove the record until the PACK command is executed.

/RECOVER is evaluated each time you open the file if you add the driver string to the data dictionary. When the driver recovers the records previously marked for deletion, you must manually rebuild keys and indexes with the BUILD statement.

`/IGNORESTATUS=on|off`	When set *on*, the driver does *not* skip deleted records when accessing the file with GET, NEXT, and PREVIOUS in file order. It also enables a PUT on a deleted or held record. / IGNORESTATUS requires opening the file in exclusive mode.
`SEND(file,'IGNORESTATUS')`	Returns the IGNORESTATUS setting (ON or OFF) in the form of a STRING(3).
`SEND(file,'DELETED')`	For use only with the SEND command, when IGNORESTATUS is on. Reports the status of the current record. If deleted, the return string is "ON;" if not, "OFF."

Unsupported/Modified Functions & Attributes

Memos: `BINARY`

FoxPro and FoxBase support only text memos.

Keys: `DUP, NOCASE, OPT, ascending|descending`

File: `ENCRYPT, OWNER, RECLAIM`

The FoxPro driver cannot read encrypted FoxPro or FoxBase files. To reclaim space from deleted records, call PACK(file).

Transaction Processing: `COMMIT(), LOGOUT(), ROLLBACK()`

The FoxPro driver does not support any transaction logging.

Record Access: `GET(file, fileptr, len), ADD(file, len)`

FoxPro and FoxBase do not support variable length records

File updates: `PUT(file, fileptr, len)`

FoxPro and FoxBase do not support variable length records

`EOF(file), BOF(file)`

Although the driver supports these functions, we do not recommend their use. They must physically access the files and add overhead. Instead, test the value returned by ERRORCODE() after each sequential access. NEXT or PREVIOUS post Error 33 (Record Not Available) if an attempt is made to access a record beyond the end or beginning of the file.

`ADD(file)` vs. `APPEND(file)`

The ADD statement tests for duplicate keys before modifying the data file or its associated KEY files. Consequently it is slower than APPEND which performs no checks and does not update KEYs. When adding large amounts of data to a database use APPEND...BUILD in preference to ADD.

`BUILD(key, str)`

When building dynamic indexes, the *str* component may take one of two forms:

`BUILD(DynNdx, '+Pre:FLD1, -Pre:FLD2')`

This form specifies the names of the fields on which to build the index. The field names must appear as specified in the fields' NAME() attribute if supplied, or must be the label name. A prefix may be used for compatibility with the Clarion conventions but is ignored.

`BUILD(DynNdx, 'T[Expression]')`

This form specifies the type and Expression used to build an index, see the miscellaneous section, below.

`COPY(file, newname)`

The COPY() command copies data and memo files using *newname*, which may specify a new file name or directory. Key or index files are copied if the *newname* is a subdirectory specification. To copy an index file to a new file, use a special form of the copy command:

```
COPY(file,'<index>|<newname>')
```

This returns *File Not Found* if an invalid index is passed. The COPY command assumes a default extension of ".IDX" for both the source and the target file names if none is specified. If you require a file name without an extension, terminate the name with a period. Given the file structure:

```
Clar2    FILE,CREATE,DRIVER('FOXPRO')
NumKey     KEY(Num),DUP
StrKey     KEY(Str1)
StrKey2    KEY(Str2)
AMemo      MEMO(100), NAME('mem')
Record     RECORD
Num          STRING(@n-_9.2)
STR1         STRING(2)
STR2         STRING(2)
Mem          STRING(10)
        . .
```

The following commands copy this file definition to A:

```
COPY(Clar2,'A:\CLAR2')
COPY(Clar2,'StrKey|A:\STRKEY')
COPY(Clar2,'StrKey2|A:\STRKEY2')
COPY(Clar2,'NumKey|A:\NUMKEY')
```

After these calls, the following files would exist on drive A: CLAR2.DBF, CLAR2.DBT, STRKEY.IDX, STRKEY2.IDX, and NUMKEY.IDX.

```
DELETE(file)
```

When the driver deletes a record from a FoxPro or FoxBase database, the record is not physically removed, instead the driver marks it inactive. Memo fields are not physically removed from the memo file, however they cannot be retrieved if they refer to an inactive record. Key values *are* removed from the index files. To remove records and memo fields permanently, execute a PACK(file).

Tip: To those programmers familiar with FoxPro, this driver processes deleted records consistent with the way FoxPro processes them after the SET DELETED ON command is issued. Records marked for deletion are ignored from processing by executable code statements, but remain in the data file.

```
HOLD(file), HOLD(file, timeout)
```

FoxPro and FoxBase perform record locking by locking the entire record within the data file. This prevents read access to other processes. Therefore we recommend minimizing the amount of time for which a record is held.

`POINTER(file), POINTER(key)`

There is no distinction between file pointers and key pointers; they both contain the same value for any given record.

`RECORDS(file), RECORDS(key)`

Under FoxPro and FoxBase the RECORDS() function reports the same number of records for the data file and its keys and indexes. Usually there will be no difference in the number of records *unless* the INDEX is out of date. Because the DELETE statement does not physically remove records, *the number of records reported by the* RECORDS() *function includes inactive records.* Exercise care when using this function.

`RENAME(file, newname)`

The RENAME command copies the data and memo files using *newname*, which may specify a new file name or directory path. Key and index files must be renamed using the same syntax as the COPY command, above.

- POSITION(file) returns a STRING(12).
- POSITION(key) returns a STRING the size of the key fields + 4 bytes.
- `BUILD(DynamicIndex, expression,filter)` is not supported.

Miscellaneous

❖ FoxPro and FoxBase allow a logical field to accept one of 11 possible values (0,1,y,Y,n,N,t,T,f,F or a space character). The space character is neither true nor false. When using a logical field from a preexisting database in a logical expression, account for all these possibilities. Remember that when a STRING field is used as an expression, it is true if it contains any data and false if it is equal to zero or blank. Therefore, to evaluate a Logical field's truth, the expression should be true if the field contains any of the "true" characters (1,T,t,Y, or y). For example, if a Logical field were used to specify a product as taxable or nontaxable, the expression to evaluate its truth would be:

(If Condition):

```
Taxable='1' OR Taxable='T' OR Taxable='t' OR Taxable='Y' OR Taxable='y'
```

❖ Clarion for Windows supports MEMO fields up to a maximum of 64K. If you have an existing file which includes a memo greater than 64K, you can use the file but not modify the large MEMOs.

