Eloquence Dialog Manual
B.06.32

Edition E1202

Legal Notices

Restricted Rights Legend

Acknowledgments

Printing History

The manual printing date indicates its current edition. The printing date will change when a new edition is printed. Minor changes may be made at reprint without changing the printing date. New editions are complete revisions of the manual.The dates on the title page change only when a new edition or a new update is published.

Manual updates may be issued between editions to correct errors or document product changes. Manuals that are published on the Eloquence website (www.hp-eloquence.com/doc) may be updated more often, please visit this website periodically for the most recent versions. To ensure that you receive the updated or new editions, you should also subscribe to the appropriate product support service.

The software code printed alongside the date indicates the version level of the software product at the time the manual or update was issued. Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one to one correspondence between product updates and manual updates.

First Edition		A.04.00
Second Edition	January 1997	A.06.00
Third Edition	October 1997	A.06.00
Fourth Edition (E1202)	December 2002	B.06.32

Printed in the Federal Republic of Germany.

Printing History

Contents

Contents

1

Eloquence Dialog System

Eloquence provides two Dialog Systems. The semigraphical one, based on “Curses”, which runs on Terminals and on PC Terminalemulators. This one is integrated in Eloquence. The graphical user intrface uses the ISA Dialog Manager runtime library and supports the Windows and Motif platform. This manual covers both user interfaces and so the manual is splited into three parts: - the common parts of both systems - semigraphical specials - graphical specials

Both Dialog Systems consist of two parts each. A runtime and a development system. The runtime system is controlled via the DLG API. It can create and manipulate the dialog by executing commands getting from the API, or creating a dialog by reading a definitions-file. The ASCII DLG runtime system is included in Eloquence, exactly in the “eloqcore” program. The DM runtime system can be a separat program, if running on a client system. The definitionsfile has to be created with the development system. The ASCII DLG development system is simply a text editor. The Dialog Manager provide a graphical editor to create the definitionsfile. This file is also an plain text file, so it can be created and maintained with an text editor, too.

Overview

A dialog consists of objects within a window.

Objects are elements with pre-defined characteristics. The functionality of an object is defined by the class (type) to which it belongs. An object has attributes (variables) which control its functionality within a pre-defined range. Attributes depend on the object class.

Object Path

The objects of a dialog are linked hierarchically, according to the visual hierarchy.

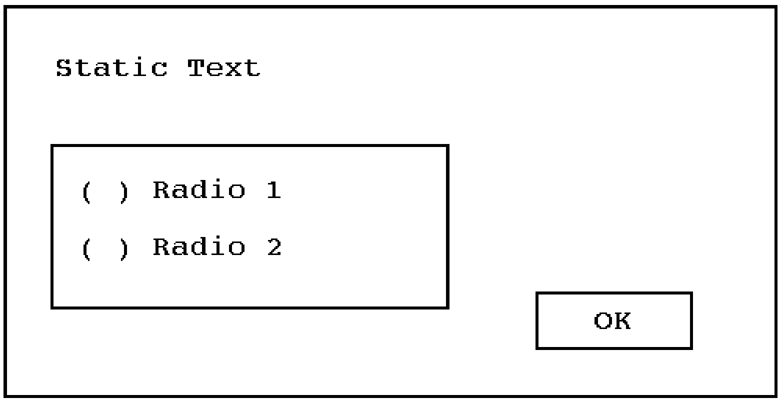
Each object has a name. The name may consist of upper and lower-case letters, numbers, and the “_” character. The name is not case-sensitive.

In order to reference (access) an object you must specify its path.

The object path is always unique.

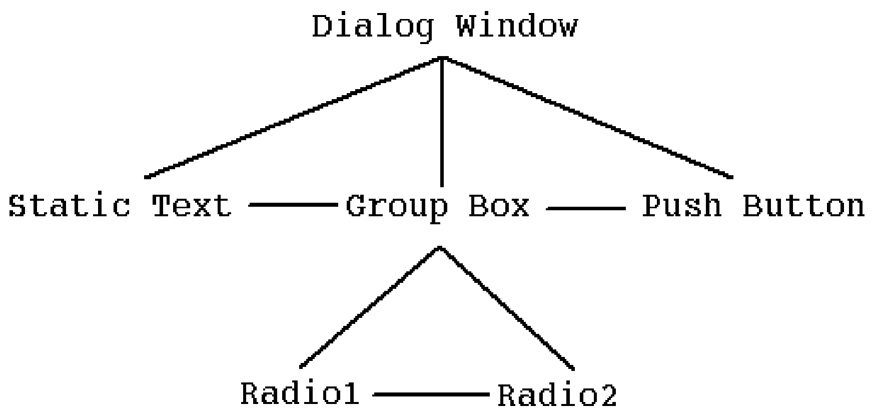
Example:

The following dialog contains a statictext, a groupbox with two radiobuttons, and a pushbutton.



The internal structure reflects the visual hierarchy:

- In the dialog window there are three objects: a statictext, a groupbox and a pushbutton
- In the group box there are two objects: two radio buttons



The object path precisely expresses this hierarchy.

Dialog.Group.Radio1 is thus the path for the radiobutton “Radio1” in the group-box “Group” in the dialog “Dialog”.

Attributes

An object has attributes (variables) which control its functionality within a predefined range. Attributes depend on the object class. Attributes can be set and requested (get).

In order to reference an attribute, its name is appended to the object path separated by a period. Attribute names are not case-sensitive.

e.g. Dialog.Group.Radio1.ACTIVE

references the attribute ACTIVE of the object Radio1, which is in the groupbox “Group” in the dialog named “Dialog”.

Attribute Types

Attributes are of different data types, depending on the type of information:

- String - character string (terminated by \0)
- Integer
- Boolean - 0 (=no, off, False), 1 (=yes, on, True)
- Any - string or integer

There are two groups of attributes.

- Base attributes - These attributes are available to all objects.
- Class-specific attributes - Each class defines additional attributes only available to objects of this class.

Base Attributes

The base attributes are available for all objects.

class Attribute

The class attribute returns the class name of the referenced object. It is set implicitly when the object is created and cannot be modified.

x, y Attribute

x, y define the position of the upper left corner of the object relative to its parent object, except for the root object (dialog) which is relative to the screen.

w, h Attribute

w, h define the object width and height. Note that all objects are clipped by the parent size. w and h must be greater than zero, otherwise the object is invisible.

The fgc, bgc Attributes

This Attribute has different meanings in ASCII as in Dialog Manager.

In ASCII it defines an display enhancement, because here are no colors available.

In Dialog Manager colors can be defined and selected with this two Attributes.

.fgc	Define dialog-object “foreground” enhancements or colors
.bgc	Define dialog-object “background” enhancements or colors

In SSCII DLG both attributes accept a numerical argument, which specifies the enhancement by an additive value:

0	use default enhancement
+2	blinking video
+4	underline video
+8	half-bright video

A value of 16 will disable default enhancements.

For example:

```
DLG SET "Dialog.EditText.fgc",9
```

will cause the addressed EditText to be displayed reverse/half bright video instead of the default underline enhancement.

For example:

```
DLG SET "Dialog.EditText.fgc",Input
```

will cause the addressed EditText to be displayed in the color befined by “Input”.

visible Attribute

The visible attribute defines if an object is visible or not. Note that an object is only visible if its parent object is visible.

sensitive Attribute

The sensitive attribute defines whether the object may receive the keyboard focus. This attribute is ignored for all object classes which do not provide keyboard support.

rule Attribute

The rule attribute is of type integer, and can have any value. The value of the rule attribute affects the dialog handling. The rule attribute serves several purposes, depending on the value and object class:

- If the object class is PushButton and the object is “executed”, the dialog will be suspended, returning the rule value. If the rule value is set to -1, the help subsystem will be called instead.
- If the object class is CheckBox, RadioButton or ListBox and the object is “executed”, and the rule value is non-zero, the dialog will be suspended, returning the rule value. If the rule value is set to -1, the help subsystem will be called instead.
- In all other cases, if the focus will be moved from an object with a non-zero rule value, the dialog will be suspended, returning the rule value.

focus Attribute

The focus attribute will serve a dual purpose. If retrieved it will return non-zero if the object has the keyboard focus. If set, it will force the setting of the keyboard focus to the specified object, its children or the next available object, whatever is available first.

So you may specify focus for parent object and the first child is selected.

focusobj

NOTE: This attribute is not available with Dialog Manager.

It can be read only and returns the Objectpath of the object which has the focus currently.

The kbind Attribute

NOTE: This attribute is not available with Dialog Manager.

It makes it possible to override the default keyboard handling for a specific object by assigning a rule for a key value.

For example:

The following statement assigns a rule value of 103 for a PageDown key (which has a key code of 338):

```
DLG SET "Object.kbind[338]",103
```

NOTE: The Eloquence **KBCODE** statement displays the code for a given key.
It is not recommended to make frequent use of the kbind attribute since this may lead to confusion due to nonstandard keyboard behavior.

The kbind attribute affects the actual object and its child objects. If you define a kbind attribute for a GroupBox, it is also active for all child objects. A different definition in a child object has priority over a parent definition.

