



GE Fanuc Automation

CIMPLICITY® Monitoring and Control Products

CIMPLICITY HMI Plant Edition

Basic Control Engine

Program Editor Operation Manual

GFK-1305D

July 2001

Following is a list of documentation icons:



Warning notices are used to emphasize that hazardous voltages, currents, temperatures, or other conditions that could cause personal injury exist in the equipment or may be associated with its use.



Caution provides information when careful attention must be taken in order to avoid damaging results.



Important flags important information.



To do calls attention to a procedure.



Note calls attention to information that is especially significant to understanding and operating the equipment or software.



Tip provides a suggestion.



Guide provides additional directions for selected topics.

CIMPLICITY is a registered trademark of GE Fanuc Automation North America, Inc.
Windows, Windows NT, Windows 98 and Windows 2000 are registered trademarks of Microsoft Corporation
Visual Basic and Visual Basic for Applications are trademarks of Microsoft Corporation
Portions of the Program Editor are copyright 1992-1995 Summit Software

This manual was produced using *Doc-To-Help*[®], by WexTech Systems, Inc.

Copyright 1998-2001GE Fanuc Automation North America, Inc.
All rights reserved

Preface

Contents of this Manual

Chapter 1. Introduction: Gives a brief description of CIMPLICITY, the Basic Control Engine option, and the Program Editor.

Chapter 2. About the Program Editor: Gives a functional overview of the Program Editor and describes window components, menu functions, toolbars, and shortcut keys.

Chapter 3. Editing Programs: Shows you how to navigate within a script, insert, select, delete, cut and paste text, add comments, and find and replace text.

Chapter 4. Editing Custom Dialog Boxes: Shows you how to use the Dialog Editor to create and modify custom dialog boxes.

Chapter 5. Debugging Your Scripts: Shows you how to use the Debugger to trace script execution, set and remove breakpoints, and use a watch variable.

Chapter 6. Running Your Programs: Shows you how to start, suspend, and stop a program.

Appendix A. Runtime Error Messages: Lists the runtime error codes and their associated messages.

Appendix B. Compiler Error Messages: Lists the compiler error codes and their associated messages.

Related Publications

For more information, refer to these publications:

CIMPLICITY HMI Plant Edition Base System User Manual (GFK-1180)

CIMPLICITY HMI Plant Edition Basic Control Engine Language Reference Manual (GFK-1283)

CIMPLICITY HMI Plant Edition Event Editor Operation Manual (GFK-1282)

Contents

Introduction	1-1
CIMPLICITY Functionality.....	1-1
Basic Control Engine Components.....	1-2
Using the Basic Control Engine Language Reference.....	1-3
Language Elements by Category	1-4
Arrays.....	1-4
Clipboard	1-4
Comments	1-4
Comparison operators	1-4
Controlling other programs	1-5
Controlling program flow	1-5
Controlling the operating environment	1-6
Conversion.....	1-6
Data types	1-7
Database.....	1-7
Date/time.....	1-7
DDE	1-8
Error handling	1-8
File I/O.....	1-9
File system	1-9
Financial.....	1-10
Getting information from Basic Control Engine	1-10
INI Files	1-11
Logical/binary operators	1-11
Math.....	1-11
Miscellaneous	1-11
Numeric operators.....	1-12
Objects	1-12
Parsing	1-12
Predefined dialogs.....	1-12
Printing	1-13
Procedures.....	1-13
String operators.....	1-13
Strings	1-13
User dialogs	1-14
Variables and constants.....	1-15
Variants.....	1-15

About The Program Editor 2-1

Functional Overview	2-1
Wizards	2-1
Watch Window	2-2
QuickWatch Object Inspector	2-2
Trace Window	2-2
Breakpoints	2-2
Step In/Step Over	2-2
InterScript Calls	2-3
Multiple Threads of Execution	2-3
Dialog Editor	2-3
On Line Help	2-3
Getting Started	2-3
Program Editor Window Components	2-4
Program Editor Menu Functions	2-5
The File Menu	2-5
The Edit Menu	2-6
The Run Menu	2-7
The Debug Menu	2-7
The Tools Menu	2-8
The View Menu	2-8
The Window Menu	2-9
The Help Menu	2-9
Program Editor Toolbars	2-10
Standard Toolbar	2-10
Tools Toolbar	2-11
Application Toolbar	2-11
Program Editor Shortcut Keys	2-12
Setting String and Stack Space	2-13

Editing Programs 3-1

About Editing Programs	3-1
Navigating within a Script	3-1
Moving the Insertion Point with the Mouse	3-1
Moving the Insertion Point to a Specified Line	3-2
Inserting Text	3-3
Selecting Text	3-4
Selecting Text With the Mouse	3-4
Selecting Text With the Keyboard	3-5
Selecting A Line With the Keyboard	3-5
Deleting Text	3-6
Deleting Selected Text	3-6
Deleting Line Breaks	3-6
Cutting and Copying Text	3-7
Cutting A Selection	3-7
Copying A Selection	3-7
Pasting Text	3-7
Undoing Editing Operations	3-7
Adding Comments to Your Script	3-8
Adding A Full Line Comment	3-8
Adding An End of Line Comment	3-8

Breaking a Statement across Multiple Lines	3-9
Searching and Replacing	3-10
Finding Text in Your Script	3-10
Replacing Text in Your Script	3-11
Checking the Syntax of a Script	3-12
Editing Dialog Box Templates	3-13

Editing Custom Dialog Boxes 4-1

About Editing Dialog Boxes	4-1
Overview of Dialog Editor	4-2
Features of the Dialog Editor	4-2
Using the Dialog Editor	4-3
Dialog Editor's Application Window	4-3
Keyboard Shortcuts for Dialog Editor	4-5
Using the Help System	4-6
Creating a Custom Dialog Box	4-7
Types of Controls	4-7
Adding Controls to a Dialog Box	4-9
Using the Grid to Help You Position Controls within a Dialog Box	4-11
Creating Controls Efficiently	4-12
Editing a Custom Dialog Box	4-14
Selecting Items	4-14
Using the Information Dialog Box	4-15
Attributes That You Can Adjust with the Dialog Box Information Dialog Box ...	4-17
Attributes That You Can Adjust with the Information Dialog Box for a Control ..	4-18
Changing The Position of an Item	4-19
Changing the Size of an Item	4-21
Changing Titles and Labels	4-22
Assigning Accelerator Keys	4-22
Specifying Pictures	4-24
Creating or Modifying Picture Libraries under Windows	4-25
Duplicating and Deleting Controls	4-26
Undoing Editing Operations	4-27
Editing an Existing Dialog Box	4-28
Pasting an Existing Dialog Box into Dialog Editor	4-28
Capturing a Dialog Box from Another Application	4-29
Opening a Dialog Box Template File	4-30
Testing an Edited Dialog Box	4-31
To test your dialog box:	4-31
Pasting a Dialog Box Template into Your Script	4-33
Exiting from Dialog Editor	4-33
Using a Custom Dialog Box in Your Script	4-34
Creating a Dialog Record	4-34
Putting Information into the Custom Dialog Box	4-35
Displaying the Custom Dialog Box	4-36
Retrieving Values from the Custom Dialog Box	4-37
Using a Dynamic Dialog Box in Your Script	4-38
Making a Dialog Box Dynamic	4-39
Menu/Tools Reference	4-41
File Menu	4-41
Edit Menu	4-42
Controls Menu	4-43
Help Menu	4-45
Pick Tool	4-45

Debugging Your Scripts	5-1
About the Debugger	5-1
Using the Debugger.....	5-1
Fabricating Event Information.....	5-2
Tracing Script Execution.....	5-3
Setting and Removing Breakpoints	5-5
Setting Breakpoints	5-5
Removing Breakpoints.....	5-7
Using a Watch Variable	5-8
Modifying the Value of a Variable.....	5-10
Running Your Programs	6-1
Running A Program.....	6-1
Suspending A Running Program	6-1
Stopping A Running Program	6-1
Appendix A - Runtime Error Messages	1-1
Introduction	A-1
Visual Basic Compatible Error Messages	A-2
Basic Control Engine-Specific Error Messages.....	A-5
Appendix B - Compiler Error Messages	2-1
Error Message List	B-1
Index	i

Introduction

CIMPLICITY Functionality

The CIMPLICITY Base System functionality -- Point Management, Alarm Management, and Data Logging facilities as well as a full-functioned User Interface -- enables CIMPLICITY users to collect data from reporting and to visualize data via lists, graphic status displays, and alarms. Standard data communications capability makes CIMPLICITY a factory floor tool that can provide services such as those listed below:

- Downtime reporting
- Production reporting
- Records of production counts at work stations
- Graphic monitoring of automatic data point values
- Fault reporting via direct point values and alarms

CIMPLICITY software's flexible system architecture and modular design allows for ease add-on of functionality

The Basic Control Engine is a product option for GE Fanuc's CIMPLICITY software. This option is fully integrated with CIMPLICITY software's Base System functionality to enhance its already powerful monitoring capability in a full range of computer integrated manufacturing environments.

The Basic Control Engine option combines the power of the CIMPLICITY event handler with a Visual Basic™ compliant language, allowing you to script and program applications and routines from the simple to the complex.

Basic Control Engine Components

The Basic Control Engine option consists of three main components

- Event Editor
- Program Editor
- Basic Control Engine

The Event Editor lets you define actions to take in response to events that occur in a process. An event can be defined as a changing point, alarm state, or even a particular time of day. One event may invoke multiple actions, or one action may be invoked by many events.

The Program Editor provides a set of sophisticated development tools that let you create programs with a Visual Basic compliant programming language. These programs can then be executed as actions in response to events. The programming language has a rich set of nearly 500 standard Basic functions, and also provides an object interface to CIMPPLICITY points, alarms, and the Status Logger, further enriching the language.

The Basic Control Engine monitors for events and executes the configured actions. The Basic Control Engine is based on a multi-threaded design that allows the system to invoke and execute multiple Visual Basic programs concurrently.

Both the Program Editor and the Visual Basic compliant language are documented in this manual. For more information on the Event Editor, see the *CIMPPLICITY HMI Event Editor Operation Manual* (GFK-1282).

Using the Basic Control Engine Language Reference

The *Basic Control Engine Language Reference Manual* is organized like a dictionary containing an entry for each language element. The language elements are categorized as follows:

<u>Category</u>	<u>Description</u>
data type	Any of the support data types, such as Integer , String , and so on.
function	Language element that takes zero or more parameters, performs an action, and returns a value
keyword	Language element that doesn't fit into any of the other categories
operator	Language elements that cause an evaluation to be performed either on one or two operands
statement	Language element that takes zero or more parameters and performs an action.
topic	Describes information about a topic rather than a language element

Each entry in the *Basic Control Engine Language Reference Manual* contains the following headings:

<u>Heading</u>	<u>Description</u>
Syntax	The syntax of the language element. The conventions used in describing the syntax are described in Chapter 1 of the <i>Basic Control Engine Language Reference Manual</i> .
Description	Contains a one-line description of that language element.
Comments	Contains any other important information about that language keyword.
Example	Contains an example of that language keyword in use. An example is provided for every language keyword.
See Also	Contains a list of other entries in the Reference section that relate either directly or indirectly to that language element.

Language Elements by Category

The following subsections list Basic Control Engine language elements by category

Arrays

ArrayDims	Return the number of dimensions of an array
ArraySort	Sort an array
Erase	Erase the elements in one or more arrays
LBound	Return the lower bound of a given array dimension
Option Base	Change the default lower bound for array declarations
ReDim	Re-establish the dimensions of an array
UBound	Return the upper bound of a dimension of an array

Clipboard

Clipboard\$ (function)	Return the content of the clipboard as a string
Clipboard\$ (statement)	Set the content of the clipboard
Clipboard.Clear	Clear the clipboard
Clipboard.GetFormat	Get the type of data stored in the clipboard
Clipboard.GetText	Get text from the clipboard
Clipboard.SetText	Set the content of the clipboard to text

Comments

'	Comment to end-of-line
Rem	Add a comment

Comparison operators

<	Less than
<=	Less than or equal to
<>	Not equal
=	Equal
>	Greater than
>=	Greater than or equal to

Controlling other programs

AppActivate	Activate an application
AppClose	Close an application
AppFind	Return the full name of an application
AppGetActive\$	Return the name of the active application
AppGetPosition	Get the position and size of an application
AppGetState	Get the window state of an application
AppHide	Hide an application
AppList	Fill an array with a list of running applications
AppMaximize	Maximize an application
AppMinimize	Minimize an application
AppMove	Move an application
AppRestore	Restore an application
AppSetState	Set the state of an application's window
AppShow	Show an application
AppSize	Change the size of an application
AppType	Return the type of an application
SendKeys	Send keystrokes to another application
Shell	Execute another application

Controlling program flow

Call	Call a subroutine
Choose	Return a value at a given index
Do...Loop	Execute a group of statements repeatedly
DoEvents (function)	Yield control to other applications
DoEvents (statement)	Yield control to other applications
End	Stop execution of a script
Exit Do	Exit a Do loop
Exit For	Exit a For loop
For...Next	Repeat a block of statement a specified number of times
GoSub	Execute at a specific label, allowing control to return later
Goto	Execute at a specific label
If...Then...Else	Conditionally execute one or more statements
IIf	Return one of two values depending on a condition

Main	Define a subroutine where execution begins
Return	Continue execution after the most recent GoSub
Select...Case	Execute one of a series of statements
Sleep	Pause for a specified number of milliseconds
Stop	Suspend execution, returning to a debugger (if present)
Switch	Return one of a series of expressions depending on a condition
While...Wend	Repeat a group of statements while a condition is True

Controlling the operating environment

Command, Command\$	Return the command line
Environ Environ\$	Return a string from the environment

Conversion

Asc	Return the value of a character
CBool	Convert a value to a Boolean
CCur	Convert a value to Currency
CDate	Convert a value to a Date
CDbl	Convert a value to a Double
Chr, Chr\$	Convert a character value to a string
CInt	Convert a value to an Integer
CLng	Convert a value to a Long
CSng	Convert a value to a Single
CStr	Convert a value to a String
CVar	Convert a value to a Variant
CVDate	Convert a value to a Date
CVErr	Convert a value to an error
Hex, Hex\$	Convert a number to a hexadecimal string
IsDate	Determine if an expression is convertible to a date
IsError	Determine if a variant contains a user-defined error value
IsNumeric	Determine if an expression is convertible to a number
Oct, Oct\$	Convert a number to an octal string
Str, Str\$	Convert a number to a string
Val	Convert a string to a number

Data types

Boolean	Data type representing True or False values
Currency	Data type used to hold monetary values
Date	Data type used to hold dates and times
Double	Data type used to hold real number with 15-16 digits of precision
Integer	Data type used to hold whole numbers with 4 digits of precision
Long	Data type used to hold whole numbers with 10 digits of precision
Object	Data type used to hold OLE automation objects
Single	Data type used to hold real number with 7 digits of precision
String	Data type used to hold sequences of characters
Variant	Data type that holds a number, string, or OLE automation objects

Database

SQLBind	Specify where to place results with SQLRetrieve
SQLClose	Close a connection to a database
SQLError	Return error information when an SQL function fails
SQLExecQuery	Execute a query on a database
SQLGetSchema	Return information about the structure of a database
SQLOpen	Establishes a connection with a database
SQLRequest	Run a query on a database
SQLRetrieve	Retrieve all or part of a query
SQLRetrieveToFile	Retrieve all or part of a query, placing results in a file

Date/time

Date, Date\$ (functions)	Return the current date
Date, Date\$ (statements)	Change the system date
DateAdd	Add a number of date intervals to a date
DateDiff	Subtract a number of date intervals from a date
DatePart	Return a portion of a date
DateSerial	Assemble a date from date parts
DateValue	Convert a string to a date

Day	Return the day component of a date value
Hour	Return the hour part of a date value
Minute	Return the minute part of a date value
Month	Return the month part of a date value
Now	Return the date and time
Second	Return the seconds part of a date value
Time, Time\$ (functions)	Return the current system time
Time, Time\$ (statements)	Set the system time
Timer	Return the number of elapsed seconds since midnight
TimeSerial	Assemble a date/time value from time components
TimeValue	Convert a string to a date/time value
Weekday	Return the day of the week of a date value
Year	Return the year part of a date value

DDE

DDEExecute	Execute a command in another application
DDEInitiate	Initiate a DDE conversation with another application
DDEPoke	Set a value in another application
DDERequest, DDERequest\$	Return a value from another application
DDESend	Establish a DDE conversation, then sets a value in another application
DDETerminate	Terminate a conversation with another application
DDETerminateAll	Terminate all conversations
DDETimeout	Set the timeout used for non-responding applications

Error handling

Erl	Return the line with the error
Err (function)	Return the error that caused the current error trap
Err (statement)	Set the value of the error
Error	Simulate a trappable runtime error
Error, Error\$	Return the text of a given error
On Error	Trap an error
Resume	Continue execution after an error trap

File I/O

Close	Close one or more files
EOF	Determine if the end-of-file has been reached
FreeFile	Return the next available file number
Get	Read data from a random or binary file
Input#	Read data from a sequential file into variables
Input, Input\$	Read a specified number of bytes from a file
Line Input#	Read a line of text from a sequential file
Loc	Return the record position of the file pointer within a file
Lock	Lock a section of a file
Lof	Return the number of bytes in an open file
Open	Open a file for reading or writing
Print #	Print data to a file
Put	Write data to a binary or random file
Reset	Close all open files
Seek (function)	Return the byte position of the file pointer within a file
Seek (statement)	Set the byte position of the file pointer within a file
UnLock	Unlock part of a file
Width#	Specify the line width for sequential files
Write #	Write data to a sequential file

File system

ChDir	Change the current directory
ChDrive	Change the current drive
CurDir, CurDir\$	Return the current directory
Dir, Dir\$	Return files in a directory
DiskDrives	Fill an array with valid disk drive letters
DiskFree	Return the free space on a given disk drive
FileAttr	Return the mode in which a file is open
FileCopy	Copy a file
FileDateTime	Return the date and time when a file was last modified
FileDirs	Fill an array with a subdirectory list
FileExists	Determine if a file exists
FileLen	Return the length of a file in bytes

FileList	Fill an array with a list of files
FileParse\$	Return a portion of a filename
GetAttr	Return the attributes of a file
Kill	Delete files from disk
MkDir	Create a subdirectory
Name	Rename a file
RmDir	Remove a subdirectory
SetAttr	Change the attributes of a file

Financial

DDB	Return depreciation of an asset using double-declining balance method
Fv	Return the future value of an annuity
IPmt	Return the interest payment for a given period of an annuity
IRR	Return the internal rate of return for a series of payments and receipts
MIRR	Return the modified internal rate of return
NPer	Return the number of periods of an annuity
Npv	Return the net present value of an annuity
Pmt	Return the payment for an annuity
PPmt	Return the principal payment for a given period of an annuity
Pv	Return the present value of an annuity
Rate	Return the interest rate for each period of an annuity
Sln	Return the straight-line depreciation of an asset
SYD	Return the Sum of Years' Digits depreciation of an asset

Getting information from Basic Control Engine

Basic.Capability	Return capabilities of the platform
Basic.Eoln\$	Return the end-of-line character for the platform
Basic.FreeMemory	Return the available memory
Basic.HomeDir\$	Return the directory where Basic Control Engine is located
Basic.OS	Return the platform id
Basic.PathSeparator\$	Return the path separator character for the platform
Basic.Version\$	Return the version of Basic Control Engine

INI Files

ReadIni\$	Read a string from an INI file
ReadIniSection	Read all the item names from a given section of an INI file
WriteIni	Write a new value to an INI file

Logical/binary operators

And	Logical or binary conjunction
Eqv	Logical or binary equivalence
Imp	Logical or binary implication
Not	Logical or binary negation
Or	Logical or binary disjunction
Xor	Logical or binary exclusion

Math

Abs	Return the absolute value of a number
Atn	Return the arc tangent of a number
Cos	Return the cosine of an angle
Exp	Return e raised to a given power
Fix	Return the integer part of a number
Int	Return the integer portion of a number
Log	Return the natural logarithm of a number
Random	Return a random number between two values
Randomize	Initialize the random number generator
Rnd	Generate a random number between 0 and 1
Sgn	Return the sign of a number
Sin	Return the sine of an angle
Sqr	Return the square root of a number
Tan	Return the tangent of an angle

Miscellaneous

()	Force parts of an expression to be evaluated before others
_	Line continuation
Beep	Make a sound
Inline	Allow execution or interpretation of a block of text

Numeric operators

*	Multiply
+	Add
-	Subtract
/	Divide
\	Integer divide
^	Power
Mod	Remainder

Objects

CreateObject	Instantiate an OLE automation object
GetObject	Return an OLE automation object from a file, or returns a previously instantiated OLE automation object
Is	Compare two object variables
Nothing	Value indicating no valid object

Parsing

Item\$	Return a range of items from a string
ItemCount	Return the number of items in a string
Line\$	Retrieve a line from a string
LineCount	Return the number of lines in a string
Word\$	Return a sequence of words from a string
WordCount	Return the number of words in a string