❖ You can determine when your application encounters a large MEMO by detecting when the memo pointer variable is non-blank, but the memo appears to be blank. Error 47 (Bad Record Declaration) is posted, and any modification to the MEMO field is ignored.

❖ FoxPro and FoxBase support a maximum of 10 characters in a field name. If you require more, use an External Name with 10 characters or less.

❖ FoxPro and FoxBase support the use of expressions to define keys. Within the Dictionary Editor, you can place the expression in the external name field in the *Key Properties* dialog. The general format of the external name is :

```
'FileName=T[Expression]'
```

Where *FileName* represents the name of the index file (which can contain a path and file extension), and *T* represents the type of the index. Valid types are: C = character, D = date, and N = numeric. If the type is D or N then *Expression* can name only one field.

❖ Multiple-index (.CDX) files require the NAME() attribute on a KEY or INDEX to specify the storage type of the key and any expression used to generate the key values. The general format of the NAME() attribute on a KEY or INDEX is:

```
NAME('TagName|FileName[PageSize]=T[Expression],COMPRESSED')
```

The following documents the parameters for the NAME() attribute:

`TagName`	Names an index tag within a multiple index file. If the TagName is omitted the driver creates an .IDX file with the name specified in FileName.
`FileName`	Names the index file, and optionally contains a path and extension.
`PageSize`	May only be specified when creating a .CDX file (if a TagName is specified). It is a number in the range 2-32 specifying the number of 512-byte blocks in each index page. This value is only used when creating the file. If multiple values are specified via declarations for different tags in the same .MDX file, the largest value will be selected. The default value is 2.
`T`	Specifies the type of the index; legal types are C = character, D = date, N = numeric. If the type is D or N then *Expression* may name only one field.
`Expression`	Specifies the expression used to generate the index. The expression may refer to multiple fields, and invoke multiple of xBase functions. The functions currently supported are listed below. Square brackets must enclose the expression.
`COMPRESSED`	When specified, the FoxPro Driver creates a FoxPro 2 compatible compressed .IDX file.

Elements of the NAME() attribute may be omitted from the right. When specifying an Expression, the type and name must also be specified. If the Expression is omitted, the driver determines the Expression from the key fields when the file is created, or from the index file when opened.

If the type is omitted, the driver determines the index type from the first key component when the file is created, or from the index file when opened.

If the NAME() attribute is omitted altogether, the index file name is determined from the key label. The path defaults to the same location as the .DBF.

Tag names are limited to 9 characters in length; if the supplied name is too long it is automatically truncated.

All field names in the NAME() attribute must be specified without a prefix.

- FoxPro additionally supports the use of the Xbase FOR statement in expressions to define keys. The expressions supported in the FOR condition must be a simple condition of the form:

  ```
  expression comparison_op expression
  ```

  comparison_op may be one of the following: <, <=, =<, <>, =, =>, >= or >.

  The expression may refer to multiple fields in the record, and contain xBase functions. Square brackets must enclose the expression. The currently supported functions appear below. If the driver encounters an unsupported Xbase function in a preexisting file, it posts error 76 'Invalid Index String' when the file is opened for keys and static indexes.

  String expressions may use the '+' operator to concatenate multiple string arguments. Numeric expressions use the '+' or '-' operators with their conventional meanings. The maximum length of a FoxPro or FoxBase expression is 250 characters.

Supported xBase Commands

`ALLTRIN(string)`	Removes leading and trailing spaces.
`CTOD(string)`	Converts a string key to a date. The *string* format mm/dd/yy; the result takes the form ‘yyyymmdd’. The yyyy element of the date defaults to the twentieth century. An invalid date results in a key containing blanks.
`DELETED()`	Returns TRUE if the record is deleted.
`DTOC(date)`	Converts a date key to string format ‘mm/dd/yy.’
`DTOS(date)`	Converts a date key to string format ‘yyyymmdd.’
`FIXED(float)`	Converts a float key to a numeric.
`FLOAT(numeric)`	Converts a numeric key to a float.
`IIF(bool,val1,val2)`	Returns val1 if the first parameter is TRUE, otherwise returns val2.
`LEFT(string, n)`	Returns the leftmost *n* characters of the string key as a string of length *n*.
`RIGHT(string, n)`	Returns the rightmost *n* characters of the string key as a string of length *n*.
`RTRIM(string)`	Removes spaces from the right of a string.
`STR(numeric [,length[, decimal places]])`	converts a numeric to a string. The length of the string and the number of decimal places are optional. The default string length is 10, and the number of decimal places is 0.
`SUBSTR(string,offset,n)`	Returns a substring of the *string* key starting at *offset* and of *n* characters in length.
`TRIM(string)`	Removes spaces from the right of a string (identical to RTRIM).
`UPPER(string)`	Converts a string key to upper case.
`VAL(string)`	Converts a string key to a numeric.

TOPSPEED DATABASE FILES

The TopSpeed Database file system is a high-performance, high-security, proprietary file driver for Clarion development tools. It is *not* compatible with Clarion 2.1 and 3.0 files.

Data tables, keys, indexes and memos can all be stored together in a single DOS file. The default file extension is *.TPS. A separate "Transaction Control File" takes the *.TCF extension.

The TopSpeed driver can optionally store multiple tables in a single DOS file. This allows you to open as many data tables, keys and indexes as necessary using *a single DOS file handle*. This feature may be especially useful when there are a large number of small tables, or when a group of related files are normally accessed together. All keys, indexes, and Memos are always stored internally.