The following GET and SET operations on the kbind attribute are defined:

- If a zero is assigned to a kbind attribute or a kbind attribute with index zero, all key assignments are removed. For example:

```
DLG SET "Object.kbind",0
DLG SET "Object.kbind[0]",0
```

- If a zero value is assigned to a kbind index, the specified key assignment will be removed. For example:

```
DLG SET "Object.kbind[338]",0
```

- If a nonzero value is assigned to a kbind index, the specified key will be assigned with the given rule value. For example:

```
DLG SET "Object.kbind[338]",103
```

- The value returned by a kbind attribute or a kbind attribute with an zero index is the number of kbind assignments. For example:

```
DLG GET "Object.kbind",N
DLG GET "Object.kbind[0]",N
```

- The value returned by a kbind attribute with an index N is the Nth key assignment. For example:

```
DLG GET "Object.kbind[1]",N
```

The following example program will demonstrate the various DLG GET/SET operations on the kbind attribute:

```
DLG SET "Object.kbind",0          ! Clear all key assignments
DLG SET "Object.kbind[65]",20     ! ’A’ returns 20
DLG SET "Object.kbind[32]",21     ! space returns 21

DLG GET "Object.kbind",Nkeys      ! Get number of values
DISP "Nkeys=";Nkeys

FOR I=1 TO Nkeys                  ! Get all values
   DLG GET "Object.kbind["&VAL$(I)&"]",K
   DISP "kbind["&VAL$(I)&"]=";K
NEXT I
```

The following keys can not be reassigned, because they are processed internally:

- The ESC key (27)

- The BREAK key (0) and the ^Y key (25)
- The ^L key (12)
- The ^N key (14)
- The Insert key (331)
- The key assigned to the alt attribute.

help Attribute

The help attribute may hold a string used by the help subsystem to identify the section in the helpfile.

udata Attribute

The udata (”user data”) attribute is used to store a value of any data type. If you need to assign your own information to an object (e.g. to validate data entry) you may use the udata attribute.

first, next Attribute

The first, next attributes are not regular attributes like the ones described above. The .first attribute returns the first dialog name when used without an object name. If an object name is provided, it returns the first child object name for the given object.

The .next attribute returns the next dialog name when used without an object name. If an object name is provided, it returns the next object name.

An empty string is returned if requested information is not available (e.g., no child or last object).

alt Attribute

NOTE: This attribute is not available with Dialog Manager.

The alt attribute is only available for the Dialog Object. If it is nonzero, it is the key number of the key to be used as an ALT key in this dialog. The alt attribute must be defined in order to enable keyboard accelerators in the dialog (see below).

With Dialog Manager the “Alt”-Key is used to work with keyboard accelarators.

Keyboard Accelerators

If a '&' character is contained in the label text of a StaticText, CheckBox, RadioButton or PushButton object type, the following character will be displayed underlined and used as a keyboard accelerator.

NOTE: If you want to display a “&” in a StaticText of an ASCII DLG Dialog, you must not define the alt attribute. In a Dialog Manager Dialog you have to define three “&” to display an “&”.

If the alt attribute is enabled in the dialog, pressing the ALT key (In ASCII DLG: specified by the alt attribute) and a keyboard accelerator, causes the focus to change to the first object defining the accelerator key.

If the object type is StaticText, keyboard focus is changed to the next object in Tab order. If the object type is of type CheckBox, RadionButton or PushButton, the object is selected.

If pressing a key which is not accepted by the current object (the object which has the focus), it will be used as an accelerator.

For example, pressing an alphanumeric key while the current object is of type PushButton will cause an accelerator lookup.

Example for an ASCII DLG dialogfile:

```
Dialog Sample {
   ...
   .alt = 266    # Alt key is F2

   StaticText Label1 {
      ...
      .text = "Label &1"
   }
   EditText Edit {
      ...
   }
   PushButton OK {
      ...
      .text = "&OK"
   }
}
```

Pressing the keys F2 and 1 will change the focus to the EditText “Edit”. Pressing the keys F2 and O will trigger the PushButton “OK”.

Object Classes

On the following pages, the different object classes of the ASCII DLG are defined. All additional Objects of the Dialog Manager are defined in the Dialog MAnager Manual.

The definition contains information on whether the objects of this object class can have the cursor focus and whether it can have children or not. Then all attributes which are available for this object class are described, together with their data types, their default values and whether the attribute values can be set and/or requested (get).

Accessing Dialog Manager Objects

Eloquence only knows about a subset of Dialog Manager objects and attributes as necessary to translate Eloquence DLG to Dialog Manager.

In order to enable access to Dialog Manager objects and attributes not available in Eloquence DLG there is a “bypass option” implemented in Eloquence.

If you use an exclamation mark (’`!`’) instead of a dot (’`.`’) as a separator between object path and attribute, no mapping will be performed. Instead, the *native* Dialog Manager objects and attributes are accessed.

Two examples of *native* Dialog Manager objects are *menubar* and *menuitem*. An example of a *native* Dialog Manager attribute is the *format* attribute of *edittext* objects:

```
   DLG SET "MyWindow.Edit!format","%8'3,02sf/1"
```

This will set the field format of the *edittext* object named *Edit*.

Dialog Object

The root object of a dialog must always be of class Dialog. All other objects have to be generated in a dialog. The dialog is an optical frame for other objects. The coordinates are interpreted as relative to the screen.

Class name:	**Dialog**
Focus :	No
Children :	Yes

Attributes:

Name	Type	G/S	Default value
.class	Strg	G	Dialog
.x	Int	G/S	0
.y	Int	G/S	0
.w	Int	G/S	0
.h	Int	G/S	0
.visible	Bool	G/S	True
.sensitive	Bool	G/S	False
.rule	Int	G/S	0
.focus	Bool	G/S	False
.help	Strg	G/S	
.udata	Any	G/S	
.title	Strg	G/S	
.border	Int	G/S	3
.f1	Int	G/S	-1
.f2	Int	G/S	0
.f3	Int	G/S	0
.f4	Int	G/S	0
.f5	Int	G/S	0
.f6	Int	G/S	0
.f7	Int	G/S	0
.f8	Int	G/S	0
.cr	Int	G/S	0

The .x and .y attributes define the position of the upper left corner of the object. These values are relative to the screen for the Dialog object class.

The .border attribute defines the type of the border. The following values are supported:

0	no border
1	thin border
2	thick border
3	shadow border
4	reverse shadow border

The .f1…f8 attributes assign values to the function keys F1 .. F8. If a non-zero value is assigned to a function key, and this function key is pressed, the dialog is suspended, returning the appropriate f1 .. f8 attribute value and the object path where the focus was set. If the assigned value is -1 and the function key is pressed, the help subsystem is started.

If not specified, function keys are ignored except for f1, which calls the help subsystem.

The .cr attribute assigns a value to the Return key. If the value is non-zero and the Return key is pressed, the dialog is suspended, returning the appropriate cr attribute value and the object path where the focus was set. The default behaviour of the Return key is to move the focus to the next object.

Groupbox Object

A groupbox is used to group objects together. If you move the groupbox, all objects in this group are moved.

Class name:	**GroupBox**
Focus :	No
Children :	Yes

Attributes:

Name	Type	G/S	Default value
.class	Strg	G	GroupBox
.x	Int	G/S	0
.y	Int	G/S	0

Name	Type	G/S	Default value
.w	Int	G/S	0
.h	Int	G/S	0
.visible	Bool	G/S	True
.sensitive	Bool	G/S	False
.rule	Int	G/S	0
.focus	Bool	G/S	False
.help	Strg	G/S	
.udata	Any	G/S	
.title	Strg	G/S	
.border	Int	G/S	1

The .title attribute provides a text, which is placed in the top line of the groupbox object.

If a GroupBox is not sensitive, all of its child-objects are insensitive, too (they can not get the keyboard focus). A GroupBox is sensitive by default.

The .border attribute defines the type of the border. The following values are supported:

0	no border
1	thin border
2	thick border
3	shadow border
4	reverse shadow border

StaticText Object

A statictext is used to place text in the dialog.

Class name:	**StaticText**
Focus :	No
Children :	No

Attributes:

Name	Type	G/S	Default value
.class	Strg	G	StaticText
.x	Int	G/S	0
.y	Int	G/S	0
.w	Int	G/S	length of text
.h	Int	G/S	1
.visible	Bool	G/S	True
.sensitive	Bool	G/S	False
.rule	Int	G/S	0
.focus	Bool	G/S	False
.help	Strg	G/S	
.udata	Any	G/S	
.text	Strg	G/S	

The .text attribute contains the displayed string.

If the .w attribute is not specified, it will default to the text length set initially.

PushButton Object

A pushbutton is used for making decisions. Dialogs are generally suspended (terminated) and the control returned to the application by pressing a pushbutton.

Class name :	**PushButton**
Focus :	Yes
Children :	No

Keyboard usage:

Use Spacebar or Return to "execute" a pushbutton.

Attributes:

Name	Type	G/S	Default value
.class	Strg	G	PushButton
.x	Int	G/S	0
.y	Int	G/S	0
.w	Int	G/S	length of text
.h	Int	G/S	1
.visible	Bool	G/S	True
.sensitive	Bool	G/S	True
.rule	Int	G/S	0
.focus	Bool	G/S	False
.help	Strg	G/S	
.udata	Any	G/S	
.text	Strg	G/S	
.border	Int	G/S	1

The .text attribute contains the string.