Predefined dialogs

AnswerBox	Display a dialog asking a question
AskBox\$	Display a dialog allowing the user to type a response
AskPassword\$	Display a dialog allowing the user to type a password
InputBox, InputBox\$	Display a dialog allowing the user to type a response
MsgBox (function)	Display a dialog containing a message and some buttons
MsgBox (statement)	Display a dialog containing a message and some buttons
OpenFilename\$	Display a dialog requesting a file to open
SaveFilename\$	Display a dialog requesting the name of a new file
SelectBox	Display a dialog allowing selection of an item from an array

Printing

Print	Print data to the screen
Spc	Print a number of spaces within a Print statement
Tab	Used with Print to print spaces up to a column position

Procedures

Declare	An external routine or a forward reference
Exit Function	Exit a function
Exit Sub	Exit a subroutine
Function...End Function	Create a user-defined function
Sub...End Sub	Create a user-defined subroutine

String operators

&	Concatenate two strings
Like	Compare a string against a pattern

Strings

Format, Format\$	Return a string formatted to a given specification
InStr	Return the position of one string within another
LCase, LCase\$	Convert a string to lower case
Left, Left\$	Return the left portion of a string
Len	Return the length of a string or the size of a data item
LSet	Left align a string or user-defined type within another
LTrim, LTrim\$	Remove leading spaces from a string
Mid, Mid\$ (functions)	Return a substring from a string
Mid, Mid\$ (statements)	Replace one part of a string with another
Option Compare	Change the default comparison between text and binary
Option CStrings	Allow interpretation of C-style escape sequences in strings
Right, Right\$	Return the right portion of a string
RSet	Right align a string within another
RTrim, RTrim\$	Remove trailing spaces from a string
Space, Space\$	Return a string of spaces
StrComp	Compare two strings
String, String\$	Return a string consisting of a repeated character
Trim, Trim\$	Trim leading and trailing spaces from a string
UCase, UCase\$	Return the upper case of a string

User dialogs

Begin Dialog	Begin definition of a dialog template
CancelButton	Define a Cancel button within a dialog template
CheckBox	Define a combo box in a dialog template
ComboBox	Define a combo box in a dialog template
Dialog (function)	Invoke a user-dialog, returning which button was selected
Dialog (statement)	Invoke a user-dialog
DlgControlId	Return the id of a control in a dynamic dialog
DlgEnable (function)	Determine if a control is enabled in a dynamic dialog
DlgEnable (statement)	Enable or disables a control in a dynamic dialog
DlgFocus (function)	Return the control with the focus in a dynamic dialog
DlgFocus (statement)	Set focus to a control in a dynamic dialog
DlgListBoxArray (function)	Set the content of a list box or combo box in a dynamic dialog
DlgListBoxArray (statement)	Set the content of a list box or combo box in a dynamic dialog
DlgSetPicture	Set the picture of a control in a dynamic dialog
DlgText (statement)	Set the content of a control in a dynamic dialog
DlgText\$ (function)	Return the content of a control in a dynamic dialog
DlgValue (function)	Return the value of a control in a dynamic dialog
DlgValue (statement)	Set the value of a control in a dynamic dialog
DlgVisible (function)	Determine if a control is visible in a dynamic dialog
DlgVisible (statement)	Set the visibility of a control in a dynamic dialog
DropListBox	Define a drop list box in a dialog template
GroupBox	Define a group box in a dialog template
ListBox	Add a list box to a dialog template
OKButton	Add an OK button to a dialog template
OptionButton	Add an option button to a dialog template
OptionGroup	Add an option group to a dialog template
Picture	Add a picture control to a dialog template
PictureButton	Add a picture button to a dialog template
PushButton	Add a push button to a dialog template
Text	Add a text control to a dialog template
TextBox	Add a text box to a dialog template

Variables and constants

=	Assignment
Const	Define a constant
DefBool	Set the default data type to Boolean
DefCur	Set the default data type to Currency
DefDate	Set the default data type to Date
DefDbf	Set the default data type to Double
DefInt	Set the default data type to Integer
DefLng	Set the default data type to Long
DefObj	Set the default data type to Object
DefSng	Set the default data type to Single
DefStr	Set the default data type to String
DefVar	Set the default data type to Variant
Dim	Declare a local variable
Global	Declare variables for sharing between scripts
Let	Assign a value to a variable
Private	Declare variables accessible to all routines in a script
Public	Declare variables accessible to all routines in all scripts
Set	Assign an object variable
Type	Declare a user-defined data type

Variants

IsEmpty	Determine if a variant has been initialized
IsError	Determine if a variant contains a user-defined error
IsMissing	Determine if an optional parameter was specified
IsNull	Determine if a variant contains valid data
IsObject	Determine if an expression contains an object
VarType	Return the type of data stored in a variant

About The Program Editor

Functional Overview

The Program Editor utilizes an embedded Basic language that is syntactically compatible with Microsoft's Visual Basic for Applications™. This language provides the rich Basic command set, in addition to CIMPLICITY software specific extensions. Both the command set and extensions are documented in the remainder of this document.

The Program Editor, provides an integrated development and debug environment, including the following features:

- Wizards
- Watch Window
- Quickwatch Object Inspector
- Trace Window
- Breakpoints
- Step In, Step Over, Stop
- Set Variable to value
- Interscript calls
- Multiple threads of execution.
- Dialog Editor
- On Line Help

Wizards

You can use the Wizards to automatically generated procedure or function calls. You fill in the blanks, and the wizard generates the code. The wizards are located on the Tools toolbar. The Tools toolbar default location is on the left side of the screen.

You can use the wizards to generate:

- Point access functions
- Alarm manipulation functions
- Error logging functions that send messages to the CIMPLICITY Status Log

You can also use the right mouse button to bring up a popup menu containing the wizards while you are editing a document.

You can also create points on your devices while you are in the wizards.

Watch Window

You can use the watch feature to place commonly used variables into a watch windows. When you step over a line, or when execution of the program pauses, the values of the variables in the watch window are updated.

Currently, the watch window does *not* support object inspection, user defined structures, or arrays (however, you may view single elements of an array , using the <array>[<element>] syntax. These features are available from the QuickWatch inspector.

The watch window size can be adjusted by dragging the separator.

QuickWatch Object Inspector

You can use the QuickWatch object inspector to inspect variables, objects, arrays and user defined types (UDT's.) Simply select a variable in the program and press the **QuickWatch** toolbar button or select **QuickWatch** from the right mouse button popup menu and you are on your way. The inspector utilizes a tree structure to help you navigate through your objects and structures.

Trace Window

You can resize the trace window in the lower part of the document window as needed by dragging the separator. The trace window receives the results of all **Trace** commands executed by your program. Controls are provided to toggle tracing on/off and to clear the trace window. The trace window is useful for diagnostics and debugging.

Breakpoints

You can set breakpoints in any script you are viewing. When an executing script reaches the line, execution will pause and you may inspect variables or continue execution. Up to 255 breakpoints are supported per script.

Step In/Step Over

Step In and **Step Over** provide tools to single step through your script's execution. **Step Over** steps over the current line while **Step In** will step into the current any function or procedure within the line. When stepping into another script, the contents of the debugger window are replaced with the called script. You can view the "Call Stack" to see the stack of function calls which brought you to the current point of execution.

InterScript Calls

As applications grow in size, you may find it useful to have "libraries" of commonly used Basic procedures and functions. Promoting code re-use will simplify maintenance and development.

To use a library, simply load the script into a window in the editor; the code is now available for all other scripts in the application. You can set breakpoints in the library and when another script reaches that execution point it will pause in its own window at the line of code.

Multiple Threads of Execution

You can run many scripts at the same time from within the development environment. Each window is a thread of execution, and you may start and stop windows as needed.

Dialog Editor

A integrated dialog editor is provided to allow you to create custom User Interface applications.

On Line Help

A full API reference for the Basic Control Engine language is available on-line, by pressing the F1 key within the application.

Getting Started

To start the Program Editor, you can double-click on the **Program Editor** icon in the CIMPLICITY Common cabinet, or in your project group.

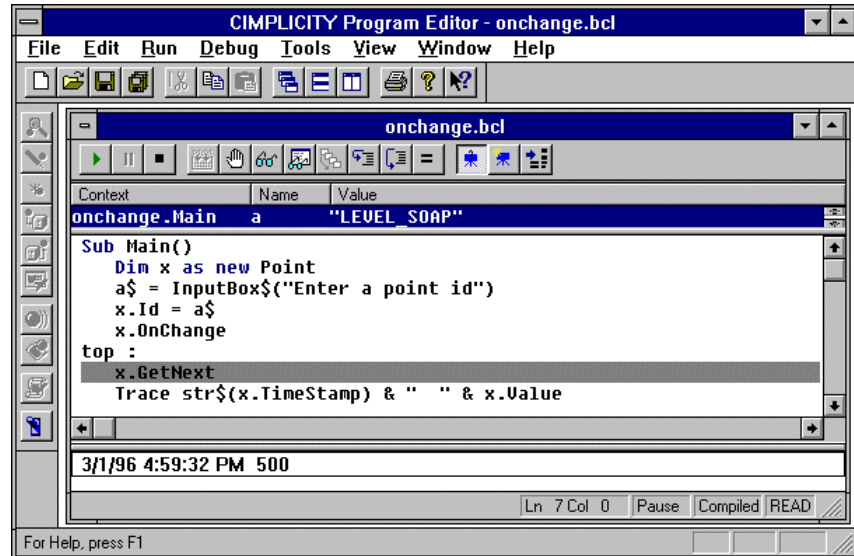


Program Editor

The Program Editor window is displayed.

Program Editor Window Components

When you are using the Program Editor, the window looks similar to this:



The main window contains the Menu bar, and the Standard and Tools toolbars. Each script window contains its own Debug toolbar.

A script window contains three areas. These are, in order from top to bottom:

- Watch window
- Script window
- Trace window

These areas can be resized by dragging the separators.

Program Editor Menu Functions

You can use the menu options to open, close, print, and compile files, to edit a file, to run a file, to debug a file, to access tools, to view status and toolbars, to arrange windows, and access help

The File Menu

When you select the **File** menu, the following drop-down list is displayed:

N <u>ew</u>	
O <u>pen...</u>	Ctrl+O
S <u>ave</u>	Ctrl+S
S <u>ave A</u> ll	
S <u>ave A</u> s...	
<hr/>	
P <u>rint...</u>	Ctrl+P
P <u>rint P</u> review	
<hr/>	
1 G:\C I MPL I CITY\...\onchange.bcl	
2 G:\C I MPL I CITY\...\search.bcl	
3 G:\C I MPL I CITY\...\dialogs2.bcl	
4 G:\C I MPL I CITY\...\dialogs.bcl	
<hr/>	
C <u>ompile</u>	
C <u>reate P</u> rogram...	
<hr/>	
E <u>x</u> it	

The **File** menu functions are:

New	Creates a new document for the Program Editor.
Open	Opens an existing document for the Program editor.
Save	Saves the active document.
Save All	Saves all the open files in the Program Editor.
Print	Prints the active document
Print Preview	Displays the active document as it will be printed
Recent File	Displays the list of most recently accessed files.
Compile	Compiles the active document.
Create Program	Creates a new document for the Program Editor.
Exit	Exits the Program Editor.

The Edit Menu

When you select the **Edit** menu, the following drop-down list is displayed:

U ndo	Ctrl+Z
C ut	Ctrl+X
C opy	Ctrl+C
P aste	Ctrl+V
C lear	Delete
F ind...	Ctrl+F
F ind N ext	F3
R eplace...	
G oto Line	Ctrl+G
I nsert Dialog...	
E dit Dialog...	
F ont...	Ctrl+D
O ptions...	

The **Edit** menu functions are:

Undo	Undoes the last action.
Cut	Cuts the selection and puts it on the Clipboard.
Copy	Copies the selection and puts it on the Clipboard
Paste	Inserts Clipboard contents.
Clear	Deletes the selection.
Find	Finds user-identified text in the document.
Find Next	Finds next occurrence of user-identified text in the document.
Replace	Replaces user-identified text with new text.
Goto Line	Goes to the selected line.
Insert Dialog	Inserts a dialog box.
Edit Dialog	Edits an inserted dialog box.
Font	Selects a font.
Options	Sets string and stack space.

The Run Menu

When you select the **Run** menu, the following drop-down list is displayed:

S tart F5
B reak
E nd

The **Run** menu options are:

Start	Run the program
Break	Break executing program
End	End the running or paused program

The Debug Menu

When you select the **Debug** menu, the following drop-down list is displayed:

A dd Watch... Shift+F9
D elete Watch
Q uickWatch...
M odify...
S tep F8
S tep I nto Shift+F8
C all Stack...
<input checked="" type="checkbox"/> T race
C lear Trace
T oggle Breakpoint F9
C lear all Breakpoints
S et N ext Statement
S et C ommand L ine...
R eset Public V ariables

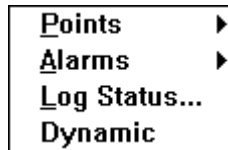
The **Debug** menu options are:

Add Watch	Displays the <i>Add Watch</i> dialog box, in which you can specify the name of a script variable
Delete Watch	Deletes the watch from the selected variable
Quick Watch	
Modify	Modifies the value of a variable.
Step	Executes the next line of the script. If the line calls a procedure, the called procedure is run in its entirety.
Step Into	Executes the next line of the script. If the line calls a procedure, the next line to execute will be the first line of the called procedure.
Call Stack	Displays the stack of current calls.
Trace/	Enables/disables output to the Trace window

Clear Trace	
Toggle Breakpoint	Toggles a breakpoint in the script
Clear all Breakpoints	Clears all breakpoints from the script
Set Next statement	Sets the next statement to be executed in a paused program to the currently selected line.
Set Command Line	Set the command line for the script. This can be retrieved via the basic Command\$ parameter. The Basic Control Engine will pass the Event & Action which caused the script to be run. See BCE Manual
Reset Public Variables	Reset's the contents of public and private variables to an empty state.

The Tools Menu

When you select the **Tools** menu, the following drop-down list is displayed:

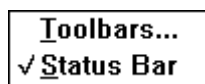


The **Tools** menu options are:

Points	Displays a submenu that lets you browse for points, edit a point, and create a new point. You can also use this menu item to include Setpoints, Get Points, and create local variables in the program.
Alarms	Displays a submenu that lets you generate or update alarms in the program.
Log Status	Displays a dialog box that lets you generate messages for the Status Log.
Dynamic	Toggles Dynamic Configuration of points, alarm, etc.

The View Menu

When you select the **View** menu, the following drop-down list is displayed:

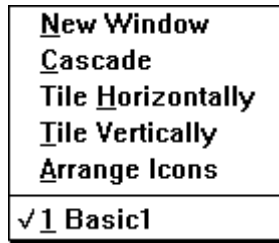


The **View** menu options are

Toolbars	Displays the list of available toolbars. You can toggle the display of each toolbar.
Status Bar	Toggles the display of the Status Bar at the bottom of program windows.

The Window Menu

When you select the **Window** menu, the following drop-down list is displayed:



The **Window** menu options are:

New Window	Opens a new window.
Cascade	Arranges the windows so that they overlap.
Tile Horizontally	Tiles the windows horizontally.
Tile Vertically	Tiles the windows vertically.
Arrange Icons	Arranges the program icons in the Program Editor window.
Current Programs	Displays the list of current programs.

The Help Menu

When you select the **Help** menu, the following drop-down list is displayed:



The **Help** menu options are:














Index	Displays the main Help window for the Program Editor.
Using Help	Displays the main Help window for Microsoft Windows.
About...	Displays program information, version number, and copyright for the Program Editor

Program Editor Toolbars

The main window contains the **Standard** and **Tools** Toolbars. In addition, each script window contains an **Application** toolbar.











Standard Toolbar

The **Standard** Toolbar functions are:

	New	Create a new document.
	Open	Open an existing document
	Save	Save the active document
	Save All	Save all the open files
	Cut	Cut the selection and put it on the Clipboard
	Copy	Copy the selection and put it on the Clipboard
	Paste	Insert Clipboard contents
	Cascade Windows	Arrange windows so they overlap
	Tile Horizontally	Arrange windows as non-overlapping tiles
	Tile Vertically	Arrange windows as non-overlapping tiles
	Print	Print the active document
	About	Display program information, version number, and copyright
	Help	Display Help for clicked on buttons, menus, and windows















Tools Toolbar

The **Tools** Toolbar functions are:

	Browse Point	Browse for Points
	Edit Point	Edit Point ID
	New Point	Create a new Point
	Get Point	Get Point Value
	Set Point	Set a Point
	Dim Point	Dimension a Point Object
	Gen Alarm	Generate an Alarm
	Update Alarm	Update an Alarm
	Log Status	Log a status message
	Dynamic	Toggle Dynamic Configuration

Application Toolbar

The **Application** Toolbar functions are:

	Start	Start or continue execution
	Break	Interrupt execution
	End	End execution
	Compile	Compile the document
	Toggle Breakpoint	Set or clear a breakpoint
	QuickWatch	Quickwatch a variable
	Add Watch	Add a watch to a variable
	Call Stack	Display call stack
	Step Into	Step into the current line
	Step	Step over the current line
	Modify	Modify the value of a variable
	Toggle Trace	Enable/Disable Tracing
	Clear Trace	Clear the contents of the trace window
	Command Line	Set the command line for the script

Program Editor Shortcut Keys

You can use the following shortcut keys to initiate commonly used functions:

Ctrl+N	Creates a new document
Ctrl+O	Opens an existing document
Ctrl+S	Saves the active document
Ctrl+P	Prints the active document
Ctrl+Z	Undoes the last edit action
Ctrl+X	Cuts the selection and puts it on the Clipboard
Ctrl+C	Copies the selection and puts it on the Clipboard
Ctrl+V	Inserts the contents of the Clipboard
Delete	Cuts the selection
Ctrl+F	Opens the <i>Find</i> dialog box
F3	Finds the next occurrence of the string in the <i>Find</i> dialog box
Ctrl+G	Opens the <i>Go To Line</i> dialog box
Ctrl+D	Opens the <i>Font</i> dialog box
Shift+F9	Opens the <i>Add Watch</i> dialog box
F8	Steps to the next line in the script
Shift+F8	Steps to the next line in the script. If it is a procedure call, the next line is the first line in the procedure.
F9	Toggles a breakpoint for the debugger
F5	Starts running a script

Setting String and Stack Space

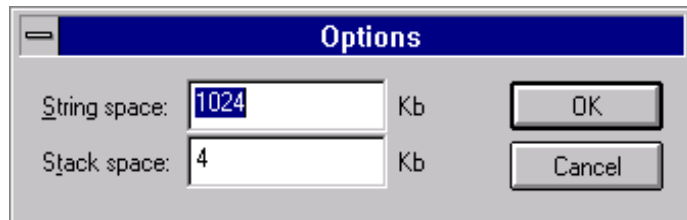
The Basic Control Engine has two regions of memory, String Space, and Stack Space.

- String Space holds all string variables, arrays, and public data. Default String Space size is 1 MB (1024 KB).
- Stack Space holds all local variables and intermediate values. Default Stack Space size is 4 KB.

You can change the size of either of these spaces. The changes apply to all Basic Control Engine scripts that run as executables within the Program Editor or from the Event Manager.

To change String Space and Stack Space sizes:

1. Select **Options** from the *Edit* menu.
2. The Options dialog opens.



3. Enter the number of kilobytes of String Space in **String Space**.
4. Enter the number of kilobytes of Stack Space in **Stack Space**.
5. Select **OK**.
6. When the message

You must stop and restart BASIC for the changes to take effect.

appears, select **OK**.

Also, note the following:

- You can substantially reduce your stack usage by explicitly defining the types of variables (in other words, don't use variants).
- Recursive routines have an impact on Stack Space.
- If you use arrays or arrays of User Defined Types, allocation occurs in the String Space which may alleviate some stack usage.

Editing Programs

About Editing Programs

Although, in some respects, editing code with the Program Editor is like editing regular text with a word-processing program, the Program Editor also has certain capabilities specifically designed to help you edit your code.

This section describes how to move around within your script, select and edit text, add comments to your script, break long statements across multiple lines, search for and replace selected text, and perform a syntax check of your script. The section ends with a brief discussion of editing dialog box templates, which is explained in much more detail in Chapter 4.

Navigating within a Script

The lists of keyboard shortcuts in the preceding section contain a group of navigating shortcuts, which you can use to move the insertion point around within your script. When you move the insertion point with a keyboard shortcut, Program Editor scrolls the new location of the insertion point into view if it is not already displayed.

You can also reposition the insertion point with the mouse and the Goto Line command, as explained below.

Program Editor differs from most word-processing programs in that it allows you to place the insertion point anywhere within your script, including in "empty spaces." (Empty spaces are areas within the script that do not contain text, such as a tabs expanded space or the area beyond the last character on a line.)

Moving the Insertion Point with the Mouse

This approach is especially fast if the area of the screen to which you want to move the insertion point is currently visible.

To move the insertion point with the mouse:

1. Use the scroll bars at the right and bottom of the display to scroll the target area of the script into view if it is not already visible.
2. Place the mouse pointer where you want to position the insertion point.