In addition, the TopSpeed file system supports the **BLOB** data type (Binary Large OBject), a string field which is completely variable-length and may be greater than 64K in size (in both 16 and 32-bit applications). A BLOB must be declared before the RECORD structure. Memory for a BLOB is dynamically allocated and de-allocated as necessary. For more information, see *BLOB* in Chapter 10 of the *Language Reference.*

Files:	**CWTPS16.LIB**	Windows Export Library (16-bit)
	CWTPS32.LIB	Windows Export Library (32-bit)
	CLTPS16.LIB	Windows Static Link Library (16-bit)
	CLTPS32.LIB	Windows Static Link Library (32-bit)
	CWTPS16.DLL	Windows Dynamic Link Library (16-bit)
	CWTPS32.DLL	Windows Dynamic Link Library (32-bit)

Tip: This new driver offers speed, security, and takes up fewer resources on the end users system.

Data Types

```
BYTE                     DECIMAL
SHORT                    STRING
USHORT                   CSTRING
LONG                     PSTRING
ULONG                    MEMO
SREAL                    GROUP
REAL                     BLOB
```

Maximum File Specifications

File Size	:	Limited only by disk space
Records per File	:	Unsigned Long (4,294,967,295)
Record Size	:	64K
Field Size	:	64K
Fields per Record	:	64K
Keys/Indexes per File	:	240
Key Size	:	64K
Memo fields per File	:	255
Memo Field Size	:	64K
BLOB fields per File	:	255
BLOB Size	:	Hardware dependent
Open Data Files	:	Operating system dependent

Driver Strings and SEND functions

Driver strings (the second parameter of the DRIVER attribute) are all preceded by a forward slash character (/). SEND function commands can take two formats—one with an equal sign modifies a switch setting and return the value of the previous switch setting; the other format (without an equal sign) returns the value of the switch.

Driver strings are sent to the file driver when the file is opened. The SEND function sends a command to modify a setting after the file is open. Some driver strings have no effect after the file is open, so the SEND function syntax to modify the setting is not listed. However, the SEND function syntax to return the value of the switch is listed for all driver strings.

`SEND(file,'TCF=file name')`

`/TCF=file name` Specifies a transaction control file other than the default \TOPSPEED.TCF. The file holds all multi-file commits until the program terminates or a SEND(TCF=file name) executes.

`SEND(file,'PNM=name')`

`/PNM=name` Retrieves the names of the tables in a group (a single DOS file).

To retrieve the first name, issue this command: SEND(file,'PNM='). This returns the name of the first table. Subsequent calls pass the name received and return the next name.

For example, given a file with three tables—Supp, Part, and Ship, the example below displays an alphabetical listing:

```
CODE
name = ' '

  LOOP
  name = (SEND(Supp,'PNM=' & name)
    If name
      DISPLAY name
    ELSE
      BREAK
    END
  END
```

Unsupported Functions and Attributes

- Record Access: `GET(file, fileptr, len), ADD(file, len), APPEND(file, len)`

  The TopSpeed Driver does not support variable length records

  ```
  GET(file,1)
  ```

  This relies on a valid pointer returned from the POINTER() function. You cannot use GET(file,1) to retrieve the first record because 1 is not a valid pointer.

- File updates: `PUT(file, fileptr, len)`

  The TopSpeed Driver does not support variable length records

- Keys: `NAME()`

  The TopSpeed Driver does not support external names for keys. All keys are stored internally.

Miscellaneous

- SHARE and open access modes:

  <u>The following open access modes are supported</u> *<u>Share required</u>*

  ```
  34 (12h) Read/Write, deny write (default for OPEN)     Yes
  66 (42h) Read/Write, deny none (default for SHARE)     Yes
  64 (40h) Read Only,  deny none                         Yes
  18 (12h) Read/Write, deny all                          No
  16 (10h) Read Only,  deny all                          No
  32 (20h) Read Only,  deny write                        No
  ```

  For the modes indicated, SHARE.EXE (which implements DOS record locking) must be loaded in AUTOEXEC.BAT or CONFIG.SYS. The following example loads SHARE in AUTOEXEC.BAT, providing 500 maximum file locks, and the default 2048 bytes for the storage area.

  ```
  C:\DOS\SHARE.EXE /L:500
  ```

  If SHARE.EXE is required but not loaded, the program generates a runtime error when OPEN or SHARE is called (deny none modes), or when an update is attempted (deny write modes).

- `APPEND()`

  APPEND() is recommended over ADD() if the total size of the keys exceeds the amount of RAM available, if there is more than one key, or when adding a large number of records. The size of a key (for this purpose) is the number of entries times (the sum of key fields + 10 bytes). If the records being added are already in an approximate key order, then you can discount that key for the purposes of the above calculation.

  As an example, if a file has two 40 byte keys and 2 Megabytes of RAM are available, then ADD() becomes (relatively) slow when the database size exceeds about 2,000,000 / (40 + 10 + 40 + 10) = 20,000 records.

- `BUILD(file), BUILD(key)`

  The TopSpeed driver implements incremental building; this means that building a key only reads records starting from the first record appended since the key was last built. The driver merges the new keys with the existing key. Thus building a large key where only a few recently added records have been modified should be *fast*. Building an index is similar, but must start at the minimum physical record whose position in the index has changed since the index was last built.

  Dynamic indexes are not retained, so cannot be built incrementally.

- `LOCK(file)`

  LOCK() only affects other LOCK() calls. The only effect of a successful call to LOCK() is that other processes will get an error FLALLK when they call LOCK().

- `LOGOUT(), COMMIT(), ROLLBACK()`

  A transaction control file is used to ensure that transactions which update more than one DOS file are committed atomically. By default the transaction control file (.TCF) has the name “\TOPSPEED.TCF.” A SEND() command allows you to change this.

  The .TCF file must be accessible when any files controlled by it are accessed. If a transaction involves updating more than one shared network file, you should specify a transaction control file on the network. It is not necessary to use the same TCF file for all transactions; however, it must reside where it can be read by everyone accessing the file. If not, after a crash/power-fail during a COMMIT(), some files may be updated, and others not. (The files will not be corrupted - they may just not be consistent with one another).

A .TCF file can be deleted only if all files controlled by it may have been opened (for writing) since a crash/power-fail.

❖ `POINTER(key)`

The value returned by POINTER(key) corresponds to a physical data record. Consequently when that record is removed by a call to DELETE() the pointer becomes invalid. Any subsequent access using the pointer fails. If you require fuzzy matching whereby the nearest record is returned, use the POSITION() function and appropriate access functions.