If .w attribute is not specified, it will default to the text length set initially.

If the .h attribute is equal to or greater than 3, the pushbutton is framed.

If the .rule attribute is set to -1, the pushbutton acts as a HELP button. The help subsystem is called with the text defined in the .help attribute as indicator to the helpfile.

CheckBox Object

A checkbox is an object with the activation states “on” or “off”.

Class name:	**CheckBox**
Focus :	Yes
Children :	No

If the object is active, an ’X’ will be displayed in front of its text, when working with ASCII DLG.

Keyboard usage:

Use Spacebar to switch the state.

Attributes:

Name	Type	G/S	Default value
.class	Strg	G	CheckBox
.x	Int	G/S	0
.y	Int	G/S	0
.w	Int	G/S	length of text
.h	Int	G/S	1
.visible	Bool	G/S	True
.sensitive	Bool	G/S	True
.rule	Int	G/S	0
.focus	Bool	G/S	False
.help	Strg	G/S	
.udata	Any	G/S	
.text	Strg	G/S	
.active	Bool	G/S	False

The .text attribute contains the string.

If .w attribute is not specified, it will default to the text length set initially.

The .active attribute contains the state of the checkbox.

If the .rule attribute is non-zero and the object state has been changed, the dialog is suspended and the rule value will be returned. If the .rule attribute is -1, the help subsystem will be called instead.

RadioButton Object

A radiobutton is an object with the activation states “on” or “off”. In contrast to the checkbox, a radiobutton is used to make a single selection from a group of options. If the object is active and the ASCII DLG system is used, a ‘*’ will be displayed.

Class name:	**Radiobutton**
Focus :	Yes
Children :	No

Keyboard usage:

The Spacebar is used to set the current radiobutton. All other radiobuttons in the same hierarchy level are switched off.

Attributes:

Name	Type	G/S	Default value
.class	Strg	G	RadioButton
.x	Int	G/S	0
.y	Int	G/S	0
.w	Int	G/S	length of text
.h	Int	G/S	1
.visible	Bool	G/S	True
.sensitive	Bool	G/S	True
.rule	Int	G/S	0
.focus	Bool	G/S	False
.help	Strg	G/S	
.udata	Any	G/S	
.text	Strg	G/S	
.active	Bool	G/S	False

The .text attribute contains the string.

If .w attribute is not specified, it will default to the text length set initially.

The .active attribute contains the state of the radiobutton. If a radiobutton becomes active, all other radiobuttons in the same hierarchy level will become inactive. If you have more than one group of radiobuttons, you have to put each group into a groupbox.

If the .rule attribute is non-zero and the object state has been changed, the dialog is suspended and the rule value will be returned. If the .rule attribute is -1, the help sub-system will be called instead.

EditText Object

An edittext is a text which can be modified interactively. It can be single-line or multi-line. It can be scrolled horizontally, and vertically if multi-line.

Class name:	**EditText**
Focus :	Yes
Children :	No

Keyboard usage:

Home, Shift + Home, Cursor Left, Cursor Right can be used for editing within a single-line edittext. ,

Additionally, in a multi-line edittext or a listbox object, the following keys can be used: Page Up, Page Down, Cursor Up, Cursor Down.

EditText is set to overwrite mode by default. You may toggle overwrite/insert mode using the INSERT key. The cursor form will change to indicate insert/over-write mode.

Attributes:

Name	Type	G/S	Default value
.class	Strg	G	EditText
.x	Int	G/S	0
.y	Int	G/S	0
.w	Int	G/S	0
.h	Int	G/S	1
.visible	Bool	G/S	True
.sensitive	Bool	G/S	True
.rule	Int	G/S	0
.focus	Bool	G/S	False
.help	Strg	G/S	
.udata	Any	G/S	
.length	Int	G	

Name	Type	G/S	Default value
.content	Strg	G/S	
.editable	Bool	G/S	True
.multiline	Bool	G/S	False
.border	Int	G/S	1
.title	Strg	G/S	
.hsb	Bool	G/S	True
.vsb	Bool	G/S	True
.maxchars	Int	G/S	
.maxlines	Int	G/S	
.vheight	Int	G	
.vwidth	Int	G	
.cx	Int	G/S	
.cy	Int	G/S	
.line	Strg	G/S	
.file	Strg	S	
.clear	Any	S	
.add	Strg	S	
.topitem	Int	S	
.writefile	Strg	S	
.ins	Strg	S	
.delln	Int	S	

The .editable attribute specifies if the object may be edited. You can change this attribute at any time.

The EditText object class distinguishes two different modes.

- single-line mode

- multi-line mode

This is defined by the .multiline attribute.

Single-Line

The following attributes have no effect in single-line mode.

- border and title
- hsb and vsb (horizontal and vertical scrollbar)
- maxlines

The EditText object class provides a single line which can be scrolled horizontally. With ASCII DLG a ’<’ or ’>’ mark indicates that the text is scrolled to the left or right side out of view.

Multi-Line

If the .multiline attribute is non-zero, a window with horizontal/vertical scrollbars is displayed.

The .maxlines and .maxchars attributes specifiy how many characters or lines are accepted.

The .hsb and .vsb attributes indicate whether a horizontal or vertical scrollbar should be provided to indicate the position and the relation of the visible part of the text to the whole.

Without an index, the .line attribute will get or set the line as indicated by the cy attribute. If an index is specified, the specified line is get or set. The first line is specified by 1.

The .title attribute provides a text which is placed in the top line of the object.

NOTE: The title attribute is not available with Dialog Manager

The .border attribute specifies whether a border should be placed around the object. The following values are supported:

0	no border
1	thin border
2	thick border
3	shadow border
4	reverse shadow border

The .content attribute contains the text. Lines are separated by a (newline, lf) character. They can also be set or requested using a string array.

The .length attribute returns the number of characters of the text.

The .vwidth and .vheight attributes return the number of lines and the length of the widest line of the text.

The .cx and .cy attributes define the cursor position.

The .line attribute contains the text of the line where the cursor is positioned.

For example:

```
DLG SET "Edit.line[12]","Text"
```

The .clear, .file and .add attributes are not regular attributes. They perform a specific operation on the object:

- The .clear attribute accepts any kind of value and clears the whole text.
- The .add attribute adds a line of text to the object.
- The .file attribute reads the specified file into the object.

NOTE: When the Dialog Manager runtime is running on a remote machine, the file has to be accessible on that machine.

The writefile attribute is of type string. It specifies a file name, where the contents of the EditText are written to.

Using this attribute, you can write your own Text Editor very simply:

```
DLG SET "dialog.edit.file","/tmp/text"           ! read file
DLG DO "dialog",R                                ! handle dialog
DLG SET "dialog.edit.writefile","/tmp/text.new"  ! write file
```

NOTE: When the Dialog Manager runtime is running on a remote machine, the file will be created on that machine.

If the .rule attribute is non-zero, and if the focus has changed, the dialog is suspended and the rule value is returned.

The .topitem attribute indicates the first visible line in an EditText. When set, the EditText is scrolled appropriately.

NOTE: The topitem attribute is not available with Dialog Manager

The .ins attribute is of type string. It inserts the given string at the current cursor position.

The .delln attribute is of type integer. It deletes the given number of lines starting at the cursor line.

ListBox Object

A listbox contains a dynamic list of text elements from which one text element can be selected.

Class name: **ListBox**

Focus : Yes

Children : No

Keyboard usage:

The Spacebar is used to select/deselect a listbox entry which is then displayed reversed if selected. You may use the cursor keys to navigate inside the listbox.

Attributes:

Name	Type	G/S	Default value
.class	Strg	G	ListBox
.x	Int	G/S	0
.y	Int	G/S	0
.w	Int	G/S	0
.h	Int	G/S	0
.visible	Bool	G/S	True
.sensitive	Bool	G/S	True
.rule	Int	G/S	0
.focus	Bool	G/S	False
.help	Strg	G/S	
.udata	Any	G/S	
.length	Int	G	
.content	Strg	G/S	
.multiline	Bool	G/S	True

Name	Type	G/S	Default value
.border	Int	G/S	1
.title	Strg	G/S	
.hsb	Bool	G/S	True
.vsb	Bool	G/S	True
.vheight	Int	G	
.vwidth	Int	G	
.cy	Int	G/S	
.line	Strg	G/S	
.activeline	Int	G/S	
.file	Strg	S	
.clear	Any	S	
.add	Strg	S	
.topitem	Int	S	
.ins	Strg	S	
.delln	Int	S	

The ListBox object class allows you to display and select a single value.

The .activeline attribute contains the number of the selected line. A zero means that no line has been selected.

If the .rule attribute is non-zero and the focus attribute has been changed, the dialog is suspended and the rule value is returned. If the rule value is set to -1, the help subsystem will be called instead.

The .topitem attribute indicates the first visible line in a ListBox.

When set, the ListBox is scrolled appropriately.

NOTE: This attribute is not available with Dialog Manager.

The ins attribute is of type string. It inserts the given string before the current line.

NOTE: ListBox always inserts complete lines.

The delln attribute is of type integer. It deletes the given number of lines starting at the cursor line.

The ListBox object type supports an additional index specified for the line attribute.

Without an index, the line attribute will get or set the line as indicated by the cy attribute. If an index is specified, the specified line is got or set. The first line is specified by 1.