3. Click the left mouse button. The insertion point is repositioned.

Note

When you scroll the display with the mouse, the insertion point remains in its original position until you reposition it with a mouse click. If you attempt to perform an editing operation when the insertion point is not in view, Program Editor automatically scrolls the insertion point into view before performing the operation.

Moving the Insertion Point to a Specified Line

This approach is especially fast if the area of the screen to which you want to move the insertion point is not currently visible but you know the number of the target line.

To move the insertion point to a specified line in your script:

1. Press **F4**. Program Editor displays the *Goto Line* dialog box.



2. Enter the number of the line in your script to which you want to move the insertion point.
3. Click the **OK** button or press **Enter**.

The insertion point is positioned at the start of the line you specified. If that line was not already displayed, Program Editor scrolls it into view.

Note

The insertion point cannot be moved so far below the end of a script as to scroll the script entirely off the display. When the last line of your script becomes the first line on your screen, the script will stop scrolling, and you will be unable to move the insertion point below the bottom of that screen.

Inserting Text

In Program Editor, inserting text and other characters such as tabs and line breaks works about the same way as it does in a word-processing program: you position the insertion point at the desired location in the script and start typing.

However, as noted in the preceding subsection, Program Editor lets you position the insertion point in "empty spaces," which means that you can also insert text into empty spaces—a feature that comes in handy when you want to insert a comment in the space beyond the end of a line in your script. (Adding comments to your script is discussed later in this section.) When you insert characters beyond the end of a line, the space between the insertion point and the last character on the line is backfilled with tab characters.

Another way in which Program Editor differs from word-processing programs is that in Program Editor, text does not wrap. If you keep entering text on a given line, eventually you will reach a point at which you can enter no more text on that line. Therefore, *you* control the line breaks by pressing **Enter** when you want to insert a new line in your script. The effect of pressing **Enter** depends on where the insertion point is located at the time:

- If you press **Enter** with the insertion point at or beyond the end of a line, a new line is inserted after the current line.
- If you press **Enter** with the insertion point at the start of a line, a new line is inserted before the current line.
- If you press **Enter** with the insertion point within a line, the current line is broken into two lines at that location.

If you press **Tab**, a tab character is inserted at the location of the insertion point, which causes text after the tab to be moved to the next tab position. If you insert new text within a tab's expanded space, the text that originally appeared on that line is moved to the next tab position each time the new text that you are entering reaches the start of another tab position.

Selecting Text

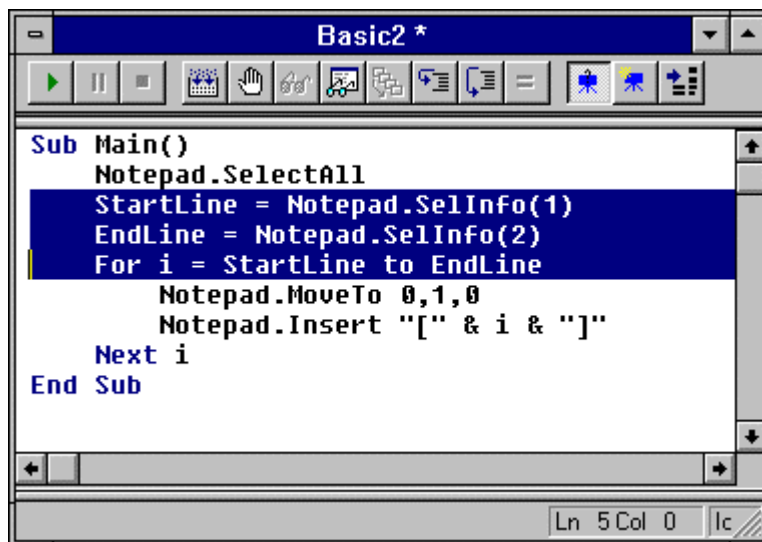
You can use either the mouse or the keyboard to select text and other characters in your script. Regardless of which method you use, you should be aware that in Program Editor, you can select either a portion of one line or a series of whole lines, but you cannot select a portion of one line plus one or more whole lines. When you are selecting multiple lines and start or end your selection partway through a line, Program Editor automatically extends the selection to include the entire starting and ending lines.

Selecting Text With the Mouse

To select the text in your script, place the mouse pointer where you want your selection to begin, then do one of the following:

- While pressing the left mouse button, drag the mouse until you reach the end of your selection, and release the mouse button.
- While pressing **Shift**, place the mouse pointer where you want your selection to end and click the left mouse button.

The selected text is highlighted on your display, as shown in this example:



The screenshot shows a window titled "Basic2 *" with a toolbar at the top. The main area contains a code block with the following text: `Sub Main()
Notepad.SelectAll
StartLine = Notepad.SelInfo(1)
EndLine = Notepad.SelInfo(2)
For i = StartLine to EndLine
 Notepad.MoveTo 0,1,0
 Notepad.Insert "[" & i & "]"
Next i
End Sub`. The lines from `StartLine = Notepad.SelInfo(1)` to `Next i` are highlighted in blue. The status bar at the bottom right shows "Ln 5 Col 0" and "Ic".

Another way to select one or more whole lines with the mouse is to start by placing the mouse pointer in the left margin beside the first line you want to select. The pointer becomes a reverse arrow, which points toward the line of text. Click the left mouse button to select a single line; press the left mouse button and drag up or down to select multiple lines.

Selecting Text With the Keyboard

Here's how to use keyboard shortcuts to select text in your script.

1. Place the insertion point where you want your selection to begin.
2. While pressing **Shift**, use one of the navigating keyboard shortcuts to extend the selection to the desired ending point.

The selected text is highlighted on your display.

Note

When you intend to select an entire single line of text in your script, it is important to remember to extend your selection far enough to include the hidden end-of-line character, which is the character that inserts a new line in your script.

Selecting A Line With the Keyboard

Here's how to use the keyboard to select one or more whole lines in your script.

1. Place the insertion point at the beginning of the line you want to select.
2. Press **Shift + Down** arrow. The entire line, including the end-of-line character, is selected.
3. To extend your selection to include additional whole lines of text, repeat step 2.

Once you have selected text within your script, you can perform a variety of other editing operations on it, including deleting the text, placing it on the Clipboard (either by cutting the text or copying it), and pasting it.

Deleting Text

When you delete material, it is removed from your script without being placed on the Clipboard.

Deleting Selected Text

Here's how to remove one or more characters, selected text, or entire lines from your script.

- To remove a single character to the left of the insertion point, press **Backspace** once; to remove a single character to the right of the insertion point, press **Delete** once. To remove multiple characters, hold down **Backspace** or **Delete**.
- To remove text that you have selected, press **Backspace** or **Delete**.
- To remove an entire line, place the insertion point in that line and press **Ctrl+Y**.

Deleting Line Breaks

Here's how to remove an unwanted line break from your script.

1. Place the insertion point after the last character on the current line.
2. Press **Delete** once to delete the hidden end-of-line character.

The current line and the following line are combined.

-Or-

1. Place the insertion point at the start of a line.
2. Press Backspace.

The current line and the preceding line are combined.

Note

If any spaces were entered at the end of the current line, you may have to press **Delete** one or more additional times to remove these hidden characters first before you can delete the end-of-line character.

Pressing **Backspace** with the insertion point at the start of a line has no effect—that is, it will not combine the current line with the preceding line.

Cutting and Copying Text

You can place material from your script on the Clipboard by either cutting it or copying it.

Cutting A Selection

Here's how to place on the Clipboard text that you have cut from your script.

- Press **Ctrl+X**.

The selection is removed from your script and placed on the Clipboard.

Copying A Selection

Here's how to place on the Clipboard text that you have copied from your script.

- Press **Ctrl+C**.

The selection remains in your script, and a copy of it is placed on the Clipboard.

Pasting Text

Once you have cut or copied material to the Clipboard, here's how to paste it into your script at another location.

1. Position the insertion point where you want to place the contents of the Clipboard.
2. Press **Ctrl+V**.

The contents of the Clipboard appear at the location of the insertion point.

If you wish to delete a block of text and insert the contents of the Clipboard in its place, you can combine the two operations by first selecting the text you want to remove and then pressing **Ctrl+V** to replace it with the contents of the Clipboard.

Undoing Editing Operations

You can undo editing operations that produce a change in your script, including:

- The insertion of a series of characters
- The insertion of a block of text from the Clipboard
- The deletion of a series of characters
- The deletion or cutting of a block of text

You cannot undo operations that don't produce any change in your script, such as moving the insertion point, selecting text, and copying material to the Clipboard.

Here's how to reverse the effect of the preceding editing operation.

- Press **Ctrl+Z**.

Your script is restored to the way it looked before you performed the editing operation.

You can undo the last 100 operations.

Adding Comments to Your Script

You can add comments to your script to remind yourself or others of how your code works. Comments are ignored when your script is executed.

The apostrophe symbol (') is used to indicate that the text from the apostrophe to the end of the line is a comment.

Adding A Full Line Comment

Here's how to designate an entire line as a comment.

1. Type an apostrophe (') at the start of the line.
2. Type your comment following the apostrophe.

When your script is run, the presence of the apostrophe at the start of the line will cause the entire line to be ignored.

Adding An End of Line Comment

Here's how to designate the last part of a line as a comment.

1. Position the insertion point in the empty space beyond the end of the line of code.
2. Type an apostrophe (').
3. Type your comment following the apostrophe.

When your script is run, the code on the first portion of the line will be executed, but the presence of the apostrophe at the start of the comment will cause the remainder of the line to be ignored.

Although you can place a comment at the end of a line containing executable code, you cannot place executable code at the end of a line containing a comment because the presence of the apostrophe at the start of the comment will cause the balance of the line (including the code) to be ignored.

Breaking a Statement across Multiple Lines

By default, in Program Editor, a single statement can extend only as far as the right margin, and each line break represents a new statement. However, you can override this default if you want to break a long statement across two or more lines.

Here's how to indicate that two or more lines of code should be treated as a single statement when your script is run.

1. Type the statement on multiple lines, exactly the way you want it to appear.
2. Place the insertion point at the end of the first line in the series.
3. Press the spacebar once to insert a single space.
4. Type an underscore (_).

Note

The underscore is the *line-continuation character*, which indicates that the statement continues on the following line.

5. Repeat steps 2–4 to place a line-continuation character at the end of each line in the series except the last.

When you run your script, the code on this series of lines will be executed as a single statement, just as if you had typed the entire statement on the same line.

Searching and Replacing

Program Editor makes it easy to search for specified text in your script and automatically replace instances of specified text.

Finding Text in Your Script

Here's how to locate instances of specified text quickly anywhere within your script.

1. Move the insertion point to where you want to start your search. (To start at the beginning of your script, press **Ctrl+Home**.)
2. Press **Ctrl+F**. Program Editor displays the *Find* dialog box:



3. In the **Find What** field, specify the text you want to find.
4. Select the **Match Case** check box if you want the search to be case-sensitive. Otherwise, the search will be case-insensitive.
5. Click the **Find Next** button or press **Enter**. The *Find* dialog box remains displayed, and Program Editor either highlights the first instance of the specified text or indicates that it cannot be found.
6. If the specified text has been found, repeat step 5 to search for the next instance of it.

Note

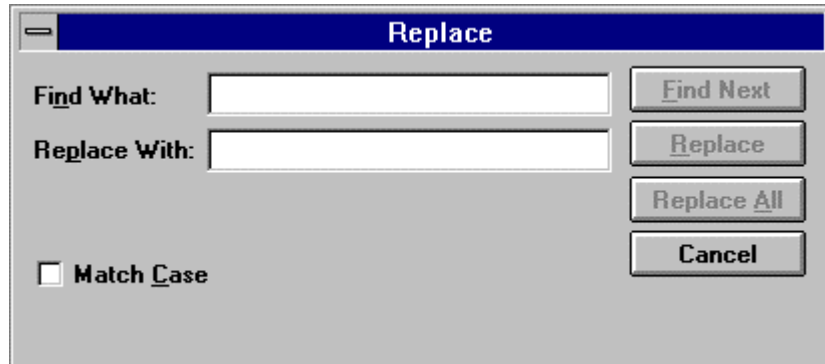
If the *Find* dialog box blocks your view of an instance of the specified text, you can move the dialog box out of your way and continue with your search.

You can also click the **Cancel** button, which removes the *Find* dialog box while maintaining the established search criteria, and then press **F3** to find successive occurrences of the specified text. (If you press **F3** when you have not previously specified text for which you want to search, Program Editor displays the *Find* dialog box so you can specify the desired text.)

Replacing Text in Your Script

Here's how you can automatically replace either all instances or selected instances of specified text.

1. Move the insertion point to where you want to start the replacement operation. (To start at the beginning of your script, press **Ctrl+Home**.)
2. Choose the **Replace** command from the **Search** menu. Program Editor displays the *Replace* dialog box:



3. In the **Find What** field, specify the text you want to replace.
4. In the **Replace With** field, specify the replacement text.
5. Select the **Match Case** check box if you want the replacement operation to be case-sensitive. Otherwise, it will be case-insensitive.
6. To replace all instances of the specified text, click the **Replace All** button. Program Editor either replaces the specified text throughout your script and indicates the number of occurrences it has changed, or it indicates that the specified text cannot be found.
7. To replace selected instances of the specified text, click the **Find Next** button. Program Editor either highlights the first instance of the specified text or indicates that it cannot be found.
8. If the specified text has been found, either click the **Replace** button to replace that instance of it or click the **Find Next** button to highlight the next instance (if any).

Each time you click the **Replace** button, Program Editor replaces that instance of the specified text and automatically highlights the next instance.

Checking the Syntax of a Script

When you try to run or debug a script whose syntax hasn't been checked, Program Editor first performs a syntax check automatically.

Here's how to perform a syntax check manually when you are editing your script, without having to run it.

1. From the **File** or the toolbar menu, choose the **Compile** command. The Program Editor either indicates that no errors have been found or displays an error message that specifies the first line in your script where an error has been found and briefly describes the nature of that error.
2. Click the **OK** button or press **Enter**. If Program Editor has found a syntax error, the line containing the error is highlighted on your display.
3. Correct the syntax error.
4. Repeat steps 1–3 until you have found and corrected all syntax errors.

Editing Dialog Box Templates

Here's how to invoke Dialog Editor and use it to create a new dialog box template for use in your script.

Inserting A New Dialog Box Template

To insert a new dialog box template into your program:

1. Place the insertion point where you want the new dialog box template to appear in your script.
2. From the **Edit** menu, choose the **Insert Dialog** command. The Program Editor's application window is temporarily disabled, and the *Dialog Editor* appears, displaying a new dialog box in its application window.
3. Use *Dialog Editor* to create your dialog box.
4. Exit from *Dialog Editor* and return to Program Editor.

Program Editor automatically places the new dialog box template generated by Dialog Editor in your script at the location of the insertion point.

Editing An Existing Dialog Box

Here's how to invoke Dialog Editor and use it to modify a dialog box template contained in your script.

1. Select the code for the entire dialog box template.
2. From the **Edit** menu, choose the **Edit Dialog** command. Program Editor's application window is temporarily disabled, and *Dialog Editor* appears, displaying in its application window a dialog box created from the code you selected.
3. Use Dialog Editor to modify your dialog box.
4. Exit from Dialog Editor and return to Program Editor.

Program Editor automatically replaces the dialog box template you originally selected with the revised template generated by Dialog Editor.

Refer to Chapter 4 for a detailed discussion of how to use Dialog Editor to create and edit dialog box templates.

Editing Custom Dialog Boxes

About Editing Dialog Boxes

You can use a custom dialog box to display information to a user while providing an opportunity for the user to respond. The Dialog Editor is a tool that enables you to create and modify custom dialog boxes for use in your scripts. Although the statements used to display a custom dialog box and respond to the choices made by a user of the dialog box may seem complicated, the Dialog Editor makes it easy to generate these statements.

This chapter contains the following topics:

- Overview
- Using the Dialog Editor
- Creating a Custom Dialog Box
- Editing a Custom Dialog Box
- Editing an Existing Dialog Box
- Testing an Edited Dialog Box
- Pasting a Dialog Box Template into Your Script
- Exiting from Dialog Editor
- Using a Custom Dialog Box in Your Script
- Using a Dynamic Dialog Box in Your Script
- Menu Reference

Overview of Dialog Editor

Sometimes your script will need to obtain information from the user. In many cases, you can obtain this information by using one of the Basic Control Engine's predefined dialog boxes in your script. When you must go beyond the information-gathering capabilities provided by predefined dialog boxes, you can use Dialog Editor to create a custom dialog box for use in your script.

Dialog Editor is a tool that allows you to generate a *dialog box template* in a script simply by editing an on-screen dialog box layout. You can then incorporate the template that Dialog Editor generates into your script.

The balance of this section provides general information that you'll need in order to work with Dialog Editor, including:

- Features that Dialog Editor supports
- An introduction to Dialog Editor's application window
- A list of keyboard shortcuts
- How to use the Help system

Then, in the following sections, you'll learn how to use Dialog Editor to create and edit custom dialog boxes and to edit dialog boxes captured from other applications. You'll also learn how to test an edited dialog box and incorporate the dialog box template generated by Dialog Editor into your script. And finally, you'll learn how to exit from Dialog Editor.

Features of the Dialog Editor

Dialog Editor supports the following features:

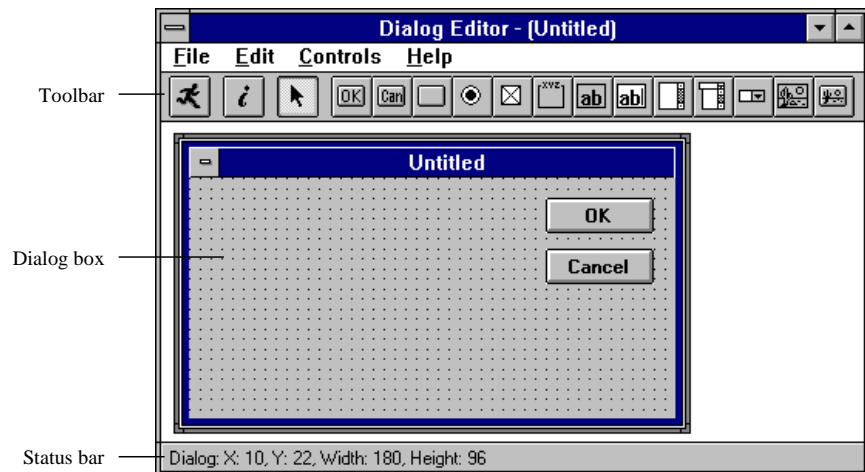
- Visual editing of a dialog box template in a script
- The creation of dynamic dialog boxes

Using the Dialog Editor

This section presents general information that will help you work most effectively with Dialog Editor. It includes an overview of Dialog Editor's application window—the interface you'll use to create and edit dialog box templates in a script—as well as a list of keyboard shortcuts and information on using the Help system.

Dialog Editor's Application Window

Before you begin creating a new custom dialog box, Dialog Editor's application window looks like this:



















The application window contains the following elements:

- **Toolbar:** a collection of tools that you can use to provide instructions to the Dialog Editor, as discussed in the following subsection
- **Dialog box:** the visual layout of the dialog box that you are currently creating or editing
- **Status bar:** provides key information about the operation you are currently performing, including the name of the currently selected control or dialog box, together with its position on the display and its dimensions; the name of a control you are about to add to the dialog box with the mouse pointer, together with the pointer's position on the display; the function of the currently selected menu command; and the activation of Dialog Editor's testing or capturing functions.

Note: Dialog boxes created with Dialog Editor normally appear in an 8 point Helvetica font, both in Dialog Editor's application window and when the corresponding code is run.

Dialog Editor's Toolbar

The following list briefly explains the purpose of each of the tools on Dialog Editor's toolbar, which you can use to add controls to your dialog box, make various changes to the dialog box and its controls, and test the dialog box's functioning.

<u>Icon</u>	<u>Tool</u>	<u>Function</u>
	Run	Runs the dialog box for testing purposes.
	Information	Displays the Information dialog box for the selected dialog box or control.
	Pick	Lets you select, move, and resize items and control the insertion point.
	OK Button	Adds an OK button to your dialog box.
	Cancel Button	Adds a Cancel button to your dialog box.
	Push Button	Adds a push button to your dialog box.
	Option Button	Adds an option button to your dialog box.
	Check Box	Adds a check box to your dialog box.
	Group Box	Adds a group box to your dialog box.
	Text	Adds a text control to your dialog box.
	Text Box	Adds a text box to your dialog box.
	List Box	Adds a list box to your dialog box.
	Combo Box	Adds a combo box to your dialog box.
	Drop List Box	Adds a drop list box to your dialog box.
	Picture	Adds a picture to your dialog box.
	Picture Button	Adds a picture button to your dialog box.

The types of dialog box controls that you can add with the control tools are fully described in the next section of the chapter.

Keyboard Shortcuts for Dialog Editor

The following keyboard shortcuts can be used for some of the operations you will perform most frequently in Dialog Editor.