❖ `STREAM(), FLUSH()`

When reading a large number of records, use STREAM() or open the file in a deny write mode e.g. OPEN(f) rather than SHARE(f). After the records have been read, call FLUSH() to allow other users access.

It is very important to use STREAM() when adding/appending/ putting a large number of records. STREAM() will typically make processing about 20 times faster. For example, adding 1000 records might take nearly 2 minutes without STREAM(), but *only 5 seconds* with STREAM. It is not necessary to use STREAM() or FLUSH() on a logged out file (performance on logged out files is always good).

Tip: When utilizing STREAM() to update a large number of records, the driver stores uncommitted or unflushed pages in memory, and it is possible to run out of memory. Calling COMMIT(), FLUSH(), or LOGOUT() periodically prevents this. To calculate the maximum “updates” between each COMMIT(), divide the available memory by the update size. When appending, the update size is approximately the size of the record in bytes. When adding, the update size is approximately the size of the records and key component fields in bytes. When updating records using PUT(), it’s theoretically possible for the update size to reach 7K. In practice, we recommend committing data every 100 or so updates.

❖ POSITION(file) returns a STRING the size of the key fields + 4 bytes.

❖ POSITION(key) returns a STRING the size of the key fields + 4 bytes.

Storing multiple Tables (data files) in a single DOS file.

By using the special escape sequence '\!' in the NAME() attribute of a TopSpeed file declaration, you can specify that a single DOS file will store more than one table. For example, to declare a single DOS file 's&p.tps' which is to contain 3 logical tables, called *supp*, *part* and *ship*:

```
Supp  FILE,DRIVER('TopSpeed'),PRE(Supp),CREATE,NAME('S&P\!Supp')
    ...
Part  FILE,DRIVER('TopSpeed'),PRE(Part),CREATE,NAME('S&P\!Part')
    ...
Ship  FILE,DRIVER('TopSpeed'),PRE(Ship),CREATE,NAME('S&P\!Ship')
    ...
```

The data files share a single DOS file handle, opened when the first file is opened, and closed when the last file is closed. The first open mode determines the open mode for *all* the other files. If the first open mode is read-only, then no updates of any kind can be performed successfully (ACCDNID will be returned).

If one file in a group is logged out, then all the files in the group are effectively logged out. If one file in a group is flushed, then all files in the group are flushed.

This feature is especially useful when there are a large number of small tables, or when the application must normally access group of related files together.

If no escape sequence is specified, then a default table name 'unnamed' is supplied, so that the following are all equivalent:

```
foo     FILE,DRIVER('TopSpeed')
foo     FILE,DRIVER('TopSpeed'),NAME('foo')
foo     FILE,DRIVER('TopSpeed'),NAME('foo\!unnamed')
```

A SEND() command allows the programmer to determine the names of the files within a group. Files can be renamed within a group; for example, given the above declarations the following command will rename the file called Supp to Old_Supp:

```
RENAME(Supp,'S&P\!Old_Supp')
```

Renaming to another existing group normally involves copying/ removal, so is less efficient.

If your are using the OWNER attribute on multiple tables in a TopSpeed database file, all tables *must* have the same OWNER attribute.

USING THE TOPSPEED DATABASE RECOVERY UTILITY

The TopSpeed file system is designed to automatically repair most errors. If the data file is physically damaged during a system malfunction, the TopSpeed Database Recovery Utility can recover the undamaged portions of your data.

Note: The TopSpeed Database Recovery Utility is an emergency repair tool and should not be used on a regular basis. Use it only when a file has been damaged.

The TopSpeed Database Recovery Utility reads the damaged file and writes the recovered records to a new file. It uses the information stored in the file's header or scans the file recovering undamaged portions. Optionally, you can provide an example file containing table (individual file) and key layout.

The TopSpeed Database Recovery Utility is a freely distributable utility designed to enable your end users to recover damaged files.

The recovery utility is designed to work interactively or transparently via command line parameters. Interactively, you can use the utility to recover damaged files and provide the parameters via two wizard dialogs. Using the command line parameters, you can incorporate it in your application using a RUN() statement or create a shortcut (in Windows 95) or Program Manager Icon (in Windows 3.1x) with the parameters to enable end users to recover data files.

Using the TopSpeed Database Recovery Utility Interactively

1. Start the utility by double-clicking on the TopSpeed Database Recovery Utility Icon In the Clarion for Windows 1.5 Program Group.

   The **TopSpeed Database Recovery Utility** dialog appears. The utility consists of two wizard dialogs.

2. In the **Source** (file to recover) section, specify the file name or press the **Browse** button to select it from a standard file open dialog.

3. If the file has a password, type it in the **Password** entry box.

   If the database file contains multiple tables (data files), each table *must* have the same password.

4. Optionally, in the **Destination** (result file) section, specify the file name for the target file or press the **Browse** button to select it from a standard file open dialog.

By default the .TPR extension is added to the source file name. This parameter is optional. If omitted, the original (source file) is overwritten and a backup file is created. The source file is renamed to *filename*.TPx, where x is automatically incremented from 1 to 9 each time a new file is created. If all nine numbers are used, any subsequent files created are given the extension .TP$ and are overwritten.

5. If the result file is to have a different password, type it in the **Password** entry box. If omitted, the password is removed.

6. Press the **Next** button.

   The second wizard dialog for the TopSpeed Database Recovery Utility appears.

7. Optionally, specify the **Example File** file name or press the **Browse** button to select it from a standard file open dialog.

   The utility uses the Example File to determine table layouts and key definitions in the event those areas of the source file are damaged. The default extension is .TPE, but if you choose, you may use any valid DOS extension

Tip: We recommend shipping an example file when you deploy your application. This improves data recovery from a damaged file.

8. If the example file has a password, type it in the **Password** entry box.

9. If you want the utility to rebuild Keys, check the **Build Keys** box.

   If omitted, the keys are rebuilt by the original application when it attempts to open it.