For example:

```
DLG SET "List.line[12]","Text"
```

NOTE: This attribute is not available with Dialog Manager

All other attributes are the same as for the EditText object class.

HelpText Object

The helptext object class is used to realize the online help system. It is similar to the EditText object class, but has some special properties.

NOTE: This object is not available with Dialog Manager.

Class name: **HelpText**

Focus : Yes

Children : No

Attributes:

Name	Type	G/S	Default value
.class	Strg	G	Helptext
.x	Int	G/S	0
.y	Int	G/S	0
.w	Int	G/S	0
.h	Int	G/S	0
.visible	Bool	G/S	True
.sensitive	Bool	G/S	True
.rule	Int	G/S	0
.focus	Bool	G/S	False
.help	Strg	G/S	
.udata	Any	G/S	
.border	Int	G/S	1
.title	Strg	G/S	
.hsb	Bool	G/S	True
.vsb	Bool	G/S	True
.file	Strg	G/S	
.topic	Strg	G/S	

Name	Type	G/S	Default value
.forw	Strg	G	
.backw	Strg	G	
.link	Strg	G	

The .file attribute specifies the help file to use.

The .topic attribute specifies the current help tag.

The .forw and .backw attributes specify the next/previous help tags.

The .link attribute specifies the currently selected text reference.

This object will be maintained automatically by the DLG HELP statement and the help subsystem.

See Discussion on help below.

API (Application Programming Interface)

This section describes the Eloquence statements which allow access to dialog window functionality.

All statements begin with DLG for DIALOG. A return variable can be specified for each statement. If a return variable is present, a run-time error specific to the dialog system does not lead to an Eloquence run-time error. Instead, the return variable will contain the corresponding error number.

DLG STOP

Syntax:

DLG STOP

DLG STOP has the same effect on the dialog system as the STOP statement on the Eloquence program execution. All windows are closed and all dialogs deleted.

DLG STOP is executed automatically when the Eloquence program terminates.

DLG LOAD

Syntax:

DLG LOAD “*File Name*” [;Rv]

DLG LOAD reads the corresponding Dialog Resource file and generates the dialogs defined in it. For the syntax specifications of Dialog Reaource files see chapter , Dialog Resource File,

The DLG LOAD statement checks the syntax of the dialog file and reports a self explaining error message to the screen, if it is executed interactively.

For example:

```
DLG LOAD "dialog.dlg"
LINE #5: DLG NEW "Test.select","Lisrbox" faile
```

DLG NEW

Syntax:

DLG NEW “*ObjectPath*”,”*ClassName*” [;Rv]

The DLG NEW statement creates an dialog object dynamically.

You have to specify the object path and an object type.

It’s also possible to specify an object path instead of the object type. The specified object and all it's children are copied to the given target path.

With this functionality you are able to use *Models* in your dialogs. For more information about Models in ASCII DLG see chapter , Using Models, more information concerning Dialog Manager Models, see the Dialog Manager Manual.

DLG DEL

Syntax:

DLG DEL “*ObjectPath*” [;Rv]

DLG DEL deletes the specified object and all its children.

DLG SET

Syntax:

DLG SET “*ObjectPath.Attribute*”,*ValueSpec* [;Rv]

DLG SET modifies the attributes of objects. Valid arguments are numeric values, strings and string arrays, depending on the attribute type. To set the content attribute of objects of class Edittext and Listbox use either a string variable or a string array as an argument. If you use a string variable as an argument, use the newline character (CHR$(10)) as a line separator. If you use a string array as an argument, place each element on a different line.

DLG GET

Syntax:

DLG GET “*ObjectPath.Attribute*”,*Variable* [;Rv]

DLG GET is used to get the attributes values of objects. Valid arguments are numeric variables, string variables and string arrays, depending on attribute type.

To get the contents of objects of class Edittext and Listbox use either a string variable or a string array as an argument. If you use a string variable as an argument, the newline character (CHR$(10)) is used as a line separator. If you use a string array as an argument, each element is placed on a different line.

DLG DRAW

Syntax:

DLG DRAW “*ObjectPath*” [;Rv]

Setting attributes will not affect the display until DLG DRAW or DLG DO are executed. DLG DRAW refeshes the dialog and displayes it, if it is not currently visible.

To make the dialog invisible again, set the .visible attribute to false.

DLG DO

Syntax:

DLG DO “*Dialog.Path*” [[,*ExitRule*] [,*ExitObject*]] [;Rv]

DLG DO starts the dialog handling. If the focus for an object of the dialog has not previously been set, either the last active object will remain active, or the focus is set to the first available object.

NOTE: With Dialog Manager it is not possible to set the focus on an invisible dialog.

If the variables *Exit_Rule* (numeric) and *Exit_Object$* (string) are used, they are set to the object path and to the .rule value of the object, which had the focus when the dialog terminated. If the dialog was terminated by pressing a function key, the appropriate value will be returned.

A dialog terminates when either a pushbutton is pressed or when an object is executed and the rule value of this object is neither zero nor -1.

DLG HELP

Syntax:

DLG HELP “*HelpTag*” [;Rv]

DLG HELP activates the online help subsystem explicitly from inside your application using the help tag as the keyword to search in the help text file. Normally the help subsystem is called from within the dialog.

If the keyword does not exist, DLG HELP returns without an error. If *Rv* was specified, it contains the error number 663.

For more information on using Help see chapter , Help Subsystem,

2

Eloquence ASCII Windows

Dialog Resource File

A Dialog Resource file can be loaded using DLG LOAD. This is a normal HP-UX text file in a simple format:

```
Class ObjectName {
   .Attribute = value
   .Attribute = ”String”

   Class ObjectName {
      .Attribute = value
      ...
   }

   ...
}

...
```

Class defines the object class.

ObjectName defines the name of the object.

.*Attribute* defines the characteristic of the object.

After the file has been analyzed, the objects are defined internally with DLG NEW and DLG SET.

Example

```
dialog Dialog1 {
   .x = 1
   .y = 1
   .w = 50
   .h = 12
   .title = “ Dialog1 “
   .f4 = 1

   groupbox box1 {
      .x = 2
      .y = 2
      .w = 20
      .h = 5
      .title = “ box1 “

      radiobutton r1 {
         .x = 1
         .y = 1
         .text = “Radio 1”
      }

      radiobutton r2 {
         .x = 1
         .y = 2
         .text = “Radio 2”
      }
   }

   groupbox box2 {
      .x = 25
      .y = 2
      .w = 20
      .h = 5
      .title = “ box2 “

      radiobutton r1 {
         .x = 1
         .y = 1
         .text = “Radio 1”
      }

      radiobutton r2 {
         .x = 1
         .y = 2
         .text = “Radio 2”
      }
   }

   pushbutton ok {
      .x = 2
      .y = 10
      .text = “ DONE “
      .rule = 1
   }
}
```

The include directive

The include directive makes it possible to have common definitions among dialog files (including context specific definitions).

```
   include "file"
```

This will include the given file. If a relative path is specified, the include file is loaded relative to the path, the current dialog file is loaded from.

The file name may also contain environment variables. Environment variables are expanded in order to locate the file.

For example:

```
   include "$HOME/sample.inc"
```

will include the file sample.inc from the home directory.

```
   include "sample.inc"
```

will include the file sample.inc from the same directory as the dialog file.

If the first character is a question mark ('**?**'), DLG LOAD will not fail if the include file is not present.

For example:

```
include "?$HOME/sample.inc"
```

will try to include the file sample.inc from the home directory.

Include file nesting is limited to two levels.

The define directive

The define directive makes it possible to use Variables in the dialog file.

```
Name = value
```

This will associate Name with value. This name may subsequently be used to reference the associated value.

If *Name* has already been defined, it will be updated with the current value.

For example:

```
# border types
Bd_None    = 0
Bd_Thin    = 1
Bd_Thick   = 2
Bd_shadow  = 3
Bd_rshadow = 4

ALT_KEY    = 266
TITLE      = "Sample string"

Dialog sample {
   ...
   .border = Bd_Thick
   .alt = ALT_KEY
   .title = TITLE
   ...
}
```

Using Models

You are able to use *Models* in your dialogs. The dialog description file contains the *Model* and *Dialog* description, as two dialogs.

For example:

```
Dialog Model {
   ...

   PushButton OK {
      .text = "&OK"
      .rule = 1
   }
   PushButton HELP {
      .text = "&HELP"
      .rule = -1
   }
}

Dialog Sample {
   ...

   Model.OK Ok {
      .x = 10
      .y = 10
   }
   Model.HELP Help {
      .x = 20
      .y = 10
   }
}
```

The PushButtons “OK” and “Help” will be created in the dialog Sample.

All attributes are derived as specified in the dialog Model.

It’s also possible to dynamically create objects from within Eloquence:

```
DLG NEW "Sample.OK","Model.OK"
```

This creates the object OK in the dialog sample as specified by the object Model.OK.