<u>Key(s)</u>	<u>Function</u>
Alt+F4	Closes Dialog Editor's application window.
Ctrl+C	Copies the selected dialog box or control, without removing it from Dialog Editor's application window, and places it on the Clipboard.
Ctrl+D	Creates a duplicate copy of the selected control.
Ctrl+G	Displays the Grid dialog box.
Ctrl+I	Displays the Information dialog box for the selected dialog box or control.
Ctrl+V	Inserts the contents of the Clipboard into Dialog Editor. If the Clipboard contains script statements describing one or more controls, then Dialog Editor adds those controls to the current dialog box. If the Clipboard contains the script template for an entire dialog box, then Dialog Editor creates a new dialog box from the statements in the template.
Ctrl+X	Removes the selected dialog box or control from Dialog Editor's application window and places it on the Clipboard.
Ctrl+Z	Undoes the preceding operation.
Del	Removes the selected dialog box or control from Dialog Editor's application window without placing it on the Clipboard.
F1	Displays Help for the currently active window.
F2	Runs the dialog box for testing purposes.
F3	Sizes certain controls to fit the text they contain.
Shift+F1	Toggles the Help pointer.

Using the Help System

Dialog Editor provides several ways to obtain on-line help. You can display Help for the window or dialog box that is currently active, or you can search for a specific topic

Display Help for the currently active window

To display Help for the currently active window.:

- Press **F1**.

If Dialog Editor's application window was active, the Help system contents appear. If a dialog box was active, Help for that dialog box appears.

Searching for a Topic

To pinpoint a specific topic in the Help system.:

1. From the **Help** menu, choose the **Search for Help on** command. A scrollable list of Help topics appears.
2. Select the desired topic from the list. The topic you selected is displayed in a second scrollable list, together with closely related Help topics, if any.
3. If the desired topic is not already highlighted on the second list, select it and press **Enter**.

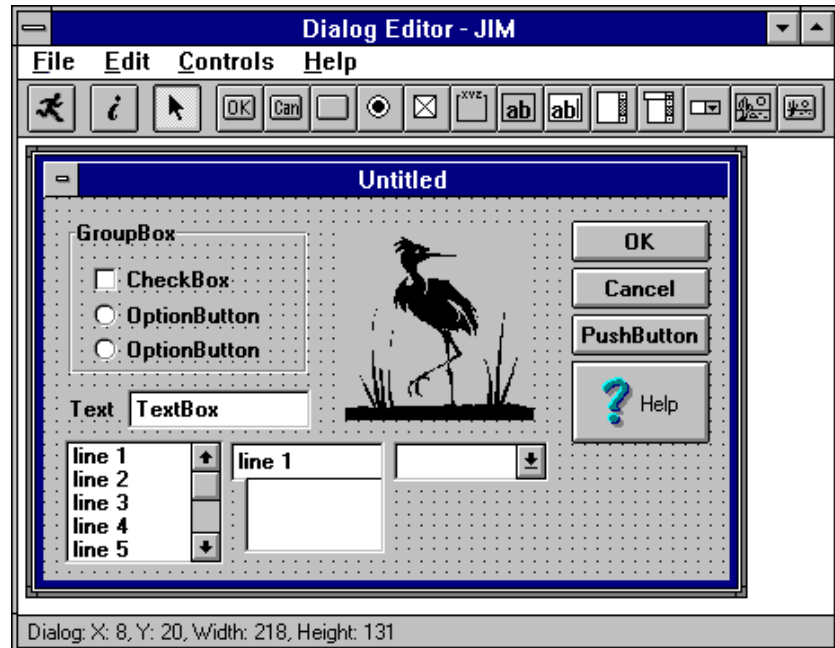
Help is displayed for the topic you selected.

Creating a Custom Dialog Box

This section describes the types of controls that Dialog Editor supports. It also explains how to create controls and initially position them within your dialog box, and offers some pointers on creating controls efficiently.

In the next section, "Editing a Custom Dialog Box," you'll learn how to make various types of changes to the controls that you've created—moving and resizing them, assigning labels and accelerator keys, and so forth.

Types of Controls



Dialog Editor supports the following types of standard Windows controls, all of which are illustrated in the above dialog box:

Push button

Push button is a command button. The default **OK** and **Cancel** buttons are special types of push buttons.

Option button

Option button is one of a group of two or more linked buttons that let users select only one from a group of mutually exclusive choices. A group of option buttons works the same way as the buttons on a car radio: because the buttons operate together as a group, clicking an unselected button in the group selects that button and automatically deselects the previously selected button in that group.

Check box

Check box is a box that users can check or clear to indicate their preference regarding the alternative specified on the check box label.

Group box

Group box is a rectangular design element used to enclose a group of related controls. You can use the optional group box label to display a title for the controls in the box.

Note

Group boxes, text controls, and pictures are passive elements in a dialog box, inasmuch as they are used purely for decorative or informative purposes. Users cannot act upon these controls, and when they tab through the dialog box, the focus skips over these controls.

Text

Text is a field containing text that you want to display for the users' information. The text in this field wraps, and the field can contain a maximum of 255 characters. Text controls can either display stand-alone text or be used as labels for text boxes, list boxes, combo boxes, drop list boxes, pictures, and picture buttons. You can choose the font in which the text appears.

Note

Group boxes, text controls, and pictures are passive elements in a dialog box, inasmuch as they are used purely for decorative or informative purposes. Users cannot act upon these controls, and when they tab through the dialog box, the focus skips over these controls.

Text box

Text box is a field into which users can enter text (potentially, as much as 32K). By default, this field holds a single line of nonwrapping text. If you choose the Multiline setting in the Text Box Information dialog box, this field will hold multiple lines of wrapping text.

Note

Group boxes, text controls, and pictures are passive elements in a dialog box, inasmuch as they are used purely for decorative or informative purposes. Users cannot act upon these controls, and when they tab through the dialog box, the focus skips over these controls.

List box

List box is a displayed, scrollable list from which users can select one item. The currently selected item is highlighted on the list.

Combo box

Combo box is a text field with a displayed, scrollable list beneath it. Users can either select an item from the list or enter the name of the desired item in the text field. The currently selected item is displayed in the text field. If the item was selected from the scrolling list, it is highlighted there as well.

Drop list box

Drop list box is a field that displays the currently selected item, followed by a downward-pointing arrow, which users can click to temporarily display a scrolling list of items. Once they select an item from the list, the list disappears and the newly selected item is displayed in the field.

Picture

Picture is a field used to display a Windows bitmap or metafile.

Note

Group boxes, text controls, and pictures are passive elements in a dialog box, inasmuch as they are used purely for decorative or informative purposes. Users cannot act upon these controls, and when they tab through the dialog box, the focus skips over these controls.

You can obtain a Windows bitmap or metafile from a file or from a specified library.

Picture button

Picture button is a special type of push, or command, button on which a Windows bitmap or metafile appears.

Note

You can obtain a Windows bitmap or metafile from a file or from a specified library.

Adding Controls to a Dialog Box

In this subsection, you'll learn how to create controls and determine approximately where they first appear within your dialog box. In the following subsection, you'll learn how to determine the positioning of controls more precisely.

Here's how to add one or more controls to your dialog box using simple mouse and keyboard methods.

Adding A Control

You can only insert a control within the borders of the dialog box you are creating. You cannot insert a control on the dialog box's title bar or outside its borders.

To add a control:

1. From the toolbar, choose the tool corresponding to the type of control you want to add.
2. Place the pointer where you want the control to be positioned and click the mouse button.

Note

When you pass the mouse pointer over an area of the display where a control can be placed, the pointer becomes an image of the selected control with crosshairs (for positioning purposes) to its upper left. The name and position of the selected control appear on the status bar. When you pass the pointer over an area of the display where a control cannot be placed, the pointer changes into a circle with a slash through it (the "prohibited" symbol).

The control you just created appears at the specified location. (To be more specific, the upper left corner of the control will correspond to the position of the pointer's crosshairs at the moment you clicked the mouse button.) The control is surrounded by a thick frame, which means that it is selected, and it may also have a default label.

After the new control has appeared, the mouse pointer becomes an arrow, to indicate that the **Pick** tool is active and you can once again select any of the controls in your dialog box.

To add another control of the same type as the one you just added, press **Ctrl+D**. A duplicate copy of the control appears.

To add a different type of control, repeat steps 1 and 2.

To reactivate the **Pick** tool, you can do one of the following:

- Click the arrow-shaped tool on the toolbar.
- Place the mouse pointer on the title bar of the dialog box or outside the borders of the dialog box (that is, on any area where the mouse pointer turns into the "prohibited" symbol) and click the mouse button.

As you plan your dialog box, keep in mind that a single dialog box can contain no more than 255 controls and that a dialog box will not operate properly unless it contains either an **OK** button, a **Cancel** button, a push button, or a picture button. (When you create a new custom dialog box, an **OK** button and a **Cancel** button are provided for you by default.)

Later in the chapter, you'll learn more about selecting controls, and you'll learn how to assign labels.

Using the Grid to Help You Position Controls within a Dialog Box

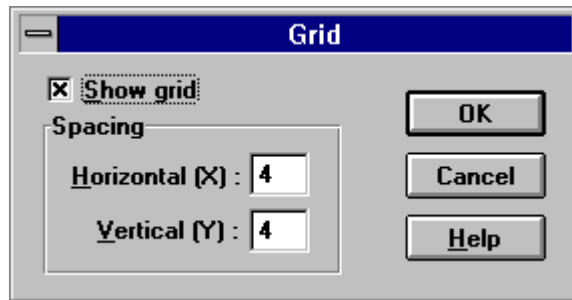
The preceding subsection explained how to determine approximately where a newly created control will materialize in your dialog box. Here, you'll learn how to use Dialog Editor's grid to help you fine-tune the initial placement of controls.

The area of your dialog box in which controls can be placed (that is, the portion of the dialog box below the title bar) can be thought of as a grid, with the X (horizontal) axis and the Y (vertical) axis intersecting in the upper left corner (the 0, 0 coordinates). The position of controls can be expressed in terms of X units with respect to the left border of this area and in terms of Y units with respect to the top border. (In fact, the position of controls is expressed in this manner within the dialog box template that you produce by working with Dialog Editor.)

Here's how to display the grid and adjust its X and Y settings, which can help you position controls more precisely within your dialog box.

To display and adjust the grid:

1. Press **Ctrl+G**. Dialog Editor displays the *Grid* dialog box:



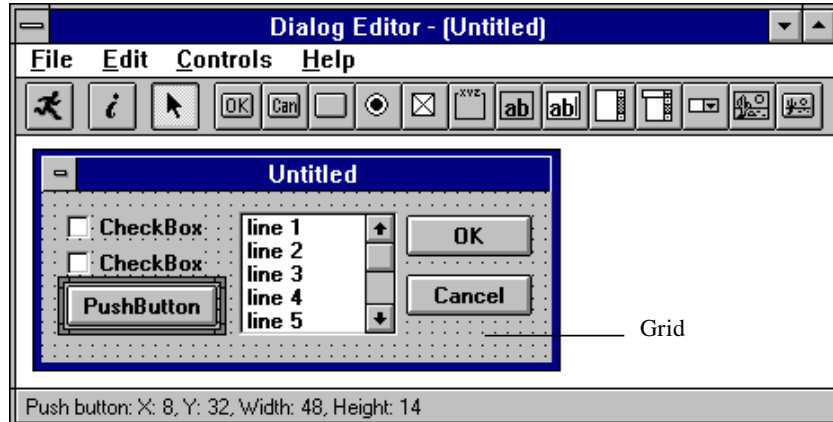
2. To display the grid in your dialog box, select the **Show grid** check box.
3. To change the current X and Y settings, enter new values in the **X** and **Y** fields.

Note

The values of X and Y in the *Grid* dialog box determine the grid's spacing. Assigning smaller X and Y values produces a more closely spaced grid, which enables you to move the mouse pointer in smaller horizontal and vertical increments as you position controls. Assigning larger X and Y values produces the opposite effect on both the grid's spacing and the movement of the mouse pointer. The X and Y settings entered in the Grid dialog box remain in effect regardless of whether you choose to display the grid.

4. Click the **OK** button or press **Enter**.

Dialog Editor displays the grid with the settings you specified. With the grid displayed, you can line up the crosshairs on the mouse pointer with the dots on the grid to position controls precisely and align them with respect to other controls.



As you move the mouse pointer over the dialog box after you have chosen a control tool from the toolbar, the status bar displays the name of the type of control you have selected and continually updates the position of the mouse pointer in X and Y units. (This information disappears if you move the mouse pointer over an area of the screen where a control cannot be placed.) After you click the mouse button to add a control, that control remains selected, and the status bar displays the control's width and height in dialog units as well as its name and position, as shown in the preceding illustration, in which the push button is selected.

Note

Dialog units represent increments of the font in which Dialog Editor creates dialog boxes (namely, 8 point Helvetica). Each X unit represents an increment equal to 1/4 of that font, and each Y unit represents an increment equal to 1/8 of that font.

Creating Controls Efficiently

Creating dialog box controls in random order might seem like the fastest approach. However, the order in which you create controls has some important implications, so a little advance planning can save you a lot of work in the long run.

Here are several points about creating controls that you should keep in mind:

- Tabbing order:** Users can select dialog box controls by tabbing, as explained in the next subsection. The order in which you create the controls is what determines the tabbing order. That is, as users tab through the dialog box, the focus is changed from one control to the next in the order in which you created the controls (regardless of the order in which you position the controls in the dialog box). The closer you can come to creating controls in the order in which you want them to receive the tabbing focus, the fewer tabbing-order adjustments you'll have to make later on.

- **Option button grouping:** If you want a series of option buttons to work together as a mutually exclusive group, you must create all the buttons in that group one right after the other, in an unbroken sequence. If you get sidetracked and create a different type of control before you have finished creating all the option buttons in your group, you'll split the buttons into two (or more) separate groups. (Let's say you want to create an option button group with five buttons. You create three of the buttons and then create a list box, after which you finish creating the last two buttons. When you test your dialog box, you'll find that all five of these option buttons *don't* work together as a mutually exclusive group. Instead, the first three buttons will form one mutually exclusive group, and the last two buttons will form another mutually exclusive group.)
- **Accelerator keys:** As explained later in the chapter, you can provide easy access to a text box, list box, combo box, or drop list box by assigning an accelerator key to an associated text control, and you can provide easy access to the controls in a group box by assigning an accelerator key to the group box label. To do this, you must create the text control or group box first, followed immediately by the controls that you want to associate with it. If the controls are not created in the correct order, they will not be associated in your dialog box template, and any accelerator key you assign to the text control or group box label will not work properly.

If you don't create controls in the most efficient order, the resulting problems with tabbing order, option button grouping, and accelerator keys usually won't become apparent until you test your dialog box. Although you can still fix these problems at that point, as explained later in the chapter, it will definitely be more cumbersome. In short, it's easier to prevent (or at least minimize) problems of this sort than to fix them after the fact.

Editing a Custom Dialog Box

In the preceding section, you learned how to create controls and determine where they initially appear within your dialog box. In this section, you'll learn how to make various types of changes to both the dialog box and the controls in it. The following topics are included:

- Selecting items so that you can work with them
- Using the Information dialog box to check and/or change various attributes of items
- Changing the position and size of items
- Changing titles and labels
- Assigning accelerator keys
- Specifying pictures
- Creating or modifying picture libraries under Windows
- Duplicating and deleting controls
- Undoing editing operations

Selecting Items

In order to edit a dialog box or a control, you must first select it. When you select an item, it becomes surrounded by a thick frame, as you saw in the preceding section. You may select a control or dialog box using either mouse or keyboard methods.

To select a control:

With the **Pick** tool active, place the mouse pointer on the desired control and click the mouse button.

-Or-

With the **Pick** tool active, press the **Tab** key repeatedly until the focus moves to the desired control.

The control is now surrounded by a thick frame to indicate that it is selected and you can edit it.

To select the dialog box:

With the **Pick** tool active, place the mouse pointer on the title bar of the dialog box or on an empty area within the borders of the dialog box (that is, on an area where there are no controls) and click the mouse button.

-Or-

With the **Pick** tool active, press the **Tab** key repeatedly until the focus moves to the dialog box.

The dialog box is now surrounded by a thick frame to indicate that it is selected and you can edit it.

Using the Information Dialog Box

The *Information* dialog box enables you to check and adjust various attributes of controls and dialog boxes. This subsection explains how to display the Information dialog box and provides an overview of the attributes with which it lets you work. In the following subsections, you'll learn more about how to use the Information dialog box to make changes to your dialog box and its controls.

- You can use the *Dialog Box Information* dialog box to check and adjust attributes that pertain to the dialog box as a whole.
- You can use the *Information* dialog box for a control to check and adjust attributes that pertain to that particular control.

To display the Information dialog box for a dialog box:

With the **Pick** tool active, place the mouse pointer on an area of the dialog box where there are no controls and double-click the mouse button.

-Or-

With the **Pick** tool active, select the dialog box and either click the Information tool on the toolbar, press **Enter**, or press **Ctrl+I**.

Dialog Editor displays the *Dialog Box Information* dialog box:

The screenshot shows the "Dialog Box Information" dialog box. It has a title bar with a minus sign on the left. The dialog is divided into several sections:

- Position:** Two input fields for X and Y coordinates.
- Size:** Two input fields for Width (164) and Height (76).
- Text\$:** A text field containing "Untitled".
- Name:** A text field containing "UserDialog".
- .Function:** An empty text field.
- Picture Library:** An empty text field with a "Browse..." button below it.

On the right side of the dialog, there are three buttons: "OK", "Cancel", and "Help". Below the "Text\$" and "Picture Library" fields, there are two checkboxes labeled "Variable Name", both of which are unchecked.

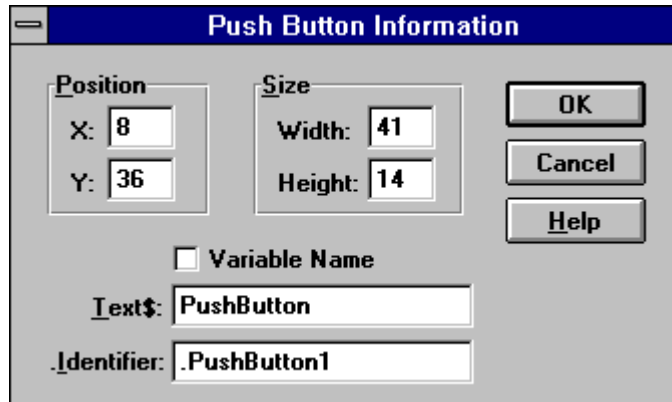
To display the Information dialog box for a control:

With the **Pick** tool active, place the mouse pointer on the desired control and double-click the mouse button.

-Or-

With the **Pick** tool active, select the control and either click the Information tool on the toolbar, press **Enter**, or press **Ctrl+I**.

Dialog Editor displays an *Information* dialog box corresponding to the control you selected. Here is an example:



Attributes That You Can Adjust with the Dialog Box Information Dialog Box

The following lists show the attributes that you can change with the Dialog Box Information dialog box for the various controls. In some cases (specified below), it's mandatory to fill in the fields in which the attributes are specified—that is, you must either leave the default information in these fields or replace it with more meaningful information, but you can't leave the fields empty. In other cases, filling in these fields is optional—that is, you can either leave the default information in the fields, replace it with more meaningful information, or leave the fields entirely empty.

Note

A quick way to determine whether it's mandatory to fill in a particular Information dialog box field is to see whether the **OK** button becomes grayed out when you delete the information in that field. If it does, then you *must* fill in that field.

In many cases, you could simply leave the generic-sounding default information in the Information dialog box fields and worry about replacing it with more meaningful information after you paste the dialog box template into your script. However, if you take a few moments to replace the default information with something specific when you first create your dialog box, not only will you save yourself some work later on but you may also find that your changes make the script code produced by Dialog Editor more readily comprehensible and hence easier to work with.

The Dialog Box Information dialog box can be used to check and adjust the following attributes, which pertain to the dialog box as a whole.

Mandatory/

Optional

Attribute

Optional	Position: X and Y coordinates on the display, in dialog units
Mandatory	Size: width and height of the dialog box, in dialog units
Optional	Style: options that allow you to determine whether the close box and title bar are displayed
Optional	Text\$: text displayed on the title bar of the dialog box
Mandatory	Name: name by which you refer to this dialog box template in your script code
Optional	.Function: name of a script function in your dialog box
Optional	Picture Library: picture library from which one or more pictures in the dialog box are obtained

Attributes That You Can Adjust with the Information Dialog Box for a Control

The Information dialog box for a control can be used to check and adjust the following attributes. The second column of the list indicates the control(s) to which each attribute pertains.