10. If you want to use the Header Information in the source file, check the **Use Header** box.

    Utilizing Header Information optimizes the utility's performance, but should not be used if the file header is corrupt. If omitted, the utility searches the entire data file and restores all undamaged pages.

11. If the application uses a Locale (.ENV) File for an alternate collating sequence, specify the .ENV file or press the **Browse** button to select it from a standard file open dialog.

12. If the file is using the OEM attribute to control the collating sequence, Check the **Use OEM** box.

    This enables the OEMTOANSI and ANSITOOEM conversion.

13. Press the **Start** button to begin the recovery process.

If the utility does not find any errors, a message appears informing you that “No Errors Detected in <fliename.ext>” and asks if you want to continue with recovery.

Command Line Parameters

The utility can also accept command line parameters which enables you to execute it from an application or Program Manager Icon (or Shortcut in Windows 95).

```
TPSFIX  sourcepath[?password] [destpath[?password]]
          [/E:examplepath[?password]] [/L:localepath] [/H] [/K] [/P] [/O]
```

sourcepath	The file name and path of the source (damaged) database file.
[?password]	The database file’s password.
destpath	The file name and path of the recovered database file.
[?password]	The recovered database file’s password.
/E:*examplepath*	The file name and path of the example database file.
[?password]	The example database file’s password
/L:*localepath*	The Locale (.ENV) file used to specify an alternate collating sequence.
/H	If specified, the utility uses the header information in the source file.
/K	If specified, the utility rebuilds all keys for the database.
/P	If specified, the user is prompted for each parameter even if they are supplied on the command line.
/O	If specified, the file uses OEMTOANSI and ANSITOOEM to determine the collating sequence. See *Internationalization* in Chapter 10 of the *Language Reference.*

Using the Utility in your Application

There are some issues to consider before allowing the utility to run.

- ❏ The database file should NOT be open when running the utility. Ensure that the file is closed before allowing the user to start the utility.
- ❏ To prevent access during the recovery process is completed, the utility locks the file automatically.
- ❏ It is more efficient and safer to allow the application to rebuild the KEYs (by omitting the /K parameter in the recovery). It is also a good way to check the status of a recovery.

Running the TopSpeed Database Recovery Utility

There are basically two methods you can use from a RUN() statement: Using the first method, you omit the *destpath* parameter so the original (source) file is overwritten. This requires an Example file.

In the Application Generator:

1. In the **Actions** dialog for a button or menu item, choose *Run a Program* from the drop down list.
2. In the **Program Name** entry box, specify *TPSFIX.EXE*.
3. In the parameters entry box, specify the parameters (see *Command Line Parameters* above).

For Example:

```
TPSFIX.EXE Datafile.TPS /E:Example.TPE /H
```

In Embedded Source Code:

```
RUN('TPSFIX.EXE Datafile.TPS /E:Example.TPE /H')
```

This recovers the “datafile.TPS” file using the “Example.TPE” file as an example for the table and key layouts, does not rebuild the keys, and uses the header information in the original file. The original file is saved to a backup file with an extension of TP1 through TP9. Each time the utility is executed, the numeric portion of the extension is incremented.

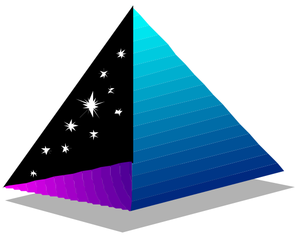
The second method requires two lines of embedded source code but gives you control over the renaming process. You insert the source code in the Accepted Embed point for the Menu Item or button.

For example:

```
COPY(datafilelabel, ‘Datafile.OLD’)      !  copies the original file
                                         !  to Datafile.OLD
RUN(TPSFIX Datafile.OLD Datafile.tps /H) !  Runs the utility using the
                                         !  renamed file as
                                         !  the source and the original
                                         !  name as the target
```

This copies the datafilelabel file to DATAFILE.OLD, recovers the file and writes it to DATAFILE.TPS using the header information in the original file.

ODBC



ODBC (Open DataBase Connectivity) is a Windows "strategic interface" for accessing data from a variety of Database Management Systems (DBMS) and data file formats, across a variety of networks and platforms. ODBC offers the same end result as the file driver libraries which ship with Clarion for Windows—file driver independence—though in a somewhat different manner.

The ODBC standard was developed and is maintained by Microsoft, which publishes an ODBC Software Development Kit (SDK), geared for use with its Visual C++ product. ODBC support is another way in which Clarion for Windows provides an extensible platform for you to create applications.

ODBC PRO'S AND CON'S

Using ODBC offers the following advantages:

- ODBC is an excellent choice in a Client-Server environment, especially if the Server is a native Structured Query Language (SQL) DBMS. It allows you to add Client-Server support to your application, without having to do much more than choose a file driver. ODBC was specifically designed to create a non-vendor-specific method of connecting front end applications to back end services. Via ODBC, the Server can handle much of the work, especially for SQL JOIN and PROJECT operations, thereby speeding up your application.

- Existing ODBC drivers cover a great many types of databases. There are ODBC drivers available for databases for which Clarion may not have a native driver—for example, for Microsoft Excel and Lotus Notes files.

- ODBC is already widespread. Major application suites such as Microsoft Office install ODBC drivers for file formats such as dBase and Microsoft Access. Keep in mind that many ODBC back end drivers have been updated and you should obtain the latest releases.

- ODBC is platform independent. One of Microsoft's prime objectives in establishing ODBC was to support easier access to legacy systems, or corporate environments where data resides on diverse platforms or multiple DBMS's. As long as an ODBC driver and back end are available, it doesn't matter whether you use Microsoft's NetBEUI, SPX/IPX, DECNet or others; your application can connect to the DBMS and access the data.

Given that there are many drivers available, and that the standard was developed by the company that developed Windows, you might consider using ODBC as the driver of choice for *all* your Windows applications. Yet, when deciding whether to use an ODBC driver or a Clarion for Windows native database driver, you must also consider possible disadvantages:

- Unfortunately, ODBC adds a layer—the ODBC Driver Manager—between your application and the database. When it comes to accessing files on a local hard drive, this generally results in slower performance. The driver manager must translate the application's ODBC API call to an SQL statement before any data access.