Keyboard Usage

As long as an object does not use the following keys for other purposes, they are used as follows:

- F1 - call Help system.
- Tab, Return and Cursor Down - switch to next object.
- Backtab (Shift + Tab) and Cursor Up - switch to previous object.
- Spacebar - activate/deactivate a radiobutton, a checkbox and a listbox entry.
- Spacebar or Return - activate a pushbutton.
- Home, Shift + Home, Cursor LEFT and Cursor RIGHT can be used for editing within a single-line edittext. All characters are inserted.
- Additionally, in a multi-line edittext or a listbox object, the following keys can be used: Page Up, Page Down, Cursor Up, Cursor Right. With Return you can either insert a line or move it to the beginning of the next line.

Help Subsystem

The Eloquence dialog subsystem supports online help.

Each time a function key with an associated .f1…f8 attribute value of -1, or if not defined otherwise, the f1 function key or an object with a rule value of -1 is executed, the help subsystem will be activated.

Each object supports an optional help attribute. If the help attribute of the current object is defined, its value will be used as the help tag (lookup value) in the help file. If no help attribute is defined, the help tag of the object's parent is used, or, if that is not defined, the help tag of the object's parent's parent, and so on.

So if the help subsystem is activated, it will first check for a context specific help tag, and then up the object hierarchy for at least a dialog-specific one. If no help tag can be located, you will hear a beep.

The layout of the help dialog must be specified by the application programmer. So you are free to define whatever you like. Only a few guidelines must be followed. A template file is explained below.

After loading the Help dialog you have to specify the helpfile name.

A help window consists of a box and several pushbuttons.

Button	Results in
CLOSE	Close help dialog
RETURN	Return to previous help tag
	Continue help dialog with the previous help tag
	Continue help dialog with the next help tag
HELP	Show help on the help window

The buttons RETURN, <<, and >> are invisible if not applicable. The help window contains the help text, and if defined, some text references which are displayed underlined.

If you move the cursor to a text reference, it becomes selected (reverse display). Pressing Return continues the help dialog with the associated help tag.

Using Spacebar and Backspace you could move to the next/previous embedded text reference.

Helpfile Format

The helpfile must contain all the help tags and text references. It’s an HP-UX text file and MUST begin in the first column with the following syntax:

; anything with a semicolon in the 1st column is a comment	
<helptag>	(names the help window)
[<<]<helptag>	(names the previous window in chain)
[>>]<helptag>	(names the next window in chain)
Help text follows until next helptag occurs	
Hypertext reference	[->keyword]<helptag> embedded in text
<helptag>	(names another window)
<end>	

Notes:

1. A text reference, when selected, causes the associated help window named by the helptag to become the active help window.

2. The [->] and helptag marks are not displayed.

Example:

```
; This is a comment line
<Dialog>
This is a help Text for Help tag DIALOG.
There are 2 associated Text references:

    [->Field 1]<Ref1> chains to text Ref1
    [->Field 2]<Ref2> chains to text Ref2
<Ref1>
[>>]<Ref2>

Sample Help Field #1
<Ref2>
[<<]<Ref1>

Sample Help Field #2
<HelpHelp>
This is Help text for Help.
<end>
```

You can use the following program to display the help.

You can use the following program to display the help.

```
10 DLG LOAD “help.dlg”
20 DLG SET “Help.Text.File”,”/users/eloq/dlg/demo”
30 DLG HELP “Dialog”
40 STOP
```

Sample Help Dialog

The help dialog consists of some required and some optional objects.

Required:

- Root object type must be of class Dialog.
- Help dialog must be named with “Help”.
- Inside the root dialog must be a HelpText object named “text”.
- You must specify two pushbuttons named: Close and Return.

Optional:

- pushbutton named “Next”
- push button named “Prev”

To support Help for help dialog:

Set the .f1 attribute to 5 and the .help attribute of your help dialog to the desired help tag e.g., “HelponHelp”. Create a pushbutton named Help with a .rule attribute value of 5.

```
# help.dlg
#
# Default help dialog
#
# This dialog is intended to be used as a template for your own
# help dialog.

dialog Help {
  .x = 1
  .y = 1
  .w = 70
  .h = 20
  .title = “ Help Title “

# to provide help for help
  .f1 = 5
  .help = “HelpHelp”

  helptext text {
    .x = 2
    .y = 2
    .w = 66
    .h = 15
# uncomment if you like it
#     .title = “ Help Dialog “
    .border = 0
  }

  pushbutton Close {
    .x = 2
    .y = 18
    .w = 8
    .text = “ CLOSE “
  }
  pushbutton Return {
    .x = 12
    .y = 18
    .w = 8
    .text = “RETURN “
  }

# PushButton Prev and Next are optional.
  pushbutton Prev {
    .x = 30
    .y = 18
    .w = 8
    .text = “  <<   “
  }
  pushbutton Next {
    .x = 40
    .y = 18
    .w = 8
    .text = “  >>   “
  }

# Help button is optional

  pushbutton Help {
    .x = 60
    .y = 18
    .w = 8
    .text = “ HELP  “
    .rule = 5
  }
}
```

FRM2DLG conversion utility

FRM2DLG converts Eloquence FORM files into Eloquence dialog files. It will create a dialog-file on the MSI volume with the same name as the FORM file. After the conversion process has finished, the dialog-file will be displayed on the screen. Press the function key F8 to terminate.

It’s located in the **`/opt/eloquence/share/contrib/`** directory.

As its location in the contrib directory indicates, FRM2DLG has been provided as a template, that you may customize to fulfill your specific needs.

NOTE: This program is not supported and provided only as an example.

Error Messages

650	General dialog failure
651	Unable to open file
652	Syntax error in dialog description
653	No space left in dialog table or MAX_DEPTH exceeded
654	Memory allocation failed in dialog
655	Duplicate dialog or object name
656	Illegal or bad object path
657	Bad or improper attribute
658	Parent object does not agree to child object
659	Bad or improper value type
660	Invalid value
661	Can’t set focus
662	No helpfile defined
663	No such help tag (informational only)

3

Eloquence Graphical User Interface

The graphical user interface is implemented through (platform dependent) dialog drivers. If you activate a dialog driver, all Eloquence DLG statements are no longer executed by Eloquence, but passed to the specified driver.

The driver will map the Eloquence DLG statements to ISA Dialog Manager intrinsic calls which will handle your display.

Overview

You may simply activate a driver using one of the following DLG SET statements:

```
DLG SET ".driver","motif"
DLG SET ".driver","alpha"
```

The first example above will redirect all DLG statements to the motif driver, while the second example will redirect all DLG statements to the alpha driver.

```
DLG SET ".driver","@client"
```

The example above will redirect all DLG statements to the network driver running on the system named *client*.

NOTE: Using the network driver requires the **RUNSRV** utility to be running on the *client* system.

NOTE: Although it's possible to use Eloquence dialog files (they are converted internally at runtime) it's strongly recommended to convert them to Dialog Manager format due to performance considerations.

Once converted, you're able to change the layout with the Dialog Manager graphical editor.

The Dialog is converted using the **CVDLG** utility:

```
cvdlg -driver motif sample.idm sample.dlg
```

This will convert the Eloquence dialog file *sample*.dlg to Dialog Manager dialog file *sample*.idm using the motif driver.

It's also possible to compile a Dialog Manager dialog file. This has some additional performance advantages during dialog file load time. Compilation is done through the Dialog Manager idm utility.

For Example:

```
idm +writebin sample.idc sample.idm
```

This will compile the Dialog Manager dialog file *sample*.idm into a file *sample*.idc.

Compiled Dialog Manager files are platform dependent.

NOTE: The idm utility program is *not* included with Eloquence. You must purchase the Dialog Manager product to be able to compile dialog files.

The DLG LOAD is able to automatically select the appropriate dialog file if a dialog driver has been activated. If you specify a file extension of “.dlg”, DLG LOAD will look for the following files (in this order):

.idc	compiled Dialog Manager file
.idm	Dialog Manager file
.dlg	Eloquence dialog file

The file extensions may be customized in the eloq.ini configuration file.

For Example:

```
DLG LOAD "sample.dlg"
```

will try to load “sample.idc” first, then “sample.idm” and at last “sample.dlg”.

NOTE: If we talk in the following documentation about dialog server we talk about software that runs on the system on which the user interface should be displayed. The dialog client is the Eloquence application, because it sends requests to the dialog server to handle the user interface.

NOTE: If we talk about systems, the server system is the system on which the Eloquence application runs, while the client is the system on which the user interface is displayed.

Eloquence Dialog Drivers

This document describes the extensions to Eloquence, if a dialog driver has been activated.

Eloquence supports the following attributes:

.driver	SET/GET dialog driver
.async	SET/GET driver communication mode

Starting a driver

To start a driver from within Eloquence, you issue the following command:

```
DLG SET ".driver","driver_spec [ini_section [arguments]]"
```

driver_spec:	*motif*	start the motif driver
	alpha	start the alpha driver
	@client	start the network driver on the system named *client*

ini_section: Optional name of a *user-defined section* in the eloq.ini file where the defaults can be overridden.

arguments: Additional arguments can optionally be specified here and will be passed-through to the Dialog Manager.

On driver start-up, the following tasks are performed:

1. The driver sets up the *Dialog Manager argument list* from the **`arguments`** specified in the **`DLG SET ".driver"`** statement, if any. The driver then reads the configuration items from the eloq.ini file.
2. If **`ini_section`** is specified in the **`DLG SET ".driver"`** statement, the driver reads additional configuration items from this *user-defined section* in the eloq.ini file. The types of entries distinguish between the different drivers, so the examples of user defined section are placed in the driver references (see chapter , Driver Reference,)
3. If **`ini_section`** is specified in the **`DLG SET ".driver"`** statement, the driver searches this *user-defined section* in the eloq.ini file for an item named *Arguments*. If this item exists, its value is appended to the *Dialog Manager argument list*.