Mandatory/ Optional	Control(s) Affected	Attribute
Mandatory	All controls	Position: X and Y coordinates within the dialog box, in dialog units
Mandatory	All controls	Size: width and height of the control, in dialog units
Optional	Push button, option button, check box, group box, and text	Text\$: text displayed on a control
Optional	Text	Font: font in which text is displayed
Optional	Text box	Multiline: option that allows you to determine whether users can enter a single line of text or multiple lines
Optional	OK button, Cancel button, push button, option button, group box, text, picture, and picture button	.Identifier: name by which you refer to a control in your script code
Mandatory	Check box, text box, list box, combo box, and drop list box	.Identifier: name by which you refer to a control in your script code; also contains the result of the control after the dialog box has been processed
Optional	Picture, picture button	.Identifier: name of the file containing a picture that you want to display or the name of a picture that you want to display from a specified picture library
Optional	Picture	Frame: option that allows you to display a 3-D frame
Mandatory	List box, combo box, and drop list box	Array\$: name of an array variable in your script code
Mandatory	Option button	.Option Group: name by which you refer to a group of option buttons in your script code

Changing The Position of an Item

Dialog Editor's display can be thought of as a grid, in which the X (horizontal) axis and the Y (vertical) axis intersect in the upper left corner of the display (the 0, 0 coordinates). The position of the dialog box you are creating can be expressed in terms of X units with respect to the left border of the parent window and in terms of Y units with respect to the top border.

As explained in the preceding section, the portion of your dialog box below the title bar can also be thought of as a grid, with the X and Y axes intersecting in the upper left corner of this area. The position of controls within the dialog box can be expressed in terms of X units with respect to the left border of this area and in terms of Y units with respect to the top border.

When you select a dialog box or control, the status bar displays its position in X and Y units as well as its width and height in dialog units. Each time you move or resize an item, the corresponding information on the status bar is updated. You can use this information to position and size items more precisely.

Dialog Editor provides several ways to reposition dialog boxes and controls.

To reposition an item with the mouse:

Here's how to move a dialog box or control by dragging it with the mouse.

1. With the **Pick** tool active, place the mouse pointer on an empty area of the dialog box or on a control.
2. Depress the mouse button and drag the dialog box or control to the desired location.

Note

The increments by which you can move a control with the mouse are governed by the grid setting. For example, if the grid's X setting is 4 and its Y setting is 6, you'll be able to move the control horizontally only in increments of 4 X units and vertically only in increments of 6 Y units. This feature is handy if you're trying to align controls in your dialog box. If you want to move controls in smaller or larger increments, press **Ctrl+G** to display the Grid dialog box and adjust the X and Y settings.

To reposition an item with the arrow keys:

Here's how to move a selected dialog box or control by pressing the arrow keys.

1. Select the dialog box or control that you want to move.
2. Press an arrow key once to move the item by 1 X or Y unit in the desired direction, *or* depress an arrow key to "nudge" the item steadily along in the desired direction.

Note

When you reposition an item with the arrow keys, a faint, partial afterimage of the item may remain visible in the item's original position. These afterimages are rare and will disappear once you test your dialog box.

To reposition a dialog box with the Dialog Box Information dialog box:

Here's how to move a selected dialog box by changing its coordinates in the Dialog Box Information dialog box.

1. Display the *Dialog Box Information* dialog box.
2. Change the **X** and **Y** coordinates in the **Position** group box, *or*, leave the X and/or Y coordinates blank.
3. Click the **OK** button or press **Enter**.

If you specified X and Y coordinates, the dialog box moves to that position. If you left the X coordinate blank, the dialog box will be centered horizontally relative to the parent window of the dialog box when the dialog box is run. If you left the Y coordinate blank, the dialog box will be centered vertically relative to the parent window of the dialog box when the dialog box is run.

To reposition a control with the Information dialog box:

Here's how to move a selected control by changing its coordinates in the Information dialog box for that control.

1. Display the *Information* dialog box for the control that you want to move.
2. Change the **X** and **Y** coordinates in the **Position** group box.
3. Click the **OK** button or press **Enter**.

The control moves to the specified position.

Note

When you move a dialog box or control with the arrow keys or with the Information dialog box, the item's movement is *not* restricted to the increments specified in the grid setting.

When you attempt to test a dialog box containing hidden controls (i.e., controls positioned entirely outside the current borders of your dialog box), Dialog Editor displays a message advising you that there are controls outside the dialog box's borders and asks whether you wish to proceed with the test. If you proceed, the hidden controls will be disabled for testing purposes. (Testing dialog boxes is discussed later in the chapter.)

Changing the Size of an Item

Dialog boxes and controls can be resized either by directly manipulating them with the mouse or by using the Information dialog box. Certain controls can also be resized automatically to fit the text displayed on them.

To resize an item with the mouse:

Here's how to change the size of a selected dialog box or control by dragging its borders or corners with the mouse.

1. With the **Pick** tool active, select the dialog box or control that you want to resize.
2. Place the mouse pointer over a border or corner of the item.
3. Depress the mouse button and drag the border or corner until the item reaches the desired size.

To resize an item with the Information dialog box:

Here's how to change the size of a selected dialog box or control by changing its Width and/or Height settings in the Information dialog box.

1. Display the *Information* dialog box for the dialog box or control that you want to resize.
2. Change the **Width** and **Height** settings in the **Size** group box.
3. Click the **OK** button or press **Enter**.

The dialog box or control is resized to the dimensions you specified.

To resize selected controls automatically:

Here's how to adjust the borders of certain controls automatically to fit the text displayed on them.

1. With the **Pick** tool active, select the option button, text control, push button, check box, or text box that you want to resize.
2. Press **F2**.

The borders of the control will expand or contract to fit the text displayed on it.

Note

Windows metafiles always expand or contract proportionally to fit within the picture control or picture button control containing them. In contrast, windows bitmaps are of a fixed size. If you place a bitmap in a control that is smaller than the bitmap, the bitmap is clipped off on the right and bottom. If you place a bitmap in a control that is larger than the bitmap, the bitmap is centered within the borders of the control. Picture controls and picture button controls must be resized manually.

Changing Titles and Labels

By default, when you begin creating a dialog box, its title reads "Untitled," and when you first create group boxes, option buttons, push buttons, text controls, and check boxes, they have generic-sounding default labels, such as "Group Box" and "Option Button."

To change a dialog box title or a control label:

Here's how to change the title of your dialog box as well as the labels of group boxes, option buttons, push buttons, text controls, and check boxes.

1. Display the *Information* dialog box for the dialog box whose title you want to change or for the control whose label you want to change.
2. Enter the new title or label in the **Text\$** field.

Note

Dialog box titles and control labels are optional. Therefore, you can leave the **Text\$** field blank.

3. If the information in the **Text\$** field should be interpreted as a variable name rather than a literal string, select the **Variable Name** check box.
4. Click the **OK** button or press **Enter**.

The new title or label is now displayed on the title bar or on the control.

Although **OK** and **Cancel** buttons also have labels, you cannot change them. The remaining controls (text boxes, list boxes, combo boxes, drop list boxes, pictures, and picture buttons) don't have their own labels, but you can position a text control above or beside these controls to serve as a de facto label for them.

Assigning Accelerator Keys

Accelerator keys enable users to access dialog box controls simply by pressing Alt + a specified letter. Users can employ accelerator keys to choose a push button or an option button; toggle a check box on or off; and move the insertion point into a text box or group box or to the currently selected item in a list box, combo box, or drop list box.

An accelerator key is essentially a single letter that you designate for this purpose from a control's label. You can assign an accelerator key directly to controls that have their own label (option buttons, push buttons, check boxes, and group boxes). (You can't assign an accelerator key to **OK** and **Cancel** buttons because, as noted above, their labels can't be edited.) You can create a de facto accelerator key for certain controls that don't have their own labels (text boxes, list boxes, combo boxes, and drop list boxes) by assigning an accelerator key to an associated text control.

To assign an accelerator key:

Here's how to designate a letter from a control's label to serve as the accelerator key for that control.

1. Display the *Information* dialog box for the control to which you want to assign an accelerator key.
2. In the **Text\$** field, type an ampersand (&) before the letter you want to designate as the accelerator key.
3. Click the **OK** button or press **Enter**.

The letter you designated is now underlined on the control's label, and users will be able to access the control by pressing Alt + the underlined letter.

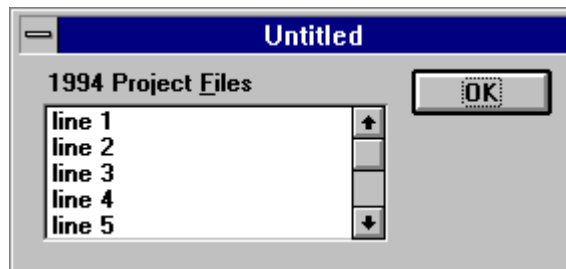
Note

Accelerator key assignments must be unique within a particular dialog box. If you attempt to assign the same accelerator key to more than one control, Dialog Editor displays a reminder that that letter has already been assigned.

If, for example, you have a push button whose label reads *Apply*, you can designate *A* as the accelerator key by displaying the Push Button Information dialog box and typing *&Apply* in the Text\$ field. When you press **Enter**, the button label looks like the following illustration, and users will be able to choose the button by typing Alt+A.



As another example, let's say you have a list box that is immediately preceded in the dialog box template by a text control whose label reads *1994 Project Files*. By using the method described above, you can designate *F* as the accelerator key. When you click **OK** or press **Enter**, the text control label looks like the following illustration, and users will be able to move the insertion point to the currently selected item in the list box by typing **Alt+F**.



Note

In order for such a de facto accelerator key to work properly, the text control or group box label to which you assign the accelerator key *must* be associated with the control(s) to which you want to provide user access—that is, in the dialog box template, the description of the text control or group box must immediately precede the description of the control(s) that you want associated with it. The simplest way to establish such an association is to create the text control or group box first, followed immediately by the associated control(s).

Specifying Pictures

In the preceding section, you learned how to add picture controls and picture button controls to your dialog box. But these controls are nothing more than empty outlines until you specify the pictures that you want them to display.

A picture control or picture button control can display a Windows bitmap or metafile, which you can obtain from a file or from a specified library. (Refer to the following subsection for information on creating or modifying picture libraries under Windows.)

To specify a picture from a file:

Here's how to display a Windows bitmap or metafile from a file on a picture control or picture button control by using the control's Information dialog box to indicate the file in which the picture is contained.

1. Display the *Information* dialog box for the picture control or picture button control whose picture you want to specify.
2. In the Picture source option button group, select File.
3. In the **Name\$** field, enter the name of the file containing the picture you want to display in the picture control or picture button control.

Note

By clicking the Browse button, you can display the Select a Picture File dialog box and use it to find the file.

4. Click the **OK** button or press **Enter**.

The picture control or picture button control now displays the picture you specified.

To specify a picture from a picture library:

Here's how to display a Windows bitmap or metafile from a library on a picture control or picture button control by first using the Dialog Box Information dialog box to specify the library and then using the control's Information dialog box to indicate the name of the picture.

1. Display the Dialog Box Information dialog box.
2. In the Picture Library field, specify the name of the picture library that contains the picture(s) you want to display in your dialog box.

Note

By clicking the Browse button, you can display the Select a Picture Library dialog box and use it to find the library.

If you specify a picture library in the Dialog Box Information dialog box, all the pictures in your dialog box must come from this library.

3. Click the **OK** button or press **Enter**.
4. Display the Information dialog box for the picture control or picture button control whose picture you want to specify.

5. In the Picture source option button group, select Library.
6. In the Name\$ field, enter the name of the picture you want to display on the picture control or picture button control. (This picture must be from the library that you specified in step 2.)
7. Click the **OK** button or press **Enter**.

The picture control or picture button control now displays the picture you specified.

Creating or Modifying Picture Libraries under Windows

The **Picture** statement allows images to be specified as individual picture files or as members of a picture library, which is a DLL that contains a collection of pictures. Currently, both Windows bitmaps and metafiles are supported. You can obtain a picture library either by creating a new one or by modifying an existing one, as described below.

Each image is placed into the DLL as a resource identified by its unique resource identifier. This identifier is the name used in the **Picture** statement to specify the image.

The following resource types are supported in picture libraries:

Resource Type	Description
2	Bitmap. This is defined in windows.h as RT_BITMAP .
256	Metafile. Since there is no resource type for metafiles, 256 is used.

To create a picture library under Windows:

Here's how to create a new picture library to contain the Windows bitmaps or metafiles that you want to display on picture controls or picture button controls in your dialog box.

1. Create a C file containing the minimal code required to establish a DLL. The following code can be used:

```
#include <windows.h>

int CALLBACK LibMain(
    HINSTANCE hInstance,
    WORD wDataSeg,
    WORD wHeapSz,
    LPSTR lpCmdLine) {
    UnlockData(0);
    return 1;
}
```

2. Use the following code to create a DEF file for your picture library:

```
LIBRARY
DESCRIPTION "My Picture Library"
EXETYPE WINDOWS
CODE LOADONCALL MOVABLE DISCARDABLE
DATA PRELOAD MOVABLE SINGLE
HEAPSIZE 1024
```

3. Create a resource file containing your images. The following example shows a resource file using a bitmap called **sample.bmp** and a metafile called **usa.wmf**.

```
#define METAFILE 256  
  
USA METAFILE "usa.wmf"  
MySample BITMAP "sample.bmp"
```

4. Create a make file that compiles your C module, creates the resource file, and links everything together.

To modify an existing picture library:

Here's how to modify an existing picture library to contain the Windows bitmaps or metafiles that you want to display on picture controls or picture button controls in your dialog box.

1. Make a copy of the picture library you want to modify.
2. Modify the copy by adding images using a resource editor such as Borland's Resource Workshop or Microsoft's App Studio.

Note: When you use a resource editor, you need to create a new resource type for metafiles (with the value 256).

Duplicating and Deleting Controls

You can use Dialog Editor's duplicating feature if you need one or more copies of a particular control. You can also use the delete features to delete a single control or clear an entire dialog box.

To duplicate a control:

Here's how to use Dialog Editor's duplicating feature, which saves you the work of creating additional controls individually if you need one or more copies of a particular control.

1. Select the control that you want to duplicate.
2. Press **Ctrl+D**. A duplicate copy of the selected control appears in your dialog box.
3. Repeat step 2 as many times as necessary to create the desired number of duplicate controls.

Duplicating is a particularly efficient approach if you need to create a group of controls, such as a series of option buttons or check boxes. Simply create the first control in the group and then, while the newly created control remains selected, repeatedly press **Ctrl+D** until you have created the necessary number of copies.

To delete a single control:

If you want to remove controls from your dialog box selectively, here's how to delete them one at a time.

1. Select the control you want to delete.
2. Press **Delete**.

The selected control is removed from your dialog box.

To delete all the controls in a dialog box:

If you want to "wipe the slate clean" and start all over again with your dialog box, here's how to remove all its controls in a single operation.

1. Select the dialog box.
2. Press **Delete**.
3. If the dialog box contains more than one control, Dialog Editor prompts you to confirm that you want to delete all controls. Click the Yes button or press **Enter**.

All the controls disappear, but the dialog box's title bar and close box (if displayed) remain unchanged.

Undoing Editing Operations

You can undo editing operations that produce a change in your dialog box, including:

- The addition of a control
- The insertion of one or more controls from the Clipboard
- The deletion of a control
- Changes made to a control or dialog box, either with the mouse or with the Information dialog box

You cannot undo operations that don't produce any change in your dialog box, such as selecting controls or dialog boxes and copying material to the Clipboard.

To undo an editing operation:

Here's how to reverse the effect of the preceding editing operation.

- Press **Ctrl+Z**.

Your dialog box is restored to the way it was before you performed the editing operation.

Editing an Existing Dialog Box

There are three ways to edit an existing dialog box: (1) You can copy the template of the dialog box you want to edit from a script to the Clipboard and paste it into Dialog Editor. (2) You can use the capture feature to "grab" an existing dialog box from another application and insert a copy of it into Dialog Editor. (3) You can open a dialog box template file that has been saved on a disk. Once you have the dialog box displayed in Dialog Editor's application window, you can edit it using the methods described earlier in the chapter.

Pasting an Existing Dialog Box into Dialog Editor

You can use Dialog Editor to modify the statements in your script that correspond to an entire dialog box or to one or more dialog box controls.

To paste an existing dialog box into Dialog Editor:

If you want to modify a dialog box template contained in your script, here's how to select the template and paste it into Dialog Editor for editing.

1. Copy the entire dialog box template (from the **Begin Dialog** instruction to the **End Dialog** instruction) from your script to the Clipboard.
2. Open Dialog Editor.
3. Press **Ctrl+V**.
4. When Dialog Editor asks whether you want to replace the existing dialog box, click the Yes button.

Dialog Editor creates a new dialog box corresponding to the template contained on the Clipboard.

To paste one or more controls from an existing dialog box into Dialog Editor:

If you want to modify the statements in your script that correspond to one or more dialog box controls, here's how to select the statements and paste them into Dialog Editor for editing.

1. Copy the description of the control(s) from your script to the Clipboard.
2. Open Dialog Editor.
3. Press **Ctrl+V**.

Dialog Editor adds to your current dialog box one or more controls corresponding to the description contained on the Clipboard.

Notes

When you paste a dialog box template into Dialog Editor, the tabbing order of the controls is determined by the order in which the controls are described in the template. When you paste one or more controls into Dialog Editor, they will come last in the tabbing order, following the controls that are already present in the current dialog box.

If there are any errors in the statements that describe the dialog box or controls, the Dialog Translation Errors dialog box will appear when you attempt to paste these statements into Dialog Editor. This dialog box shows the lines of code containing the errors and provides a brief description of the nature of each error.

Capturing a Dialog Box from Another Application

Here's how to capture the standard Windows controls from any standard Windows dialog box in another application and insert those controls into Dialog Editor for editing.

1. Display the dialog box you want to capture.
2. Open Dialog Editor.
3. Choose the **Capture Dialog** command from the **File** menu. Dialog Editor's application window moves behind all other open application windows, and the dialog box you displayed in step 1 reappears. The mouse pointer, previously an arrow, now looks like a butterfly net.
4. Place the mouse pointer over the dialog box that you want to capture. If the mouse pointer is over a standard Windows dialog box that contains some standard Windows controls, a tiny dialog box appears in front of the mouse pointer's butterfly net to indicate that the pointer has found controls that can be captured. If the mouse pointer is not over a standard Windows dialog box that contains standard Windows controls, the butterfly net remains unchanged to indicate that the mouse pointer has not found controls that can be captured.



Mouse pointer positioned over an area of the screen that does not contain standard Windows controls



Mouse pointer positioned over a standard Windows dialog box that contains some standard Windows controls

5. Click the mouse button.

Dialog Editor's application window moves in front of all other open application windows and now displays the standard Windows controls from the target dialog box.

Note

Dialog Editor only supports standard Windows controls and standard Windows dialog boxes. Therefore, if the target dialog box contains both standard Windows controls and custom controls, only the standard Windows controls will appear in Dialog Editor's application window. If the target dialog box is not a standard Windows dialog box, you will be unable to capture the dialog box or any of its controls.

Opening a Dialog Box Template File

Here's how to open any dialog box template file that has been saved on a disk so you can edit the template in Dialog Editor.

1. Choose **Open** from the **File** menu. The *Open Dialog File* dialog box appears.
2. Select the file containing the dialog box template that you want to edit and click the **OK** button.

Dialog Editor creates a dialog box from the statements in the template and displays it in the application window.

Note

If there are any errors in the statements that describe the dialog box, the Dialog Translation Errors dialog box will appear when you attempt to load the file into Dialog Editor. This dialog box shows the lines of code containing the errors and provides a brief description of the nature of each error.

Testing an Edited Dialog Box

Dialog Editor lets you run your edited dialog box for testing purposes. When you click the Test tool, your dialog box comes alive, which gives you an opportunity to make sure it functions properly and fix any problems before you incorporate the dialog box template into your script.

Before you run your dialog box, take a moment to look it over for basic problems such as the following:

- Does the dialog box contain a command button—that is, a default **OK** or **Cancel** button, a push button, or a picture button?
- Does the dialog box contain all the necessary push buttons?
- Does the dialog box contain a Help button if one is needed?
- Are the controls aligned and sized properly?
- If there is a text control, is its font set properly?
- Are the close box and title bar displayed (or hidden) as you intended?
- Are the control labels and dialog box title spelled and capitalized correctly?
- Do all the controls fit within the borders of the dialog box?
- Could you improve the design of the dialog box by adding one or more group boxes to set off groups of related controls?
- Could you clarify the purpose of any unlabeled control (such as a text box, list box, combo box, drop list box, picture, or picture button) by adding a text control to serve as a de facto label for it?
- Have you made all the necessary accelerator key assignments?

After you've fixed any elementary problems, you're ready to run your dialog box so you can check for problems that don't become apparent until a dialog box is activated.

Testing your dialog box is an iterative process that involves running the dialog box to see how well it works, identifying problems, stopping the test and fixing those problems, then running the dialog box again to make sure the problems are fixed and to identify any additional problems, and so forth—until the dialog box functions the way you intend.

To test your dialog box:

Here's how to test your dialog box and fine-tune its performance.

1. Click the **Run** tool on the toolbar, or press **F5**. The dialog box becomes operational, and you can check how it functions.
2. To stop the test, click the **Run** tool, press **F5**, or double-click the dialog box's close box (if it has one).
3. Make any necessary adjustments to the dialog box.
4. Repeat steps 1–3 as many times as you need in order to get the dialog box working properly.