  ODBC uses SQL to communicate with the back end database. Although this can be very efficient when communicating with Client/Server database engines, it is normally less efficient than direct record access when using a file system designed around single record access, such as xBase or Btrieve.

- The information required by the ODBC database manager to connect to a data source varies from one ODBC driver to another. Unlike the selection of Clarion file drivers, where file operations are virtually transparent, you may need to do some work to gather the information required to use a particular ODBC driver. This chapter provides a few tips that might make it easier, and many ODBC drivers come with a Help (.HLP) file which documents special settings (usually stored in ODBC.INI); but the burden is on *you* to solve any problems with third-party ODBC drivers.

- ODBC is not included with Windows. When distributing your application, you'll need to install the ODBC drivers and the ODBC driver manager into the end user's system, if the end user doesn't have them already. This requires the ODBC SDK from Microsoft. In some cases, the back end server may have already provided a distribution kit which installs the ODBC driver on the workstation.

- The normal Microsoft setup program that installs the ODBC driver manager adds an applet to the end user's Control Panel window for managing ODBC. It's very easy for an end user to use this tool to change the settings in the ODBC.INI file. The end user can unwittingly remove or modify the settings for the back end ODBC driver which would make it impossible for your application to connect to the data file. Additionally, since most ODBC drivers store the data directory in ODBC.INI, it's very easy for the end user to change it, again introducing a possible problem for your application.

Given the pros and cons, we recommend using the native Clarion for Windows file drivers when both a native driver and an ODBC driver exist for the same file format.

HOW ODBC WORKS

When you use ODBC to access data, four components must cooperate to make it work:

- *Your application* calls the ODBC driver manager, and sends it the appropriate requests for data, via the ODBC API.

  Clarion for Windows does this for you transparently, using either the CWODBC16.DLL (16-bit) or the CWODBC32.DLL (32-bit) application extension. When hand-coding, be sure to include this library in the project. When distributing your application, be sure to include this file with your .EXE file (unless you use produce a "one-piece" .EXE).

- *The ODBC driver manager* receives the API calls, checks ODBC.INI for information on the data source, then loads the ODBC "back-end" driver.

  The actual "interface" to the driver manager is a file called ODBCADM.EXE, which the Microsoft setup program places in the \Windows\System directory. This is the ODBC Administrator, which then loads other libraries to do its work.

- *The ODBC "back-end" driver* is another library (.DLL) which contains the executable code for accessing the data.

  Various third-parties supply "back-end" drivers. For example, Lotus Development Corp. supplies the ODBC driver for Lotus Notes. Microsoft Office distributes an ODBC SDK containing drivers for most of their database products.

- *The data source* is either a data file (usually when ODBC is used for local data access), or a remote DBMS, such as in a case where ODBC is used to access an Oracle 7 database.

  The data source has a descriptive name; for example, "Microsoft Access Databases." The name serves as the section name in the ODBC.INI file.

The ODBC driver manager *must* know the exact data source name so that it can load the right driver to access the data. Therefore, it's vitally important that *you* know the precise data source name.

ADDING ODBC SUPPORT TO YOUR APPLICATION — THE BASICS

Adding ODBC support to your application only requires choosing Clarion's ODBC driver and providing the parameters to pass to the ODBC driver manager. You provide the parameters in the OWNER and NAME attributes of the FILE declaration. When creating a Data Dictionary for ODBC tables, importing the file definitions provides this information in the appropriate fields.

The following introduces the basics, as approached from the Data Dictionary Editor. Of course, you must also be sure that the field data types in your dictionary match the variable formats supported by the DBMS you're connecting to.

1. Create a new Dictionary file.
2. Choose **File ➤ Import File**.

   The Select File Driver dialog appears.
3. Select ***ODBC*** from the drop down list, then press the **OK** button.

   The Data Sources dialog appears. This is similar to the ODBC Administrator's interface. If the data source has not yet been defined, you can add it by pressing the **New** button.
4. Highlight the desired Data Source, then press the **Next** button.
5. If the Data Source has password protection, the Logon dialog appears. Provide the User ID and password, then press the **OK** button.

   If the file contains multiple tables, the **Tables for ...** dialog appears.
6. Highlight the desired table, then press the **Finish** button.

   The file definition is imported and the **File Properties** dialog appears allowing you to modify attributes, if you choose.

   Notice the fields in the **File Properties** dialog that it has filled in during the import:

Name:	This is extracted from the Table name as defined in the ODBC database. You may modify this, if desired. It is used as the Clarion label in your source code.

Prefix: This defaults to the first three characters of the table name, you may modify this if desired.

Owner Name The ODBC data source name, and optionally, the user ID, and password, separated by commas. Some databases require additional connection information. This information follows the password and is separated by semicolons, using the syntax: *keyword=value;keyword=value*.

For example, when accessing a Sybase database, this would appear as :

```
A Data Src,UserID,PassWord,DATABASE=DataBaseName; APP=APPName
```

The data source name is the section name in the ODBC.INI file which stores all the information necessary for the ODBC manager to load the driver and access the data. The Application Generator will add the information to the OWNER attribute of the file declaration:

```
OWNER(DataSourceName, UserID, Password)
```

Full Pathname: The import process places the *table name only* in this field. The ODBC driver retrieves the physical file name from ODBC.INI.

This places the file or table name in the NAME attribute of the file declaration:

```
NAME(DataFileName) or NAME(TableName)
```

The remainder of the attributes depend on your preferences and your application.

7. Repeat the last six steps for each table in the database.

USING EMBEDDED SQL

You can use Clarion's property syntax to embed SQL statements in your program code by using PROP:SQL naming the file as the *target*. This is only appropriate when using an SQL file driver, such as the ODBC driver.

You may embed any SQL statements supported by the back-end SQL server. If you issue an SQL statement that causes a result set to be returned (such as an SQL SELECT statement), you use NEXT(file) to retrieve the result set (one row at a time) into the file's record buffer. The FILEERRORCODE() and FILEERROR() functions will return any error code and error string set by the back-end SQL server.

You may also query the contents of PROP:SQL to get the last SQL statement issued by the file driver.