   Example:

   ```
   [debug]
   Arguments = -IDMtracefile /tmp/idmtrace
   ```

This enables an additional Dialog Manager argument which creates a trace file for debugging purposes. In order to activate this item, the name of this section must be specified in the **`DLG SET ".driver"`** statement, e.g.:

```
DLG SET ".driver","motif debug"
```

4 Finally, the composed *Dialog Manager argument list* is passed to the Dialog Manager runtime system start-up function.

NOTE: For a list of valid command line arguments, please refer to the *ISA Dialog Manager* documentation.

NOTE: For details about *configuration items* and *user-defined sections* please refer to chapter *Configuring Eloquence*, sections *Customize the ELOQ.INI File on the HP-UX System* and *Customize the ELOQ.INI File on the PC Platform*.

Setting (or re-setting) driver will result in an implied **`DLG STOP`**, this statement will reset the dialog driver.

For example:

```
DLG SET ".driver","motif"
DLG SET ".driver","alpha"
```

The first example above will redirect all DLG statements to the motif driver, while the second example will redirect all DLG statements to the alpha driver.

```
DLG SET ".driver","@client"
```

The example above will redirect all DLG statements to the network driver running on the system named *client*.

NOTE: For a more specific description of the drivers, please refer to the sections *Motif Driver, Alpha Driver* and *Network Driver* in this chapter.

NOTE: Using the network driver requires the **`RUNSRV`** utility to be running on the *client* system.

Driver ASYNC mode

The dialog client and server processes may communicate in two ways:

Synchronous	Each request by the client results in a server response. Processes have to wait on each other.
Asynchronous	Only specific requests result in a server response. Processes run concurrently.

Given the fact that server operations are normally successfully completed, a lot of communication overhead is generated. Using ASYNC mode, this overhead is reduced dramatically, resulting in much better response time. However, because the server does not necessarily notify the client immediately when a problem is recognized, the error message may be retrieved at any statement executed later.

To control the driver ASYNC communication mode, you must use the following command:

```
DLG SET ".async",mode
```

where mode is either 1 (ASYNC mode active) or 0 (SYNC mode active).

Only the network driver (client-server) can run in ASYNC communication mode, but all drivers accept the syntax above.

Only the following commands will use the ASYNC communication mode:

- DLG SET
- DLG STOP
- DLG NEW
- DLG DEL

During program debug time, it is recommended to use the SYNC mode. (DLG SET ".async",0). This is also required if you expect a command to fail. Default communication mode is SYNC mode.

It might have a performance advantage to use the ASYNC mode during a transfer of a bigger amount of data. e.g. filling a ListBox or a multiline EditText.

Recovering from runtime error in ASYNC mode

If you encounter a runtime error while communicating in ASYNC mode, it’s necessary to re-sync client and server processes again. To accomplish this you should issue a DLG SET ".async" with any mode.

Driver Reference

This chapter describes the drivers provided with Eloquence:

- Motif
- Alpha
- Network

Motif Driver

`motif.drv` is the Eloquence dialog driver for the Motif GUI. It must be installed in the directory **`/opt/eloquence/lbin`**.

Starting the Motif Driver

To start the motif driver from within Eloquence, you issue the following command:

```
   DLG SET ".driver","motif"
```

This will redirect output to your display as defined in the **`DISPLAY`** environment variable. If the **`DISPLAY`** environment variable is not set or if you want to output to a different display you can specify the display address in the driver command using the **`-display`** argument.

The driver provides a method to lookup additional arguments in a *user-defined section* of your eloq.ini file. Using this method, you can maintain system-specific driver arguments at a centralized location (eloq.ini), so you can execute your Eloquence program in different environments without code changes.

Example:

```
  [myoptions]
  Arguments = -display client:0
```

This enables an additional Dialog Manager argument which specifies a different output display *client:0* where **`motif.drv`** will send its output to. In order to activate this item, the name of your *user-defined section* must be specified in the **`DLG SET ".driver"`** statement, e.g.:

```
   DLG SET ".driver","motif myoptions"
```

NOTE: For details about *configuration items* and *user-defined sections* please refer to chapter *Configuring Eloquence*, section *Customize the ELOQ.INI File on the HP-UX System*.

Driver arguments can also immediately be specified in the **`DLG SET ".driver"`** statement. This is useful for arguments which are closely related to **`motif.drv`** and not system-specific, e.g.:

```
DLG SET ".driver","motif myoptions -IDMfont 0 -IDMcolor 0"
```

These two Dialog Manager arguments set the font and color resources to *variant 0*, this is normally appropriate for **`motif.drv`**.

NOTE: If driver arguments are immediately specified in the **`DLG SET ".driver"`** statement, an *user-defined section* must be specified, too. However, if you specify a section name not present in the eloq.ini file, the *user-defined section* will be ignored.

NOTE: For a list of valid command line arguments, please refer to the *ISA Dialog Manager* documentation.

Alpha Driver

`alpha.drv` is the Eloquence dialog driver for text terminals such as HP70092 or VT220. It must be installed in the directory **`/opt/eloquence/lbin`**.

NOTE: The **`alpha.drv`** is different from Eloquence DLG since **`alpha.drv`** uses the Dialog Manager runtime functionality while DLG is a built-in Eloquence component.
Normally, text terminals are directly attached to the system running Eloquence, so DLG performs better compared to **`alpha.drv`** due to the **`alpha.drv`** client-server protocol overhead. For this reason, it is recommended to use Eloquence DLG for text terminals unless you need native Dialog Manager functionality.

Starting the Alpha Driver

To start the alpha driver from within Eloquence, you issue the following command:

```
DLG SET ".driver","alpha"
```

This will redirect output to your display as defined in the **`DISPLAY`** environment variable. If the **`DISPLAY`** environment variable is not set or if you want to output to a different display you can specify the display address in the driver command using the **`-display`** argument.

The driver provides a method to lookup additional arguments in an *user-defined section* of your eloq.ini file. Using this method, you can maintain system-specific driver arguments at a centralized location (eloq.ini), so you can execute your Eloquence program in different environments without code changes.

Example:

```
[myoptions]
Arguments = -display /dev/tty0p6
```

This enables an additional Dialog Manager argument which specifies a different output display */dev/tty0p6* where **`alpha.drv`** will send its output to. In order to activate this item, the name of your *user-defined section* must be specified in the **`DLG SET ".driver"`** statement, e.g.:

```
DLG SET ".driver","alpha myoptions"
```

NOTE: For details about *configuration items* and *user-defined sections* please refer to chapter *Configuring Eloquence*, section *Customize the ELOQ.INI File on the HP-UX System*.

Driver arguments can also immediately be specified in the **`DLG SET ".driver"`** statement. This is useful for arguments which are closely related to **`alpha.drv`** and not system-specific, e.g.:

```
DLG SET ".driver","alpha myoptions -IDMfont 2 -IDMcolor 2"
```

These two Dialog Manager arguments set the font and color resources to *variant 2*, this is normally appropriate for **`alpha.drv`**.

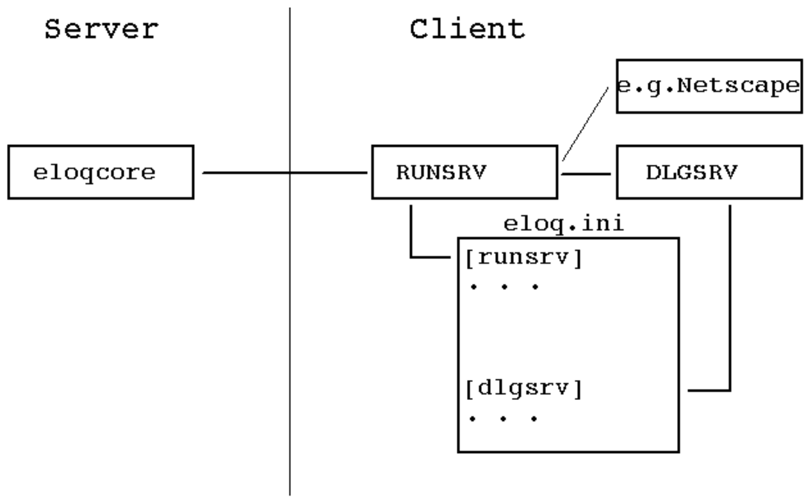
NOTE: If driver arguments are immediately specified in the **`DLG SET ".driver"`** statement, an *user-defined section* must be specified, too. However, if you specify a section name not present in the eloq.ini file, the *user-defined section* will be ignored.

NOTE: For a list of valid command line arguments, please refer to the *ISA Dialog Manager* documentation.

Network Driver

The Eloquence client-server network driver is implemented for the Microsoft Windows GUI. The name of this driver is **`DLGSRV`**. The network driver must even be set if the application server is running on the same system as the GUI server.

Overview



Starting the Network driver

To start the network driver from within Eloquence, you issue the following command:

```
DLG SET ".driver","@client"
```

where *client* is the name of the system running Microsoft Windows where the user interface (dialog server) should run.