When testing a dialog box, you can check for operational problems such as the following:

- **Tabbing order:** When you press the **Tab** key, does the focus move through the controls in a logical order? (Remember, the focus skips over items that users cannot act upon, including group boxes, text controls, and pictures.)

When you paste controls into your dialog box, Dialog Editor places their descriptions at the end of your dialog box template, in the order in which you paste them in. Therefore, you can use a simple cut-and-paste technique to adjust the tabbing order. First, click the Run tool to end the test and then, proceeding in the order in which you want the controls to receive the focus, select each control, cut it from the dialog box (by pressing **Ctrl+X**), and immediately paste it back in again (by pressing **Ctrl+V**). The controls will now appear in the desired order in your template and will receive the tabbing focus in that order.

- **Option button grouping:** Are the option buttons grouped correctly? Does selecting an unselected button in a group automatically deselect the previously selected button in that group?

To merge two groups of option buttons into a single group, click the Run tool to end the test and then use the Option Button Information dialog box to assign the same .Option Group name for all the buttons that you want included in that group.

- **Text box functioning:** Can you enter only a single line of nonwrapping text, or can you enter multiple lines of wrapping text?

If the text box doesn't behave the way you intended, click the Run tool to end the test; then display the Text Box Information dialog box and select or clear the Multiline check box.

- **Accelerator keys:** If you have assigned an accelerator key to a text control or group box in order to provide user access to a text box, list box, combo box, drop list box, or group box, do the accelerator keys work properly? That is, if you press **Alt +** the designated accelerator key, does the insertion point move into the text box or group box or to the currently selected item in the list box, combo box, or drop list box?

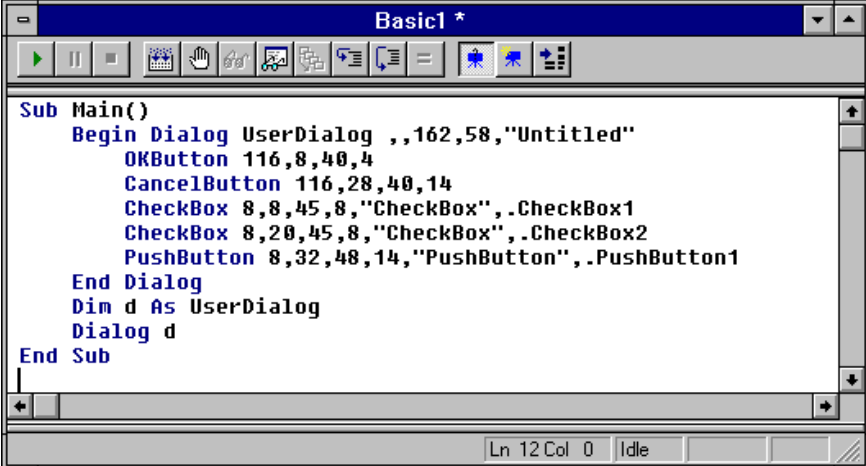
If the accelerator key doesn't work properly, it means that the text box, list box, combo box, drop list box, or group box is not associated with the text control or group box to which you assigned the accelerator key—that is, in your dialog box template, the description of the text control or group box does not immediately precede the description of the control(s) that should be associated with it. As with tabbing-order problems (discussed above), you can fix this problem by using a simple cut-and-paste technique to adjust the order of the control descriptions in your template. First, click the Run tool to end the test; then cut the text control or group box from the dialog box and immediately paste it back in again; and finally, do the same with each of the controls that should be associated with the text control or group box. The controls will now appear in the desired order in your template, and the accelerator keys will work properly.

Pasting a Dialog Box Template into Your Script

Dialog boxes and dialog box controls are communicated between Dialog Editor and your script via the Clipboard, where they are represented as statements. Here's how to copy a dialog box or control and paste it into your script.

1. Select the dialog box or control that you want to incorporate into your script.
2. Press **Ctrl+C**.
3. Open your script and paste in the contents of the Clipboard at the desired point.

The dialog box template or control is now described in statements in your script, as shown in the following example.

A screenshot of a Basic script editor window titled "Basic1 *". The window has a toolbar with various icons for execution and editing. The main text area contains the following code:

```
Sub Main()  
  Begin Dialog UserDialog ,,162,58,"Untitled"  
    OKButton 116,8,40,4  
    CancelButton 116,28,40,14  
    CheckBox 8,8,45,8,"CheckBox",.CheckBox1  
    CheckBox 8,20,45,8,"CheckBox",.CheckBox2  
    PushButton 8,32,48,14,"PushButton",.PushButton1  
  End Dialog  
  Dim d As UserDialog  
  Dialog d  
End Sub
```

The status bar at the bottom right shows "Ln 12 Col 0 Idle".

Exiting from Dialog Editor

Here's how to get out of Dialog Editor. When you exit, you can save your dialog box template (that is, the description of the dialog box) as a text file.

1. Press **Alt+F4**. If you have made changes to your dialog box template, Dialog Editor asks whether you want to save those changes.
2. If you want to save your changes to a text file, click the Yes button.

Dialog Editor displays the Save Dialog File dialog box, which you can use to specify the file to which you want to save your template.

Using a Custom Dialog Box in Your Script

After using Dialog Editor to insert a custom dialog box template into your script, you'll need to make the following modifications to your script:

1. Create a *dialog record* by using a **Dim** statement.
2. Put information into the dialog box by assigning values to its controls.
3. Display the dialog box by using either the Dialog() function or the Dialog statement.
4. Retrieve values from the dialog box after the user closes it.

Creating a Dialog Record

To store the values retrieved from a custom dialog box, you create a dialog record with a **Dim** statement, using the following syntax:

```
Dim DialogRecord As DialogVariable
```

Here are some examples of how to create dialog records:

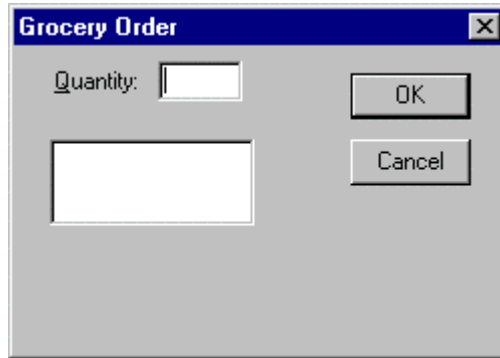
```
Dim b As UserDialog           'Define a dialog record "b".  
Dim PlayCD As CDDialog       'Define a dialog record  
                               "PlayCD".
```

Here is a sample script that illustrates how to create a dialog record named **b** within a dialog box template named **UserDialog**. Notice that the order of the statements within the script is as follows: the dialog box template precedes the statement that creates the dialog record, and the **Dialog** statement follows both of them.

```
Sub Main()  
    Dim ListBox1$()           'Initialize list box  
array.  
  
    'Define the dialog box template.  
    Begin Dialog UserDialog ,,163,94,"Grocery Order"  
        Text 13,6,32,8,"&Quantity:",.Text1  
        TextBox 48,4,28,12,.TextBox1  
        ListBox 12,28,68,32,ListBox1$,.ListBox1  
        OKButton 112,8,40,14  
        CancelButton 112,28,40,14  
    End Dialog  
  
    Dim b As UserDialog       'Create the dialog  
record.  
    Dialog b                 'Display the dialog  
box.  
End Sub
```

Putting Information into the Custom Dialog Box

If you open and run the sample script shown in the preceding subsection, you'll see a dialog box that resembles the following (the dialog box you see may be different from the one shown below):



As you can see, this custom dialog box isn't very useful. For one thing, the user doesn't see any items in the list box along the left side of the dialog box. To put information into this dialog box, you assign values to its controls by modifying the statements in your script that are responsible for displaying those controls to the user. The following table lists the dialog box controls to which you can assign values and the types of information you can control:

Control(s)	Types of Information
List box, drop list box, and combo box	Items
Text box	Default text
Check box	Values

In the following subsections, you'll learn how to define and fill an array, set the default text in a text box, and set the initial focus and tab order for the controls in your custom dialog box.

Defining and Filling an Array

You can store items in the list box shown in the example above by creating an array and then assigning values to the elements of the array. For example, you could include the following lines to initialize an array with three elements and assign the names of three common fruits to these elements of your array:

```
Dim ListBox1$(3)           'Initialize list box
array.
ListBox1$(0) = "Apples"
ListBox1$(1) = "Oranges"
ListBox1$(2) = "Pears"
```

Setting Default Text in a Text Box

You can set the default value of the text box in your script to 12 with the following statement, which must follow the statement that defines the dialog record but precede the statement or function that displays the custom dialog box:

```
b.TextBox1 = "12"
```

Setting the Initial Focus and Controlling the Tabbing Order

You can determine which control has the focus when your custom dialog box is first displayed as well as the tabbing order between controls by understanding two rules that the Basic Control Engine script follows. First, the focus in a custom dialog box is always set initially to the first control to appear in the dialog box template. Second, the order in which subsequent controls appear within the dialog box template determines the tabbing order. That is, pressing the **Tab** key will change the focus from the first control to the second one, pressing the **Tab** key again will change the focus to the third control, and so on.

Displaying the Custom Dialog Box

To display a custom dialog box, you can use either a **Dialog()** function or a **Dialog** statement.

Using the Dialog() Function

You can use a **Dialog()** function to determine how the user closed your custom dialog box. For example, the following statement will return a value when the user clicks an **OK** button or a **Cancel** button or takes another action:

```
response% = Dialog(b)
```

The **Dialog()** function returns any of the following values:

Value returned: If:

-1	The user clicked the OK button.
0	The user clicked the Cancel button.
>0	The user clicked a push button. The returned number represents which button was clicked based on its order in the dialog box template (1 is the first push button, 2 is the second push button, and so on).

Using the Dialog Statement

You can use the **Dialog** statement when you don't need to determine how the user closed your dialog box. You'll still be able to retrieve other information from the dialog box record, such as the value of a list box or other dialog box control.

An example of the correct use of the **Dialog** statement is:

```
Dialog mydlg
```

Where

Dialog is the statement that calls a declared dialog name.

mydlg is the dialog name in this example.

Retrieving Values from the Custom Dialog Box

After displaying a custom dialog box for your user, your script must retrieve the values of the dialog controls. You retrieve these values by referencing the appropriate identifiers in the dialog record.

You'll find an example of how to retrieve values from a custom dialog box in the following subsection.

Sample

In the following sample script several of the techniques described earlier in this section have been used.

In this script, the array named `ListBox1` is filled with three elements ("**Apples**", "**Oranges**", and "**Pears**"). The default value of `TextBox1` is set to 12. A variable named `response` is used to store information about how the custom dialog box was closed. An identifier named `ListBox1` is used to determine whether the user chose "**Apples**", "**Oranges**", or "**Pears**" in the list box named `ListBox1`. Finally, a `Select Case . . . End Select` statement is used to display a message box appropriate to the manner in which the user dismissed the dialog box.

```
Sub Main()  
    Dim ListBox1$(2)                'Initialize list box  
array.  
    Dim response%  
  
    ListBox1$(0) = "Apples"  
    ListBox1$(1) = "Oranges"  
    ListBox1$(2) = "Pears"  
  
    Begin Dialog UserDialog ,,163,94,"Grocery Order"  
        Text 13,6,32,8,"&Quantity:",.Text1 'First control in  
                                           'template gets the  
focus.  
        TextBox 48,4,28,12,.TextBox1  
        ListBox 12,28,68,32,ListBox1$,.ListBox1  
        OKButton 112,8,40,14  
        CancelButton 112,28,40,14  
    End Dialog  
  
    Dim b As UserDialog              'Create the dialog record.  
    b.TextBox1 = "12"               'Set default value of the text  
box  
                                     'to 1 dozen.  
    response = Dialog(b)           'Display the dialog box.  
  
    Select Case response%  
        Case -1  
            Fruit$ = ListBox1$(b.ListBox1)  
            MsgBox "Thank you for ordering " + b.TextBox1 + "  
" + Fruit$ + "."  
        Case 0  
            MsgBox "Your order has been canceled."  
    End Select  
End Sub
```

Using a Dynamic Dialog Box in Your Script

The preceding section explained how you can use a custom dialog box in your script. As you learned, you can retrieve the values from dialog box controls after the user dismisses the dialog box by referencing the identifiers in the dialog record.

You can also retrieve values from a custom dialog box while the dialog box is displayed, using the *dynamic dialog boxes* feature.

The following script illustrates the most important concepts you'll need to understand in order to create a dynamic dialog box in your script:

```
'Dim "Fruits" and "Vegetables" arrays here to make them accessible
to
'all procedures.
Dim Fruits(2) As String
Dim Vegetables(2) As String

'Dialog procedure--must precede the procedure that defines the
custom
'dialog box.
Function DialogControl(ctrl$, action%, suppvale%) As Integer
    Select Case action%
        Case 1
            DlgListBoxArray "ListBox1", fruits 'Fill list box with
                                                'items before dialog
                                                'box is visible.
            DlgValue "ListBox1", 0 'Set default value to
                                    'first item in list box.

            Case 2
                'Fill the list box with names of fruits or
                vegetables
                'when the user selects an option button.
                If ctrl$ = "OptionButton1" Then
                    DlgListBoxArray "ListBox1", fruits
                    DlgValue "ListBox1", 0

                    ElseIf ctrl$ = "OptionButton2" Then
                        DlgListBoxArray "ListBox1", vegetables
                        DlgValue "ListBox1", 0
                    End If
                End Select
            End Function

Sub Main()
    Dim ListBox1$() 'Initialize array for use by ListBox
                    'statement in template.

    Dim Produce$

'Assign values to elements in the "Fruits" and "Vegetables"
'arrays.
    Fruits(0) = "Apples"
    Fruits(1) = "Oranges"
    Fruits(2) = "Pears"
    Vegetables(0) = "Carrots"
    Vegetables(1) = "Peas"
    Vegetables(2) = "Lettuce"

'Define the dialog box template.
Begin Dialog UserDialog ,,163,94,"Grocery Order",.DialogControl
    Text 13,6,32,8,"&Quantity:",.Text1 'First control in
```

```

                                                                    'template gets the
focus.
    TextBox 48,4,28,12,.TextBox1
    ListBox 12,28,68,32,ListBox1$,.ListBox1
    OptionGroup .OptionGroup1
        OptionButton 12,68,48,8,"&Fruit",.OptionButton1
        OptionButton 12,80,48,8,"&Vegetables",.OptionButton2
    OKButton 112,8,40,14
    CancelButton 112,28,40,14
End Dialog

    Dim b As UserDialog          'Create the dialog record.
    b.TextBox1 = "12"           'Set the default value of the
text
                                                                    'box to 1 dozen.
    response% = Dialog(b)       'Display the dialog box.

    Select Case response%
    Case -1
        If b.OptionGroup1 = 0 Then
            produce$ = fruits(b.ListBox1)
        Else
            produce$ = vegetables(b.ListBox1)
        End If

        MsgBox "Thank you for ordering " & b.TextBox1 & "
" & produce$ & "."

    Case 0
        MsgBox "Your order has been canceled."
    End Select
End Sub

```

In the remainder of this section, you'll learn how to make a dialog box dynamic by examining the workings of this sample script.

Making a Dialog Box Dynamic

The first thing to notice about the preceding script, which is a more complex variation of the `dialog2.bcl` script described earlier in this chapter, is that an identifier named `.DialogControl` has been added to the **Begin Dialog** statement. As you will learn in the following subsection, this parameter to the **Begin Dialog** statement tells the Basic Control Engine to pass control to a function procedure named `DialogControl`.

Using a Dialog Function

Before the Basic Control Engine displays a custom dialog box by executing a **Dialog** statement or `Dialog()` function, it must first initialize the dialog box. During this initialization process, the Basic Control Engine checks to see whether you've defined a dialog function as part of your dialog box template. If so, the Basic Control Engine will give control to your dialog function, allowing your script to carry out certain actions, such as hiding or disabling dialog box controls.

After completing its initialization process, the Basic Control Engine displays your custom dialog box. When the user selects an item in a list box, clears a check box, or carries out certain other actions within the dialog box, the Basic Control Engine will again call your dialog function.

Responding to User Actions

The Basic Control Engine dialog function can respond to six types of user actions:


Action	Description
1	This action is sent immediately before the dialog box is shown for the first time.
2	This action is sent when: A button is clicked, such as OK , Cancel , or a push button. A check box's state has been modified An option button is selected. In this case, <i>ControlName\$</i> contains the name of the option button that was clicked, and <i>SuppValue</i> contains the index of the option button within the option button group (0 is the first option button, 1 is the second, and so on). The current selection is changed in a list box, drop list box, or combo box. In this case, <i>ControlName\$</i> contains the name of the list box, combo box, or drop list box, and <i>SuppValue</i> contains the index of the new item (0 is the first item, 1 is the second, and so on).
3	This action is sent when the content of a text box or combo box has been changed <i>and</i> that control loses focus.
4	This action is sent when a control gains the focus.
5	This action is sent continuously when the dialog box is idle. The user should return a 0 or idle processing will use up the CPU.
6	This action is sent when the dialog box is moved.

You'll find a more complete explanation of these action codes in the *A–Z Reference*. See the **DlgProc** (Function) entry in that chapter.

Menu/Tools Reference

File Menu

The File menu commands are:


<u>Menu</u> <u>Command</u>	<u>Toolbar</u> <u>Tool</u>	<u>Keyboard</u> <u>Shortcut</u>	<u>Function</u>
<u>N</u>ew			Creates a new dialog box. Dialog Editor prompts you before discarding any changes you have made to your current dialog box.
<u>O</u>pen...			Displays the Open Dialog File dialog box, which you can use to open an existing dialog box template. Dialog Editor prompts you before discarding any changes you have made to your current dialog box.
<u>S</u>ave			If you have already created a file for the current dialog box template, saves the template to that file. If you have not yet created a file for the current dialog box template, displays the Save Dialog File dialog box, which you can use to specify the file to which you want to save the current template.
Save <u>A</u>s...			Displays the Save Dialog File dialog box, which you can use to save the current dialog box template in a file under a new name.
<u>T</u>est Dialog		F5	Toggles between the run mode (in which the dialog box "comes alive" for testing purposes) and the edit mode (in which you can make changes to the dialog box).
<u>C</u>apture Dialog			Captures the standard Windows controls from a standard Windows dialog box in another Windows application.
<u>E</u>xit		Alt+F4	Closes Dialog Editor. Dialog Editor prompts you before discarding any changes you have made to your current dialog box.

Exit & Return	Alt+F4	Closes Dialog Editor and returns you to the host application. Dialog Editor prompts you before discarding any changes you have made to your current dialog box.
--------------------------	---------------	---

Edit Menu






The Edit menu commands are:

<u>Menu Command</u>	<u>Toolbar Tool</u>	<u>Keyboard Shortcut</u>	<u>Function</u>
<u>U</u> ndo		Ctrl+Z	Allows you to undo up to 10 preceding operations. The Undo command continually indicates the next operation you can undo by selecting it and grays out when there are no more operations that can be undone.
<u>C</u> ut		Ctrl+X	Removes the selected dialog box or control from Dialog Editor's application window and places it on the Clipboard.
<u>C</u> opy		Ctrl+C	Copies the selected dialog box or control, without removing it from Dialog Editor's application window, and places it on the Clipboard.
<u>P</u> aste		Ctrl+V	<p>Inserts the contents of the Clipboard into Dialog Editor.</p> <p>If the Clipboard contains statements describing one or more controls, then those controls are added to the current dialog box. If the Clipboard contains the template for an entire dialog box, then Dialog Editor creates a new dialog box from the statements in the template.</p> <p>If the statements contain errors, Dialog Editor displays the Dialog Translation Errors dialog box, which helps you pinpoint the location and nature of the errors.</p>

<u>D</u>el		Delete	Removes the selected dialog box or control from Dialog Editor's application window without placing it on the Clipboard. If you have selected the dialog box and it contains more than one control, Dialog Editor prompts you before removing all the controls from it.
Duplicate		Ctrl+D	Creates a duplicate copy of the selected control.
Size to Text		F3	Adjusts the borders of certain controls to fit the text displayed on them.
<u>I</u>nf...		Ctrl+I	Displays the Information dialog box for the selected dialog box or control. You can use this dialog box to check and adjust various attributes of controls and dialog boxes.
<u>G</u>rid...		Ctrl+G	Displays the <i>Grid</i> dialog box, which you can use to display or turn off the grid and adjust the grid's spacing.

Controls Menu

The Control menu commands are:

<u>M</u>enu <u>C</u>ommand	<u>T</u>oolbar <u>T</u>ool	<u>F</u>unction
<u>O</u>K button		Adds a default OK button to your dialog box. An OK button is a special type of push, or command, button.
<u>C</u>ancel button		Adds a default Cancel button to your dialog box. A Cancel button is a special type of push, or command, button.
<u>P</u>ush button		Adds a push, or command, button to your dialog box.
<u>O</u>ption button		Adds an option button to your dialog box. An option button is one of a group of two or more linked buttons that let users select only one from a group of mutually exclusive choices.
<u>C</u>heck box		Adds a check box to your dialog box. Users can check or clear a check box to indicate their preference regarding the alternative specified on the check box label.