Example:

```
SQLFile{PROP:SQL} = 'SELECT field1,field2 FROM table1'            |
                     & 'WHERE field1 > (SELECT max(field1)'       |
                     & 'FROM table2'
                                      !Returns a result set that you
                                      ! get one row at a time using
                                      ! NEXT(SQLFile)

SQLFile{PROP:SQL} = 'CALL GetRowsBetween(2,8)'!Call a stored procedure

SQLFile{PROP:SQL} = 'CREATE INDEX ON table1 (field1, field2 DESC)"

!No result set

SQLString = SQLFile{PROP:SQL} !Get last SQL statement the driver issued
```

SENDING AN SQL STATEMENT

With the ODBC driver, you can utilize the Clarion SEND function to send an SQL command to an external DBMS.

Use the SEND function *only* for operations which do not return a result set. Normal file operations which *do* return a result set require Clarion language statements or embedded SQL (see previous section).

For example, you can support network maintenance functions via the SEND command:

```
ReturnValue=SEND(FileLabel, ‘GRANT SELECT ON mytable TO fred’)
```

DRIVER LIMITATIONS

In addition to (third party) driver specific limitations, the following limitations apply to all ODBC connections.

- The NOCASE attribute on keys is not supported.
- International sorts are not supported.
- EOF(), BOF(), HOLD, RELEASE, LOCK, UNLOCK, STREAM, FLUSH, COPY, RENAME, and WATCH are not supported.
- MEMO’s are not supported.
- You must use SET followed by NEXT for processing in physical order. You cannot use PREVIOUS. POSITION and RESET are not supported for physical order.

TESTING YOUR ODBC APPLICATION

Here are two tips for use when developing your ODBC application.

The ODBC driver manager can create a log file documenting all ODBC calls. It includes the actual SQL statements made by the driver to the data source, and includes any errors posted.

Additionally, you can use the FILEERROR() function to trap ODBC “back-end” driver error messages it passes back to the Clarion for Windows ODBC driver. The following sections tell you how to take advantage of these tips.

ODBC Log Files

There are different log files you can produce. One is produced by the Clarion ODBC driver, the other through the ODBC Driver Manager.

The ODBC Driver Manager's logging writes every ODBC call and the SQL statements they generate to disk, as the calls are made. The Clarion ODBC driver only logs errors that occur. This allows you to match calls to SQLError in the ODBC manager's log to actual error messages. This slows down the process considerably, so this should only be activated during testing. Additionally, the log file can grow to large proportions very quickly, so you must turn it off and delete the file after using it.

Besides "snooping" on the actual SQL statements generated by the driver, you can zero in on any errors. If the application was unable to connect, you can open the log file using the Write or WordPad applet (the file is usually too big for Notepad). Scroll to the very bottom of the file, then work up until you find the word "SQLError."

To enable Clarion ODBC driver logging:

You can enable logging on a system-wide basis, on a per-file basis, or on demand using a SEND() command.

For system-wide logging:

1. Add the following to your *WIN.INI* file:

   [CWODBC]

   Trace=1

   TraceFile=[name of trace file]

For file logging:

1. In the File Properties dialog, in the Dictionary Editor, add the following in the Driver Options entry box:

   /LOGFILE=filename.ext

   where *filename.ext* is the name of the logfile you wish to create.

For logging on demand:

1. Use a SEND() command at the appropriate point in your code, using the following syntax:

   ```
   SEND(file,'/LOGFILE=filename.ext')
   ```

   where *file* is the label of the data file and *filename.ext* is the name of the logfile you wish to create.

   To turn logging off:

1. Use a SEND() command at the appropriate point in your code, using the following syntax:

   ```
   SEND(file,'/LOGFILE')
   ```

   where *file* is the label of the data file

To enable ODBC Administrator logging:

1. Start the ODBC administrator.

   You can do so by either running the ODBCADM.EXE file in the \Windows\System directory, or by DOUBLE-CLICKING the ODBC icon in Control Panel.

2. Press the **Options** button in the **Data Sources** dialog.
3. Check the **Trace ODBC Calls** box.
4. Optionally uncheck the **Stop Tracing Automatically** box if you think you need to test connecting more than once.

   It's common to test several times before pinning down the error.

5. Press the **Select File** button and name a file to log to.

   The default is called SQL.LOG.

6. Switch to your program and begin testing.

   After the errors occur, open the log file and examine it. Remember to turn off the **Trace ODBC Calls** box when done testing.

MISCELLANEOUS ODBC NOTES

You can use Clarion’s property syntax to save a data source’s connection information to a variable by using {PROP:ConnectString} with the file label as the target.

Example:

```
AFileOwner  STRING(256)
AFile FILE,DRIVER(‘ODBC’),OWNER(AFileOwner)
  AFileOwner=’DataSource’
  OPEN(Afile)
  IF NOT ERRORCODE() THEN
  AFileOwner=AFile{PROP:ConnectString}
```

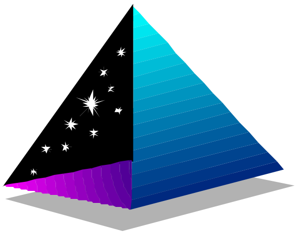
You can also use Clarion’s property syntax to set a time limit (TimeOut) for an ODBC database’s login screen. If the user does not respond in the allotted time, the connection will fail and the login is aborted. The default is to wait indefinitely for user input. Not all back ends support this feature and may ignore your value.

Example:

```
  AFile{PROP:LoginTimeOut}=60 !allow 1 minute for login
```

GETTING STARTED WITH DDE

Contents



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 for Windows allows you to 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.

CAPABILITIES

As a DDE client, your application can:

- Initiate a DDE conversation with a DDE server via the DDECLIENT function.
- Receive data from a server via the DDEREAD statement. EVENT:DDEdata tells your application when there is data for it to read.
- Send a command string to a server via 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 via the DDEWRITE statement, when the “mode” is set to DDE:auto.

  This allows you to 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.