NOTE: *Client* must be defined in your **`/etc/hosts`** file. You may also use the IP address instead of the host name.

When starting the network driver, you can pass additional arguments to the Dialog Manager runtime system. The driver provides a method to lookup additional arguments in an *user-defined section* of your eloq.ini file. Using this method, you can maintain system-specific driver arguments at a centralized location (eloq.ini), so you can execute your Eloquence program in different environments without code changes.

Example:

```
[debug]
Arguments = -IDMtracefile C:\TMP\IDMTRACE.TXT
```

This enables an additional Dialog Manager argument which creates a trace file for debugging purposes. In order to activate this item, the name of your *user-defined section* must be specified in the **`DLG SET ".driver"`** statement, e.g.:

```
DLG SET ".driver","@client debug"
```

NOTE: For details about *configuration items* and *user-defined sections* please refer to chapter *Configuring Eloquence*, section *Customize the ELOQ.INI File on the PC Platform*.

Driver arguments can also immediately be specified in the **`DLG SET ".driver"`** statement. This is useful for arguments which are closely related to the network driver and not system-specific, e.g.:

```
DLG SET ".driver","@client myoptions -IDMfont 1 -IDMcolor 1"
```

These two Dialog Manager arguments set the font and color resources to *variant 1*, this is normally appropriate for the Microsoft Windows network driver.

NOTE: If driver arguments are immediately specified in the **`DLG SET ".driver"`** statement, an *user-defined section* must be specified, too. However, if you specify a section name not present in the eloq.ini file, the *user-defined section* will be ignored.

NOTE: For a list of valid command line arguments, please refer to the *ISA Dialog Manager* documentation.

Starting the Network Driver on the Remote System

The network driver must be started on the remote system in order to get connected to the server system running Eloquence. This is done either automatically or manually.

- **Automatic Start-up**

  If the *PortRange* configuration item defined in section *[dmclnt]* of the HP-UX eloq.ini file is nonzero, Eloquence expects the **`RUNSRV`** utility to run on the remote system. **`RUNSRV`** will then be used to start-up the network driver on the remote system.

  This is the recommended method to start-up the network driver.

- **Manual Start-up**

  If the *PortRange* configuration item defined in section *[dmclnt]* of the HP-UX eloq.ini file is zero, Eloquence will wait until the network driver is started manually on the remote system.

  If you choose this method to start-up the network driver, **`DLGSRV`** expects a command line argument of *-connect servername:portnumber* which enables **`DLGSRV`** to connect to the server system (running Eloquence).

  Using this nonstandard method to start-up the network driver is not recommended.

NOTE: Additional information is provided in section *The DLGSRV Utility* later in this chapter.

The RUNSRV Utility

`RUNSRV` is a utility program which is used by Eloquence to start the Eloquence network dialog server (`DLGSRV`) on a remote PC running Microsoft Windows. To achieve this, the `RUNSRV` program must be active on the remote PC.

Starting the RUNSRV Utility If you start `RUNSRV.EXE` on the remote PC, it will minimize itself waiting for a request.

If you restore the `RUNSRV` window, a log of the recent operations performed by `RUNSRV` is displayed. If `RUNSRV` is unable to handle a request, a message box pops up.

NOTE: `RUNSRV` uses the `WINSOCK.DLL` on the PC to interface the network software.

Common Problems `Network software not loaded`

You either did not load the required network software or your network software is not compatible.

`Service runsrv (tcp) not found in your SERVICES file`

You did not install the runsrv service name in your `SERVICES` file.

`Address already in use`

The port number you used by `RUNSRV` is already in use. This may happen if the port number is not unique or if you try to run `RUNSRV` a second time.

`WinExec failed (nn): Error Message`

`RUNSRV` was unable to perform the requested operation. Error number and message reflect a problem detected by Microsoft Windows.

The Remote PC ELOQ.INI File `RUNSRV` uses the configuration information of the *[runsrv]* section in the eloq.ini file on the PC. This file is located in your `WINDOWS` directory.

NOTE: Please refer to chapter *Configuring Eloquence*, section *Customize the ELOQ.INI File on the PC Platform, Section [runsrv]* for details.

The DLGSRV Utility

`DLGSRV` is the Eloquence dialog server for the Microsoft Windows GUI. It runs on the remote PC and contains the Dialog Manager runtime functionality for Microsoft Windows.

NOTE: Please refer to the chapters *Installing Eloquence* and *Configuring Eloquence* for details about installation and PC platform prerequisites. (see chapter , Configuration of the GUI Server,)

Starting the DLGSRV Utility

Normally, **`DLGSRV`** is started automatically by the **`RUNSRV`** utility, which will provide the appropriate command line arguments. This is done when a **`DLG SET ".driver","@client"`** statement is issued. **`RUNSRV`** then uses the configuration item *DlgSrvX* or *DlgSrv* of the *[runsrv]* section in the eloq.ini file on the PC to compose the appropriate command line to start **`DLGSRV`**.

NOTE: Please refer to chapter *Configuring Eloquence*, section *Customize the ELOQ.INI File on the PC Platform, Section [runsrv]* for details about *DlgSrvX* and *DlgSrv*. Additional information is provided in *How to Run Multiple DLGSRV Simultaneously* later in the current section.

NOTE: **`DLGSRV`** uses the **`WINSOCK.DLL`** on the PC to interface the network software.

However, if the nonstandard method of manually starting **`DLGSRV`** is used, **`DLGSRV`** expects a command line argument of *-connect servername:portnumber* which enables **`DLGSRV`** to connect to the server system (running Eloquence):

```
DLGSRV.EXE -connect servername:portnumber [arguments]
```

-connect:	This introduces the connection argument *servername:portnumber*.
servername:	This is the host name of the server system (running Eloquence) **`DLGSRV`** shall connect with.

NOTE: You should provide an appropriate entry for *servername* in your **`HOSTS`** file on the remote PC. The location of your **`HOSTS`** file depends on your networking software.

portnumber:	This is the port number **`DLGSRV`** will use to connect to the server system (running Eloquence).
arguments:	Additional arguments can optionally be specified here and will be passed-through to the Dialog Manager.

NOTE: For a list of valid command line arguments, please refer to the *ISA Dialog Manager* documentation.

When **`DLGSRV`** starts up, it will minimize itself. If you restore the **`DLGSRV`** window, a message box pops up, which allows you to terminate the **`DLGSRV`** program. This may also be achieved by selecting *Close* from the system menu.

NOTE: You should not terminate **DLGSRV** this way since this may lead to loss of data and abnormal termination of the server program. **DLGSRV** will be terminated automatically by Eloquence when processing has finished.
However, to recover from program failure or communication breakdown this method of manually terminating **DLGSRV** provides a very useful “emergency exit”.

The Remote PC ELOQ.INI File **DLGSRV** uses the configuration information of the *[dlgsrv]* section in the eloq.ini file on the PC. This file is located in your **WINDOWS** directory.

NOTE: Please refer to chapter *Configuring Eloquence*, section *Customize the ELOQ.INI File on the PC Platform, Section [dlgsrv]* for details.

How to Run Multiple DLGSRV Simultaneously It is not possible to have more than one running instance of the Eloquence network dialog server for Microsoft Windows (**DLGSRV**). This is a limitation of Microsoft Windows which is not capable to deal with multiple instances of 16bit applications which have multiple writable data segments. In the past, any try to execute a second dialog server resulted in a WinExec error 16.

In order to overcome this limitation (which would prevent you from starting more than one graphical application simultaneously) a workaround has been implemented in Eloquence.

Microsoft Windows recognizes an application by its *module identifier*. A *module identifier* is simply a string at a known location in the executable file holding the internal application name. On application start-up, the Windows kernel checks if an application with this *module identifier* is already running. This has runtime benefits since the code segments of multiple instances are shared in memory.

The workaround to overcome this limitation is implemented by two means:

1 A new **DLGCLONE** utility has been provided to create additional copies of the **DLGSRV** executable, each with a different *module identifier*. These copies are named **DLGSRV2.EXE** up to **DLGSRV9.EXE**.

2 The **RUNSRV** utility now has a built-in mechanism to take care if **DLGSRV** is already active. If it detects an already active **DLGSRV**, it starts one of the additional copies created by **DLGCLONE** instead. This way, multiple instances of **DLGSRV** can be run simultaneously since Microsoft Windows recognizes them as different modules due to different *module identifiers*.

This approach has some drawbacks:

- You need about 1.2 MB of additional disc space for each possible **DLGSRV** instance.
- You need about 1.2 MB of available memory for each running **DLGSRV** instance since

the code segments are not shared between multiple instances.

To activate this mechanism, the eloq.ini file on the PC requires the following changes:

1 Replace the *DlgSrv* configuration item in section *[runsrv]* with *DlgSrvX*. *DlgSrvX* requires an additional entry of **`%s`** immediately following the base name of the **`DLGSRV`** executable file.
If your *DlgSrv* is e.g.:

```
DlgSrv = C:\ELOQ\DLGSRV.EXE -connect %s -IDMfont 1 -IDMcolor 1
```

replace it with:

```
DlgSrvX = C:\ELOQ\DLGSRV%s.EXE -connect %s -IDMfont 1 -IDMcolor 1
```

2 Add the *NDlgSrv* configuration item in section *[runsrv]*. This specifies the maximum number of simultaneously running **`DLGSRV`** instances. The maximum value for *NDlg-Srv* is 9.