Group box

Adds a group box to your dialog box. A group box is a rectangular design element used to enclose a group of related controls. You can use the optional group box label to display a title for the controls in the box.

Text

Adds a text control to your dialog box. A text control is a field containing text that you want to display for the users' information. The text in this field wraps, and the field can contain a maximum of 255 characters. Text controls can either display stand-alone text or be used as labels for text boxes, list boxes, combo boxes, drop list boxes, pictures, and picture buttons. You can choose the font in which the text appears.

Text box

Adds a text box to your dialog box. A text box is a field into which users can enter text (potentially, as much as 32K). By default, this field holds a single line of nonwrapping text. If you choose the Multiline setting in the Text Box Information dialog box, this field will hold multiple lines of wrapping text.

List box

Adds a list box to your dialog box. A list box is a displayed, scrollable list from which users can select one item. The currently selected item is highlighted on the list.

Combo box

Adds a combo box to your dialog box. A combo box consists of a text field with a displayed, scrollable list beneath it. Users can either select an item from the list or enter the name of the desired item in the text field. The currently selected item is displayed in the text field. If the item was selected from the scrolling list, it is highlighted there as well.

Drop list box

Adds a drop list box to your dialog box. A drop list box consists of a field that displays the currently selected item, followed by a downward-pointing arrow, which users can click to temporarily display a scrolling list of items. Once they select an item from the list, the list disappears and the newly selected item is displayed in the field.

Picture

Adds a picture to your dialog box. A picture is a field used to display a Windows bitmap or metafile.

Picture button

Adds a picture button to your dialog box. A picture button is a special type of push, or command, button on which a Windows bitmap or metafile appears.


Help Menu

The Help menu commands are:

<u>Menu</u> <u>Command</u>	<u>Keyboard</u> <u>Shortcut</u>	<u>Function</u>
<u>C</u>ontents	F1	Presents a list of major topics in the Help system. By clicking a topic on the list, you can display the available Help information about that topic.
<u>S</u>earch for Help On...		Displays a dialog box that allows users to search for Help topics containing specific keywords.
<u>A</u>bout Dialog Editor...		Displays the About Dialog Editor dialog box, which indicates application name, version number, copyright statement, and icon, as well as additional information such as amount of available memory and disk space.

Pick Tool

The Pick toolbar tool looks like this:

Toolbar Tool	Function
	Lets you select, move, and resize items and control the insertion point.

Debugging Your Scripts

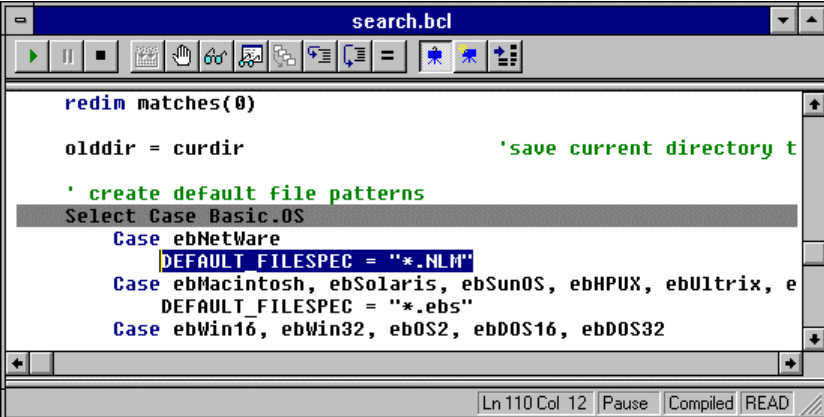
About the Debugger

This section presents some general information that will help you work most effectively with Program Editor's debugging capabilities and then explains how to trace the execution of your script, how to set and remove breakpoints, and how to add watch variables and modify their value.

Using the Debugger

While debugging, you are actually executing the code in your script line by line. Therefore, to prevent any modifications to your script while it is being run, the edit pane is read-only during the debugging process. You are free to move the insertion point throughout the script, select text and copy it to the Clipboard as necessary, set breakpoints, and add and remove watch variables, but you cannot make any changes to the script until you stop running it.

To let you follow and control the debugging process, Program Editor displays an *instruction pointer* on the line of code that is about to be executed—that is, the line that will be executed next if you either proceed with the debugging process or run your script at full speed. When the instruction pointer is on a line of code, the text on that line appears in black on a gray background that spans the width of the entire line. The following illustration shows the difference between the instruction pointer and the selection highlight (discussed in the preceding section), in which the text appears in white on a black or blue background that spans only the width of the selected text.



The screenshot shows a debugger window titled "search.bcl". The window contains a script with the following code:

```
redim matches(0)
olddir = curdir           'save current directory t
' create default file patterns
Select Case Basic.OS
  Case ebNetWare
    DEFAULT_FILESPEC = "*.NLM"
  Case ebMacintosh, ebSolaris, ebSunOS, ebHPUX, ebUltrix, e
    DEFAULT_FILESPEC = "*.ebs"
  Case ebWin16, ebWin32, ebOS2, ebDOS16, ebDOS32
```

The line "Select Case Basic.OS" is highlighted with a gray background, indicating it is the instruction pointer. The line " DEFAULT_FILESPEC = "*.NLM"" is highlighted with a blue background, indicating it is the selection highlight.

The status bar at the bottom of the window shows "Ln 110 Col 12 | Pause | Compiled | READ".

Fabricating Event Information

The Event Editor allows the user to configure a script to run in response to an event. When a project is running, the Event Manager runs a script either when a specified event or any event occurs, depending on what is specified in the script.

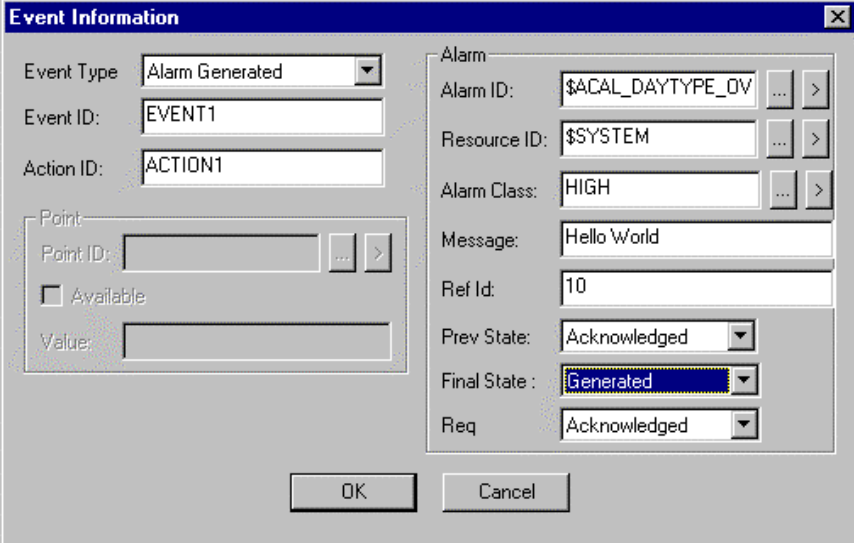
However, on one hand, when you build the script in the Program Editor there is no "real" event to trigger the script. On the other hand, when the script runs with the Event Manager, you can't debug it in the program editor.

Therefore the Program Editor provides you with an *Event Information* dialog box. The *Event Information* dialog box allows you to fabricate the event information that the APIs would provide in a real environment. Using this fabrication you can safely test the accuracy of your script.

To open the Event Information dialog box:

1. Select **Debug** on the Program Editor menu bar.
2. Select **Set Event Information**.

The Event Information dialog box opens.



Because you are using the *Event Information* dialog box to fabricate a "real world" environment, what you enter depends on what is included in your script. If the script includes specific entries for any of the fields, you have to enter what is in the script. For any entries that are not specifically referred to in the script, you can enter whatever you want.

Example

A script defines the:

Event Type as Alarm Generated

Resource ID as \$SYSTEM.

As a result, **Alarm Generated** and **\$SYSTEM** are selected in the *Event Information* dialog box.

Entries in the other fields are fictitious.

Whenever, you run the script, it will draw its information from what you entered. You only have to change your entries in the *Event Information* dialog box if a script requires a change in a specific entry. See "GFK-1282, *CIMPLICITY HMI Basic Control Engine–Event Editor and BCEUI Operation Manual*" for information on configuring events.

Tracing Script Execution

Program Editor gives you two ways to trace script execution—single step and procedure step—both of which involve stepping through your script code line by line. The distinction between the two is that the single step process traces into calls to user-defined functions and subroutines, whereas the procedure step process does not trace into these calls (although it does execute them).

Stepping Through the Script

Use one of these methods to trace the execution of your script with either the single step or procedure step method.

- Click the **Step** tool on the toolbar, or press **F8** (Single Step).
- Press **Shift+F8** (Procedure Step).

Program Editor places the instruction pointer on the **sub main** line of your script.

Note: When you initiate execution of your script with any of these methods, the script will first be compiled, if necessary. Therefore, there may be a slight pause before execution actually begins. If your script contains any compile errors, it will not be executed. To debug your script, first correct any compile errors, then initiate execution again.

To continue tracing the execution of your script line by line, repeat the trace command. Each time you repeat the trace command, Program Editor executes the line containing the instruction pointer and moves the instruction pointer to the next line to be executed.

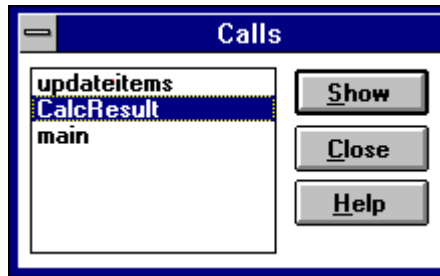
When you finish tracing the execution of your script, you can do one of the following:

- Click the **Start** tool on the toolbar (or press **F5**) to run the balance of the script at full speed.
- Click the **End** tool to halt execution of the script.

Displaying the Calls Dialog Box

When you are stepping through a subroutine, you may need to determine the procedure calls by which you arrived at that point in your script. Here's how to use the Calls dialog box to obtain this information.

1. Click the Calls tool on the toolbar. Program Editor displays the Calls dialog box, which lists the procedure calls made by your script in the course of arriving at the present subroutine.



2. From the Calls dialog box, select the name of the procedure you wish to view.
3. Click the Show button.

Program Editor highlights the currently executing line in the procedure you selected, scrolling that line into view if necessary. (During this process, the instruction pointer remains in its original location in the subroutine.)

Moving the Pointer to Another Line

When you are stepping through a subroutine, you may want to repeat or skip execution of a section of code. Here's how to use the Set Next Statement command to move the instruction pointer to another line within that subroutine.

1. Place the insertion point in the line where you want to resume stepping through the script.
2. From the **Debug** menu, choose the **Set Next Statement** command.

The instruction pointer moves to the line you selected, and you can resume stepping through your script from there.

Note

You can only use the Set Next Statement command to move the instruction pointer within the same subroutine.

Setting and Removing Breakpoints

If you want to start the debugging process at the first line of your script and then step through your code line by line until you reach the end of the code that you need to debug, the method described in the preceding subsection works fine. But if you only need to debug one or more portions of a long script, that method can be pretty cumbersome.

An alternate strategy is to set one or more *breakpoints* at selected lines in your script. Program Editor suspends execution of your script just before it reaches a line containing a breakpoint, thereby allowing you to begin or resume stepping through the script from that point.

Setting Breakpoints

You can set breakpoints to begin the debugging process partway through your script, to continue debugging at a line outside the current subroutine, and to debug only selected portions of your script.

Valid breakpoints can only be set on lines in your script that contain code, including lines in functions and subroutines. Although you can set a breakpoint anywhere within a script prior to execution, when you compile and run the script, invalid breakpoints (that is, breakpoints on lines that don't contain code) are automatically removed. While you are debugging your script, Program Editor will beep if you try to set a breakpoint on a line that does not contain code.

Start Debugging at a Select Point

Here's how to begin the debugging process at a selected point in your script.

1. Place the insertion point in the line where you want to start debugging.
2. To set a breakpoint on that line, click the **Toggle Breakpoint** tool on the toolbar, *or* press **F9**. The line on which you set the breakpoint now appears in contrasting type.
3. Click the **Start** tool on the toolbar *or* press **F5**.

Program Editor runs your script at full speed from the beginning and then pauses prior to executing the line containing the breakpoint. It places the instruction pointer on that line to designate it as the line that will be executed next when you either proceed with debugging or resume running the script.

If you want to continue debugging at another line in your script, you can use the **Set Next Statement** command in the Debug menu to move the instruction pointer to the desired line—provided the line is within the same subroutine.

Start Debugging at a Line Outside the Current Subroutine

If you want to continue debugging at a line that isn't within the same subroutine, here's how to move the instruction pointer to that line.

1. Place the insertion point in the line where you want to continue debugging.
2. To set a breakpoint on that line, press **F9**.
3. To run your script, click the Start tool on the toolbar or press **F5**.

The script executes at full speed until it reaches the line containing the breakpoint and then pauses with the instruction pointer on that line. You can now resume stepping through your script from that point.

Debugging Selected Portions of a Program

If you only need to debug parts of your script, here's how to facilitate the task by using breakpoints.

1. Place a breakpoint at the start of each portion of your script that you want to debug.

Note

Up to 255 lines in your script can contain breakpoints.

2. To run the script, click the Start tool on the toolbar or press **F5**. The script executes at full speed until it reaches the line containing the first breakpoint and then pauses with the instruction pointer on that line.
3. Step through as much of the code as you need to.
4. To resume running your script, click the Start tool on the toolbar or press **F5**. The script executes at full speed until it reaches the line containing the second breakpoint and then pauses with the instruction pointer on that line.
5. Repeat steps 3 and 4 until you have finished debugging the selected portions of your script.

Removing Breakpoints

Breakpoints can be removed either manually or automatically.

To remove a single breakpoint manually:

Here's how to delete breakpoints manually one at a time.

1. Place the insertion point on the line containing the breakpoint that you want to remove.
2. Click the **Toggle Breakpoint** tool on the toolbar *or* press **F9**.

The breakpoint is removed, and the line no longer appears in contrasting type.

To remove all breakpoints manually:

Here's how to delete all breakpoints manually in a single operation.

- Select the **Clear All Breakpoints** command from the **Debug** menu.

Program Editor removes all breakpoints from your script.

Breakpoints are removed automatically under the following circumstances: (1) As mentioned earlier, when your script is compiled and executed, breakpoints are removed from lines that don't contain code. (2) When you exit from Program Editor, all breakpoints are cleared.

Using a Watch Variable

As you debug your script, you can use Program Editor's watch pane to monitor selected variables. For each of the variables on this watch variable list, Program Editor displays the name of the variable, where it is defined, its value (if the variable is not in scope, its value is shown as <not in context>), and other key information such as its type and length (if it is a string). The values of the variables on the watch list are updated each time you enter break mode.

Adding A Watch Variable

Here's how to add a variable to Program Editor's watch variable list.

1. Click the **Add Watch** tool on the toolbar *or* press **Shift+F9**.

Program Editor displays the Add Watch dialog box.



2. In the **Variable** field, enter the name of the variable you want to add to the watch variable list.
3. In the **Procedure** field, enter the procedure name.
4. In the **Script** field, enter the script name.

You can only watch variables of fundamental data types, such as **Integer**, **Long**, **Variant**, and so on; you cannot watch complex variables such as structures or arrays. You can, however, watch individual elements of arrays or structure members using the following syntax:

[*variable* [(*index*,...)] [*.member* [(*index*,...)]]...]

Where *variable* is the name of the structure or array variable, *index* is a literal number, and *member* is the name of a structure member.

For example, the following are valid watch expressions:

<u>Watch Variable</u>	<u>Description</u>
a(1)	Element 1 of array a
person.age	Member age of structure person
company(10,23).person.age	Member age of structure person that is at element 10,23 within the array of structures called company

Note

If you are executing the script, you can display the names of all the variables that are "in scope," or defined within the current function or subroutine, on the drop-down Variable Name list and select the variable you want from that list.

5. Click the **OK** button or press **Enter**.

If this is the first variable you are placing on the watch variable list, the watch pane opens far enough to display that variable. If the watch pane was already open, it expands far enough to display the variable you just added.

Note

Although you can add as many watch variables to the list as you want, the watch pane only expands until it fills half of Program Editor's application window. If your list of watch variables becomes longer than that, you can use the watch pane's scroll bars to bring hidden portions of the list into view.

The list of watch variables is maintained between script executions.

Selecting A Watch Variable

In order to delete a variable from Program Editor's watch variable list or modify the value of a variable on the list, you must first select the desired variable. Use one of these methods to select a variable on the list.

- Place the mouse pointer on the variable you want to select and click the left mouse button.
- If one of the variables on the watch list is already selected, use the arrow keys to move the selection highlight to the desired variable.
- If the insertion point is in the edit pane, press **F6** to highlight the most recently selected variable on the watch list and then use the arrow keys to move the selection highlight to the desired variable.

Note

Pressing **F6** again returns the insertion point to its previous position in the edit pane.

Deleting a Watch Variable

Here's how to delete a selected variable from Program Editor's watch variable list.

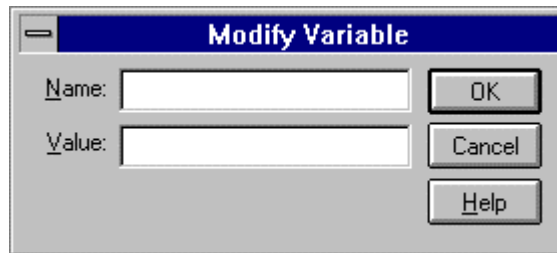
1. Select the variable on the watch list.
2. Press **Delete**.

The selected variable is removed from the watch list.

Modifying the Value of a Variable

When the debugger has control, you can modify the value of any of the variables on Program Editor's watch variable list. Here's how to change the value of a selected watch variable.

1. Select the name of the variable whose value you want to modify and press the Modify button or select Modify from the Debug menu. Program Editor displays the *Modify Variable* dialog box.



Note

The name of the variable you selected on the watch variable list appears in the **Name** field. If you want to change another variable, you can either enter a different variable in the **Name** field or select a different variable from the **Variables** list box, which shows the names of the variables that are defined within the current function or subroutine.

When you use the *Modify Variable* dialog box to change the value of a variable, you don't have to specify the context. Program Editor first searches locally for the definition of that variable, then privately, then publicly.

2. Enter the new value for your variable in the **Value** field.
3. Click the **OK** button.

The new value of your variable appears on the watch variable list.

When changing the value of a variable, Program Editor converts the new value to be of the same type as the variable being changed. For example, if you change the value of an **Integer** variable to 1.7, a conversion between a floating-point number and an **Integer** is performed, assigning the value 2 to the **Integer** variable.

When modifying a **Variant** variable, Program Editor needs to determine both the type and value of the data. Program Editor uses the following logic in performing this assignment (in this order):

<u>If the new value is</u>	<u>Then</u>
Null	The Variant variable is assigned Null (VarType 1)
Empty	The Variant variable is assigned Empty (VarType 0).
True	The Variant variable is assigned True (VarType 11).
False	The Variant variable is assigned False (VarType 11).
<i>number</i>	The Variant variable is assigned the value of <i>number</i> . The type of the variant is the smallest data type that fully represents that number. You can force the data type of the variable using a type-declarator letter following <i>number</i> , such as %, #, &, !, or @.
<i>date</i>	The Variant variable is assigned the value of the new date (VarType 7)
Anything else	The Variant variable is assigned a String (VarType 8).

Program Editor will not assign a new value if it cannot be converted to the same type as the specified variable.

Running Your Programs

Running A Program

Once you have finished editing your programs, you will want to run it to make sure it performs the way you intended. You can also pause or stop an executing script.

Here's how to compile your script, if necessary, and then execute it.

- Click the **Start** tool on the toolbar.
- Press **F5**.

The script is compiled (if it has not already been compiled), the focus is switched to the parent window, and the script is executed.

Suspending A Running Program

Here's how to suspend the execution of a script that you are running.

- Press **Ctrl+Break**. or press the **Break** toolbar button.

Execution of the script is suspended, and the instruction pointer (a gray highlight) appears on the line of code where the script stopped executing.

Note

The instruction pointer designates the line of code that will be executed next if you resume running your script.

Stopping A Running Program

Here's how to stop the execution of a script that you are running.

- Click the **End** tool on the toolbar.

Appendix A - Runtime Error Messages

Introduction

This section contains listings of all the runtime errors. It is divided into two subsections, the first describing errors messages compatible with "standard" Basic as implemented by Microsoft Visual Basic and the second describing error messages specific to the Basic Control Engine.

A few error messages contain placeholders which get replaced by the runtime when forming the completed runtime error message. These placeholders appear in the following list as the italicized word *placeholder*.