STARTING THE 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 **Edit** button.
3. Choose the **Source** item in the **Embedded Source** dialog.
4. Press the **Add** button.
5. 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                                ! Give up if no contact
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.

```
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 via 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!

2. In the next embedded source line, tell Excel to save the new (blank) sheet under a name that you specify.

```
DDEEXECUTE(Channel,'[SAVE.AS("DDE_TEST.XLS",1,"",FALSE,"",FALSE)]')
```

Knowing the name allows you to close this channel, then open another specifying the file name as the topic. Note that the Excel command string requires double-quote marks.

3. Terminate the channel started under the "System" topic.

```
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:

1. Open the DDE channel and name the file as the topic.

```
Channel = DDECLIENT('Excel','DDE_TEST.XLS')
                              ! New channel under known file name
```

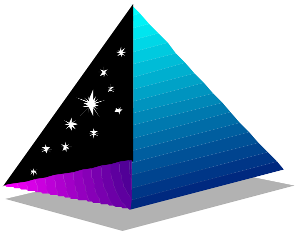
❏ To place data in a spreadsheet cell, use the DDEPOKE statement.

```
DDEPOKE(Channel,'R1C1','999')
```

Following the successful placement of the value in the spreadsheet, you could then send further Excel macro statements using DDEEXECUTE. This would allow you to 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*.

MAKING API CALLS



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 runtime*. 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 C:\CW15\LIBSRC\WINDOWS.CLW. You may also want to read *How to use DLLs not created in Clarion for Windows* in the *Frequently Asked Questions* section of Clarion’s on-line help. See also *Function and 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

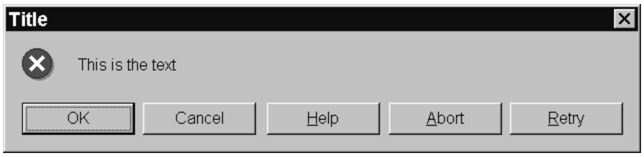
From a Clarion Language perspective, API calls can be divided into two categories: Windows API calls and other API calls. Windows API calls are calls to the functions that live in the three main Windows libraries (USER.EXE, GDI.EXE and KERNEL.EXE).

Because the Clarion Language makes extensive use of Windows API calls, many Windows API functions are *already linked* into Clarion's runtime 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 C:\CW15\LIB\WIN16.LIB and C:\CW15\LIB\WIN32.LIB with this release so that 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 that you 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 runtime.

Creating a .LIB from a .DLL

Making and using a .LIB that corresponds to a .DLL can be accomplished through the following steps:

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 Clarion for Windows 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).

Under Clarion for Windows, 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.

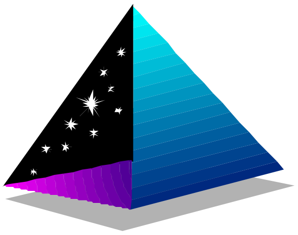
Reference the .LIB File in the Project System.

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.

The CALL Function

If you do not have a .LIB file that corresponds to the .DLL you wish to access, and cannot make one, the Clarion CALL function gives limited access to .DLL functions without explicitly linking the function. There is more overhead incurred with the CALL function than with calling the API function directly. The CALL function 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 information.

MULTI-PROGRAMMER DEVELOPMENT



Clarion for Windows' 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 for Windows programmer has a licensed copy of Clarion for Windows.

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. All team members should share a common REGISTRY.TRF and a single set of 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 utilize Clarion for Windows 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 & PHYSICALLY 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 for Windows, at each workstation, choose **Setup ➤ Application Options.**

   The **Application Options** dialog appears.

4. Check the **Multi User Development** checkbox.

This specifies that when working in the Application Generator, the copy of Clarion for Windows 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 uncheck 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 uses external .DLLs, all File definitions and all Global variables and structures 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 allows you to 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.

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 pre-planning 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.

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 "dummy" 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:

Dictionary File:	The master dictionary residing on the network.
Target Type:	DLL

In the Application's Global Properties:

Generate Global Data as External: OFF

File Control Flags

Generate All File declarations:	ON
External:	NONE EXTERNAL
Export All File declarations:	ON

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 Properties:

Dictionary File:	The master dictionary residing on the network.
Target Type:	**EXE** during the design and testing phase **DLL** when releasing to the master directory.

Note: Changing the Target 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 target type).

In the Application's Global Properties:

Generate Global Data as External: ON

File Control Flags

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** and select the corresponding .LIB for the .DLL containing the data definitions.

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' Target File 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 will generate the *.DLL and *.LIB files into the local sub-application subdirectory instead of \CW15\BIN and \CW15\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-apps 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 runtime 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:

Dictionary File:	The master dictionary residing on the network.
Target Type:	**EXE**

In the Application's Global Properties:

Generate Global Data as External:	ON
File Control Flags	
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,** Select External LIB, then select the corresponding .LIB for the .DLL containing the data definitions.

Choose **Application ➤ Insert Module,** Select External LIB, then 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.

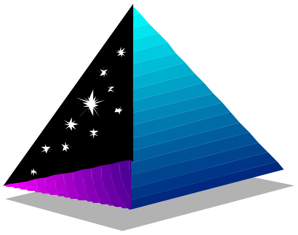
Module name: Select the corresponding .LIB for the DLL drop down list.

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.

GLOSSARY

Contents



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 for Windows 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 via 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. In general, when referring to a Windows program, “application” is the preferable term.
application generator	A program which combines prewritten, generalized executable code modules or fragments to create an application.
application generator	(Clarion) The part of the IDE which manages pre-written template procedures, obtains customizations from the developer, and generates Clarion language source code files.
API	(application programming interface) The defined set of functions provided by the operating system for use by an application.
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 runtime.
bitmap	A binary file representation of a graphic or picture; raster format defines the image by absolute pixels. Popular bitmap formats supported by Clarion for Windows 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 for Windows 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.
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 comer 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 runtime, 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.
enabled	Normal window, menu, or control state allowing focus and/or user input.
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 for Windows 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 runtime by a DLL.
include file	An external source file read and preprocessed at compile time. In Clarion for Windows, the Equates and other 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.
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 hot-key combination which directly 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 for Windows 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 rage 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.
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.
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.
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 runtime 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 for Windows, 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 for Windows 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.