Visual Basic Compatible Error Messages

The Visual Basic compatible error messages are:

<u>Error Number</u>	<u>Error Message</u>
3	Return without GoSub
5	Illegal procedure call
6	Overflow
7	Out of memory
9	Subscript out of range
10	This array is fixed or temporarily locked
11	Division by zero
13	Type mismatch
14	Out of string space
19	No Resume
20	Resume without error
26	Dialog needs End Dialog or push button
28	Out of stack space
35	Sub or Function not defined
48	Error in loading DLL
49	Bad DLL calling convention
51	Internal error
52	Bad file name or number
53	File not found
54	Bad file mode
55	File already open
57	Device I/O error
58	File already exists
59	Bad record length
61	Disk full
62	Input past end of file
63	Bad record number
64	Bad file name
67	Too many files
68	Device unavailable
70	Permission denied
71	Disk not ready

<u>Error Number</u>	<u>Error Message</u>
74	Can't rename with different drive
75	Path/File access error
76	Path not found
91	Object variable or With block variable not set
93	Invalid pattern string
94	Invalid use of Null
139	Only one user dialog may be up at any time
140	Dialog control identifier does not match any current control
141	The <i>placeholder</i> statement is not available on this dialog control type
143	The dialog control with the focus may not be hidden or disabled
144	Focus may not be set to a hidden or disabled control
150	Dialog control identifier is already defined
163	This statement can only be used when a user dialog is active
260	No timer available
281	No more DDE channels
282	No foreign application responded to a DDE initiate
283	Multiple applications responded to a DDE initiate
285	Foreign application won't perform DDE method or operation
286	Timeout while waiting for DDE response
287	User pressed Escape key during DDE operation
288	Destination is busy
289	Data not provided in DDE operation
290	Data in wrong format
291	Foreign application quit
292	DDE conversation closed or changed
295	Message queue filled; DDE message lost
298	DDE requires ddeml.dll
429	OLE Automation server can't create object
430	Class doesn't support OLE Automation
431	OLE Automation server cannot load file
432	File name or class name not found during OLE Automation operation

<u>Error Number</u>	<u>Error Message</u>
433	OLE Automation object does not exist
434	Access to OLE Automation object denied
435	OLE initialization error
436	OLE Automation method returned unsupported type
437	OLE Automation method did not return a value
438	Object doesn't support this property or method <i>placeholder</i>
439	OLE Automation argument type mismatch <i>placeholder</i>
440	OLE Automation error <i>placeholder</i>
443	OLE Automation Object does not have a default value
452	Invalid ordinal
460	Invalid Clipboard format
520	Can't empty clipboard
521	Can't open clipboard
600	Set value not allowed on collections
601	Get value not allowed on collections
603	ODBC - SQLAllocEnv failure
604	ODBC - SQLAllocConnect failure
608	ODBC - SQLFreeConnect error
610	ODBC - SQLAllocStmnt failure
3129	Invalid SQL statement; expected 'DELETE', 'INSERT', 'PROCEDURE', 'SELECT', or 'UPDATE'
3146	ODBC - call failed
3148	ODBC - connection failed
3276	Invalid database ID

Basic Control Engine-Specific Error Messages

The Basic Control Engine-specific error messages are:

<u>Error Number</u>	<u>Error Message</u>
800	Incorrect Windows version
801	Too many dimensions
802	Can't find window
803	Can't find menu item
804	Another queue is being flushed
805	Can't find control
806	Bad channel number
807	Requested data not available
808	Can't create pop-up menu
809	Message box canceled
810	Command failed
811	Network error
812	Network function not supported
813	Bad password
814	Network access denied
815	Network function busy
816	Queue overflow
817	Too many dialog controls
818	Can't find list box/combo box item
819	Control is disabled
820	Window is disabled
821	Can't write to ini file
822	Can't read from ini file
823	Can't copy file onto itself
824	OLE Automation unknown object name
825	Can't redimension a fixed array
826	Can't load and initialize extension
827	Can't find extension
828	Unsupported function or statement
829	Can't find ODBC libraries
830	OLE Automation Lbound or Ubound on non-Array value
831	Incorrect definition for dialog procedure

Appendix B - Compiler Error Messages

Error Message List

The following table contains a list of all the errors generated by the Basic Control Engine compiler. With some errors, the compiler changes placeholders within the error to text from the script being compiled. These placeholders are represented in this table by the word *placeholder*.

- | | |
|----|--|
| 1 | Variable Required - Can't assign to this expression |
| 2 | Letter range must be in ascending order |
| 3 | Redefinition of default type |
| 4 | Out of memory, too many variables defined |
| 5 | Type-character doesn't match defined type |
| 6 | Expression too complex |
| 7 | Cannot assign whole array |
| 8 | Assignment variable and expression are different types |
| 10 | Array type mismatch in parameter |
| 11 | Array type expected for parameter |
| 12 | Array type unexpected for parameter |
| 13 | Integer expression expected for an array index |
| 14 | Integer expression expected |
| 15 | String expression expected |
| 18 | Left of "." must be an object, structure, or dialog |
| 19 | Invalid string operator |
| 20 | Can't apply operator to array type |
| 21 | Operator type mismatch |
| 22 | "placeholder" is not a variable |

23 "*placeholder*" is not a array variable or a function
24 Unknown *placeholder* "*placeholder*"
25 Out of memory
26 *placeholder*: Too many parameters encountered
27 *placeholder*: Missing parameter(s)
28 *placeholder*: Type mismatch in parameter *placeholder*
29 Missing label "*placeholder*"
30 Too many nested statements
31 Encountered new-line in string
32 Overflow in decimal value
33 Overflow in hex value
34 Overflow in octal value
35 Expression is not constant
37 No type-characters allowed on parameters with explicit type
39 Can't pass an array by value
40 "*placeholder*" is already declared as a parameter
41 Variable name used as label name
42 Duplicate label
43 Not inside a function
44 Not inside a sub
46 Can't assign to function
47 Identifier is already a variable
48 Unknown type
49 Variable is not an array type
50 Can't redimension an array to a different type
51 Identifier is not a string array variable
52 0 expected
55 Integer expression expected for file number
56 *placeholder* is not a method of the object
57 *placeholder* is not a property of the object
58 Expecting 0 or 1
59 Boolean expression expected
60 Numeric expression expected
61 Numeric type FOR variable expected
62 For...Next variable mismatch
63 Out of string storage space

64	Out of identifier storage space
65	Internal error 1
66	Maximum line length exceeded
67	Internal error 3
68	Division by zero
69	Overflow in expression
70	Floating-point expression expected
72	Invalid floating-point operator
74	Single character expected
75	Subroutine identifier can't have a type-declaration character
76	Script is too large to be compiled
77	Variable type expected
78	Can't evaluate expression
79	Can't assign to user or dialog type variable
80	Maximum string length exceeded
81	Identifier name already in use as another type
84	Operator cannot be used on an object
85	<i>placeholder</i> is not a property or method of the object
86	Type-character not allowed on label
87	Type-character mismatch on routine <i>placeholder</i>
88	Destination name is already a constant
89	Can't assign to constant
90	Error in format of compiler extensions
91	Identifier too long
92	Expecting string or structure expression
93	Can't assign to expression
94	Dialog and Object types are not supported in this context
95	Array expression not supported as parameter
96	Dialogs, objects, and structures expressions are not supported as a parameter
97	Invalid numeric operator
98	Invalid structure element name following "."
99	Access value can't be used with specified mode
101	Invalid operator for object
102	Can't LSet a type with a variable-length string
103	Syntax error
104	<i>placeholder</i> is not a method of the object

- 105 No members defined
- 106 Duplicate type member
- 107 Set is for object assignments
- 108 Type-character mismatch on variable
- 109 Bad octal number
- 110 Bad number
- 111 End-of-script encountered in comment
- 112 Misplaced line continuation
- 113 Invalid escape sequence
- 114 Missing End Inline
- 115 Statement expected
- 116 ByRef argument mismatch
- 117 Integer overflow
- 118 Long overflow
- 119 Single overflow
- 120 Double overflow
- 121 Currency overflow
- 122 Optional argument must be Variant
- 123 Parameter must be optional
- 124 Parameter is not optional
- 125 Expected: Lib
- 126 Illegal external function return type
- 127 Illegal function return type
- 128 Variable not defined
- 129 No default property for the object
- 130 The object does not have an assignable default property
- 131 Parameters cannot be fixed length strings
- 132 Invalid length for a fixed length string
- 133 Return type is different from a prior declaration
- 134 Private variable too large. Storage space exceeded
- 135 Public variables too large. Storage space exceeded

Index

A

- About Editing Programs 3-1
- accelerator keys, in Dialog Editor 4-13
 - assigning to dialog controls 4-22–4-23
 - testing 4-32
- actions, dialog 4-40
- arrays
 - list of language elements 1-4

B

- BasicScript
 - functions to get information from 1-10
- binary operators
 - list of 1-11
- Breakpoints
 - Program Editor 2-2
- breakpoints, in Program Editor 5-5
 - removing 5-7
 - setting 5-5–5-6
- buttons
 - on toolbar
 - in Dialog Editor 4-4

C

- Cancel buttons
 - in Dialog Editor 4-7
- capturing dialog boxes from another application, in Dialog Editor 4-29–4-30
- check boxes
 - in Dialog Editor 4-8
- Clipboard
 - list of language elements 1-4
- combo boxes
 - in Dialog Editor 4-8
- comments
 - adding to scripts, in Program Editor 3-8
 - list of language elements 1-4

- comparison operators
 - list of 1-4
- compiler errors B-1–B-4
- constants
 - list of language elements 1-15
- control structures
 - list of 1-5
- controlling applications
 - list of language elements 1-5
- copying
 - text
 - in Program Editor 3-7
- creating dialog boxes, in Dialog Editor 4-7–4-13
- cutting text, in Program Editor 3-7

D

- data conversion
 - list of language elements 1-6
- data types
 - list of 1-7
- database
 - list of language elements 1-7
- date/time functions
 - list of language elements 1-7
- DDE
 - list of language elements 1-8
- debugging scripts, in Program Editor 5-1
 - breakpoints 5-5
 - removing 5-7
 - setting 5-5–5-6
 - instruction pointer 5-1
 - moving to another line in subroutine 5-4
 - procedure calls, tracing 5-4
 - script execution, tracing 5-2–5-3
 - watch variables 5-8
 - adding 5-8–5-9
 - deleting 5-10
 - modifying value of 5-10
 - selecting 5-9
- deleting controls, in Dialog Editor 4-27
- deleting text, in Program Editor 3-6
- dialog actions 4-40

- dialog boxes, in Dialog Editor *See also* dialog controls;
 - dialog controls, in Dialog Editor; Dialog Editor;
 - user dialogs
- attributes of, adjusting with Information
 - dialog box 4-17
- capturing from another application 4-29–4-30
- creating 4-7–4-13
- editing 4-1, 4-14–4-27
- incorporating dialog boxes or dialog controls into
 - script 4-33
- Information dialog box for, displaying 4-15
- moving
 - with arrow keys 4-19
 - with Information dialog box 4-20
 - with mouse 4-19
- opening dialog box template files
 - in Dialog Editor 4-30
- pasting dialog box controls
 - into Dialog Editor 4-28, 4-29
- pasting dialog boxes into Dialog Editor 4-28, 4-29
- resizing 4-21
 - with Information dialog box 4-21
 - with mouse 4-21
- selecting 4-14
- testing 4-31
 - for basic problems 4-31
 - for operational problems 4-32
 - when there are hidden controls 4-20
- titles of, changing 4-22
- dialog controls, in Dialog Editor *See also* dialog controls, in Dialog Editor
 - adding, in Dialog Editor 4-9–4-10
 - Cancel buttons
 - in Dialog Editor 4-7
 - check boxes
 - in Dialog Editor 4-8
 - combo boxes
 - in Dialog Editor 4-8
 - deleting, in Dialog Editor 4-27
 - drop list boxes
 - in Dialog Editor 4-9
 - duplicating, in Dialog Editor 4-26
 - group boxes
 - in Dialog Editor 4-8
 - list boxes
 - in Dialog Editor 4-8
 - moving, in Dialog Editor 4-20
 - OK buttons
 - in Dialog Editor 4-7
 - option buttons
 - in Dialog Editor 4-7
 - picture buttons
 - in Dialog Editor 4-9
 - picture controls
 - in Dialog Editor 4-9
 - positioning with grid, in Dialog Editor 4-11–4-12
 - push buttons
 - in Dialog Editor 4-7
 - resizing, in Dialog Editor 4-21
 - selecting, in Dialog Editor 4-14
 - text boxes
 - in Dialog Editor 4-8, 4-32
 - text controls
 - in Dialog Editor 4-8
- dialog controls, in Dialog Editor *See also* dialog boxes, in Dialog Editor; dialog controls; Dialog Editor; user dialogs
 - accelerator keys
 - assigning to 4-22–4-23
 - testing 4-32
 - adding to dialog box 4-9–4-10
 - attributes of, adjusting with Information
 - dialog box 4-17, 4-18
 - Cancel buttons 4-7
 - check boxes 4-8
 - combo boxes 4-8
 - creating efficiently 4-12–4-13
 - deleting
 - all controls 4-27
 - one control 4-27
 - drop list boxes 4-9
 - duplicating 4-26
 - group boxes 4-8
 - hidden 4-20
 - Information dialog box for, displaying 4-16
 - labels of, changing 4-22
 - list boxes 4-8
 - moving
 - with arrow keys 4-19
 - with Information dialog box 4-20
 - with mouse 4-19
 - OK buttons 4-7
 - option buttons 4-7
 - grouping 4-13, 4-32
 - pasting controls into Dialog Editor 4-28
 - picture buttons 4-9. *See also* dialog controls, in Dialog Editor, specifying pictures
 - picture controls 4-9. *See also* dialog controls, in Dialog Editor, specifying pictures
 - positioning with grid 4-11–4-12
 - push buttons 4-7
 - resizing 4-21
 - automatically 4-21
 - with Information dialog box 4-21
 - with mouse 4-21
 - selecting 4-14
 - specifying pictures 4-24
 - from file 4-24
 - from picture library 4-24–4-25

- tabbing order of 4-12, 4-29, 4-32
- text boxes 4-8, 4-32
- text controls 4-8
- types of 4-7–4-9
- Dialog Editor 2-3, 4-1, 4-2. *See also* dialog boxes, in
 - Dialog Editor; dialog controls; dialog controls, in Dialog Editor; user dialogs
- application window 4-3
- Controls menu 4-43–4-44
- controls supported by 4-7–4-9
- Dialog Translation Errors dialog box 4-29, 4-30
- Edit menu 4-42–4-43
- exiting from 4-33
- features of 4-2
- File menu 4-41–4-42
- grid, displaying and adjusting 4-11–4-12
- help
 - for current window 4-6
 - on selected topics 4-6
- Help menu 4-45
- Information dialog box 4-15–4-18
 - adjusting control attributes with 4-17, 4-18
 - adjusting dialog box attributes with 4-17
 - displaying
 - for controls 4-16
 - for dialog boxes 4-15
- keyboard shortcuts 4-5
- Pick tool 4-45
- picture libraries 4-25–4-26
 - creating for use in 4-25–4-26
 - modifying for use in 4-26
- toolbar of 4-4
- undoing editing operations 4-27
- dialog procedures
 - actions sent to 4-40
- dialogs, built-in
 - listing of 1-12
- drop list boxes
 - in Dialog Editor 4-9
- duplicating controls, in Dialog Editor 4-26

E

- editing custom dialog boxes 4-1, 4-14–4-27
- Editing Program
 - About 3-1
- Editing Programs 3-1
- environment, controlling
 - list of language elements 1-6
- error messages
 - BasicScript-specific A-5
 - compatible with Visual Basic A-2
 - compiler B-1–B-4
 - runtime A-2–A-4

- errors
 - list of language elements 1-8
- executing scripts, in Program Editor
 - pausing execution of 6-1
 - starting execution of 6-1
 - stopping execution of 6-1
 - tracing execution of 5-2–5-3
- exiting
 - from Dialog Editor 4-33
- exiting from Dialog Editor 4-33

F

- file system
 - list of language elements 1-9
- files
 - list of language elements 1-9
- financial functions
 - list of 1-10
- Functional Overview
 - Program Editor 2-1

G

- grid, in Dialog Editor 4-11–4-12
- group boxes
 - in Dialog Editor 4-8

H

- help
 - in Dialog Editor 4-6, 4-45

I

- Information dialog box, in Dialog Editor 4-15–4-18
- ini files
 - list of language elements 1-11
- inserting text, in Program Editor 3-3
- insertion point, moving, in Program Editor 3-1
 - to specified line 3-2
 - with mouse 3-1–3-2
- InterScript Calls
 - Program Editor 2-3

K

- keyboard shortcuts
 - in Dialog Editor 4-5

L

- line continuation
 - in Program Editor 3-9
- list boxes
 - in Dialog Editor 4-8
- logical operators
 - list of 1-11

M

- math functions
 - list of 1-11
- Menu Functions
 - Program Editor 2-5
- menus, reference for
 - in Dialog Editor 4-34–4-45
- messages, runtime error A-2–A-4
- moving
 - controls, in Dialog Editor 4-20
 - dialog boxes, in Dialog Editor 4-20
- Multiple Threads of Execution
 - Program Editor 2-3

N

- numeric operators
 - list of 1-12

O

- objects
 - list of language elements 1-12
- OK buttons
 - in Dialog Editor 4-7
- option buttons
 - in Dialog Editor 4-7, 4-13, 4-32

P

- parsing
 - list of language elements 1-12
- pasting text
 - in Script Editor 3-7
- picture buttons
 - in Dialog Editor 4-9
- picture controls
 - in Dialog Editor 4-9
- picture libraries, creating or modifying 4-25–4-26
- pictures, specifying, in Dialog Editor 4-24–4-25
- printing
 - list of language elements 1-13
- procedures
 - list of language elements 1-13
 - tracing calls of, in Program Editor 5-4

- Program Editor 3-1
 - Breakpoints 2-2
 - comments, adding to script 3-8
 - dialog box templates, editing for use in 3-13
 - Dialog Editor 2-3
 - Functional Overview 2-1
 - Getting Started 2-3
 - insertion point, moving 3-1
 - to specified line 3-2
 - with mouse 3-1–3-2
 - instruction pointer 5-1
 - moving to another line in subroutine 5-4
 - InterScript Calls 2-3
 - Menu Functions 2-5
 - Multiple Threads of Execution 2-3
 - On Line Help 2-3
 - QuickWatch 2-2
 - scripts
 - checking syntax of 3-12
 - navigating within 3-1–3-2
 - pausing execution of 6-1
 - running 6-1
 - stopping execution of 6-1
 - tracing execution of 5-2–5-3
 - selection highlight 3-4
 - Shortcut Keys 2-12
 - statements, continuing on multiple lines 3-9
 - Step In 2-2
 - Step Over 2-2
 - text
 - copying 3-7
 - cutting 3-7
 - deleting 3-6
 - inserting 3-3
 - pasting 3-7
 - replacing 3-11
 - searching for 3-10
 - selecting 3-4–3-5
 - Toolbars 2-10
 - Trace Window 2-2
 - undoing editing operations 3-7
 - Watch Window 2-2
 - Window Components 2-4
 - Wizards 2-1
 - Program Editor Menus
 - Debug Menu 2-7
 - Edit Menu 2-6
 - File Menu 2-5
 - Help Menu 2-9
 - Run Menu 2-7
 - Tools Menu 2-8
 - View Menu 2-8
 - Windows Menu 2-9

- Program Editor Toolbars
 - Application Toolbar 2-11
 - Standard Toolbar 2-10
 - Tools Toolbar 2-11
- push buttons
 - in Dialog Editor 4-7

Q

- QuickWatch Object Inspector
 - Program Editor 2-2

R

- replacing text, in Program Editor 3-11
- resizing
 - controls, in Dialog Editor 4-21
 - dialog boxes, in Dialog Editor 4-21
- runtime errors A-2–A-4

S

- searching for text, in Program Editor 3-10
- selecting
 - controls, in Dialog Editor 4-14
 - dialog boxes, in Dialog Editor 4-14
- selecting text, in Program Editor 3-4–3-5
- Shortcut Keys
 - Program Editor 2-12
- status bar
 - in Dialog Editor 4-3
- Step In
 - Program Editor 2-2
- Step Over
 - Program Editor 2-2
- string operators
 - list of 1-13
- strings
 - list of language elements 1-13
- syntax, checking, in Program Editor 3-12

T

- testing dialog boxes, in Dialog Editor 4-20, 4-31–4-32
- text boxes
 - in Dialog Editor 4-8, 4-32
- text controls
 - in Dialog Editor 4-8
- toolbar
 - in Dialog Editor 4-3, 4-4
- Toolbars
 - Program Editor 2-10
- Trace Window
 - Program Editor 2-2

U

- undo
 - in Dialog Editor 4-27
 - in Program Editor 3-7
- user dialogs
 - idle processing for 4-40
 - list of language elements 1-14

V

- variables
 - list of language elements 1-15
 - watch, in Program Editor 5-8–5-10
- variants
 - list of language elements 1-15
- Visual Basic, error messages A-2

W

- watch variables, in Program Editor 5-8
 - adding 5-8–5-9
 - deleting 5-10
 - modifying value of 5-10
 - selecting 5-9
- Watch Window
 - Program Editor 2-2
- Window Components
 - Program Editor 2-4
- Wizards
 - Program Editor 2-1