If you require e.g. four instances of **`DLGSRV`** running simultaneously, you specify:

```
NDlgSrv = 4
```

3 After saving your changes to the eloq.ini file, run the **`DLGCLONE`** utility.

NOTE: Always run the **`DLGCLONE`** utility after changing the values of *DlgSrvX* and/or *NDlgSrv*. It will automatically adjust the number of copies of the **`DLGSRV`** executable to the value given in the *NDlgSrv* configuration item. The copies will be created in the directory named by the *DlgSrvX* configuration item.

NOTE: The *DlgSrv* configuration item serves for backward compatibility and may not be supported in future releases anymore. *DlgSrvX* is much more flexible and should be used instead.

NOTE: Please refer to chapter *Configuring Eloquence*, section *Customize the ELOQ.INI File on the PC Platform, Section [runsrv]* for details about *DlgSrvX* and *NDlgSrv*.

32bit Network Dialog System

The DLGSRV network dialog driver and the RUNSRV utility are available on the 32bit Windows environment (Windows 95 and NT). They are named DLGSRV32 and RUNSRV32, respectively.

The 32bit version supports the same functions as the 16bit version, so read the section above for further details. The differences are explained below.

NOTE: The RUNSRV32 utility will become obsolete in future releases. Its functionality will be integrated into the eloqd server process.

The DLGSRV32 and RUNSRV32 programs get the configuration from the **eloqcl.ini** configuration file, which is located in the ‘etc’ directory of the Eloquence installation directory.

The DLGCLONE utility is obsolete since DLGSRV32 can be natively used in multiple instances.

Dialog Definitions File

Overview

The layout and some functions of dialogs can be stored in dialog definition files.

This files consists of several parts, which have different functions. To understand how the ISA Dialog Manager works, it's necessary to have some knowlegde about the definitions file.

When starting to create an dialog you have to load some basic definitions, which are defined in the Defaultsfile. During creation of the dialog you add your definitions to this file.This are resource definitions, as fonts or colors, models and the dialog definitions of one or more certain dialogs.

To create the resource definitions and models only ones, it is recommended to use modular definition files. This feature helps to encapsulate reusable information.

The DEFAULTS File

The defaults file is used to provide defaults to dynamically created objects and will be used by the **CVDLG** utility program to convert dialog files. In addition it controls the dynamic behavior of the dialog server.

This chapter will shortly introduce each major element of the Eloquence defaults file. Please refer to the *ISA Dialog Manager* documentation for further reference.

Callback Functions

The callback functions are used by Eloquence to filter unexpected events. If a unexpected event is recognized, it will be ignored. All further processing is done through the appropriate rules.

Eloquence needs the functions below to keep track of the current object state.

Function name	**Called**
EqPushButtonCB	if a pushbutton is selected
EqCheckBoxCB	if a checkbox is (de-) activated
EqRadioButtonCB	if a radiobutton is activated
EqEditTextCB	if an edittext looses focus

Returning from DLG DO

Returning from the **`DLG DO`** statement is achieved by calling the procedure **`EqExitEventLoop`**. It will return control back to Eloquence with a given return code and an object path.

Function name	**Description**
EqExitEventLoop	This will terminate the **`DLG DO`** statement with the given object and rule value.

Resources

Resources are platform dependent. *Resource variants* allow to have the same dialog source file for all platforms.

Whenever there are resource variants, they are used in the following manner:

Variant	**Platform**
0	Motif GUI
1	Microsoft Windows GUI
2	Alpha Windows

To activate the appropriate resource variant you specify it on the driver command line, e.g.:

```
-IDMfont 1 -IDMcolor 1
```

This will start-up Dialog Manager with font and color variant 1. Please refer to the *ISA Dialog Manager* documentation for a description of valid command line arguments.

The following font resources are allocated by default:

- **`BaseFont`** is used to calculate all sizes and positions.
  BaseFont should be a fixed space font.
- **`NormalFont`** is used to display all text fields

Attributes

There are some user-defined attributes allocated for use by Eloquence:

- **`integer EqRule`**

This is used to emulate the Eloquence *rule* attribute.

- **`integer EqActiveline`**
  This is used to emulate the Eloquence *activeline* attribute.
- **`integer EqKbRule[10]`**
  This is used to emulate the Eloquence function key handling.

Popup Box

The Eloquence **`POPUP BOX`** statement is mapped to the **`EqPopup`** window and its associated rules.

It will be configured dynamically by Eloquence according to the needs of a **`POPUP BOX`** statement.

Rules

- **`rule boolean EqSaveDialog( string Fname )`**
  This is used by the **`CVDLG`** utility to save the converted dialog files.
- **`on CHECKBOX activate, deactivate`**
  This is used to map *CheckBox* behavior.
- **`on RADIOBUTTON activate`**
  This is used to map *RadioButton* behavior.
- **`on LISTBOX dbselect`**
  This is used to map *ListBox* behavior.
- **`on LISTBOX key EqKbSelect`**
  This is used to map *ListBox* behavior.
- **`on EDITTEXT deselect`**
  This is used to map *EditText* behavior.
- **`on dialog key EqKbF1`** … **`on dialog key EqKbF8`**
  This is used to map Eloquence function key handling.

Using Modular Dialog Files

Dialog Manager Versions A.03.02a and above allow modular dialog files. This new feature helps to encapsulate reusable information into module files while the dialog file itself is stripped down to its dialog specific information.

NOTE: This feature is currently unsupported. There is a severe bug in the underlying ISA Dialog Manager library (a file descriptor leak), which will cause a fatal dialog server failure after processing a few dialogs.

Although this feature has bugs and is therefore unsupported, we strongly recommend that you get in touch with modular dialog files, because they have great advantages due to reduced redundancies. This will help you to reduce maintenance effort significantly.

The next Eloquence patch will provide full support for modular dialog files assuming the Dialog Manager bug has been fixed.

You find a detailed description of modular dialogs in the Dialog Manager release notes 8/95, Version A.03.02a. These release notes will discuss only Eloquence specific issues.

There is a new modular Eloquence *defaults.eq* file. You find it and the related modules at the following locations:

HP-UX:	`/opt/eloquence/lib/module`
Windows:	`C:\DLG\MODULE`

This file contains the include directives for the following modules:

eqrsrc	fonts and colors
eqbind	functions, accelerators, standard rules
eqdef	defaults
eqpopup	EqPopup window

You find these files in the module directory named above.

Additionally, there are the related interface files with the extension `.if`.

To use these modules, follow these steps:

1. In order to access the module files, you must establish the **`IDMLIB`** access path. This can be done either by using the *IdmLib* configuration item in the eloq.ini file or by setting the **`IDMLIB`** environment variable.

   - **HP-UX: eloq.ini, section [dmsrv]**
     IdmLib = /opt/eloquence/lib/module:/opt/eloquence/lib/mymodule
   - **HP-UX: `IDMLIB` environment variable**

IDMLIB = /opt/eloquence/lib/module:/opt/eloquence/lib/mymodule
export IDMLIB
(You should include these statements in your *.profile*.)

- **Windows: eloq.ini, section [dlgsrv]**
  IdmLib = C:\DLG\MODULE;C:\DLG\MYMODULE
- **Windows: `IDMLIB` environment variable**
  set IDMLIB = C:\DLG\MODULE;C:\DLG\MYMODULE
  (You should include these statements in your *AUTOEXEC.BAT*.)

The *mymodule* directory is an example for a directory containing your own modules. The directories are separated by a colon ('`:`', Motif) or a semicolon ('`;`', Microsoft Windows).

2 Adjust the following files according to the contents of your current *defaults.eq* file (if you have never changed the *defaults.eq* please continue with step 4):

- eqrsrc.mod
- eqbind.mod
- eqdef.mod
- eqpopup.mod

3 If you had to change the files in step 2 you should re-create the related interface files.

To do this you change to the module directory named above and execute the **`idm`** utility with the +*writeexport* option:

```
idm +writeexport interfacefile modulefile
```

Example:

```
idm +writeexport eqdef.if eqdef.mod
```

On Microsoft Windows you should specify the full access path to the **`idm`** utility and the interface file. You can use the **`IDMLIB`** value to access the module files (if the **`IDMLIB`** environment variable has been defined), e.g.:

```
C:\IDM\IDM +writeexport C:\DLG\MODULE\EQDEF.IF IDMLIB:EQDEF.MOD
```

Repeat this for each of the four files in the following order:

• eqrsrc.mod	(eqrsrc.if)
• eqbind.mod	(eqbind.if)
• eqdef.mod	(eqdef.if)
• eqpopup.mod	(eqpopup.if)

If you do not have the Dialog Manager development software you should manually adapt the existing interface files:

- Each object defined and exported in a module file has a corresponding declaration statement in the associated interface file.
- So, if you removed exported objects from a module file, you should lookup the corresponding declaration in the interface file and remove it, too.

- If you added new exported objects, you should add a corresponding declaration statement to the interface file.

4 At last you should change your eloq.ini configuration file. You should add a new *user-defined section* which overrides the *DefaultsFile* configuration item to use the new modular one. When starting the dialog server with the **`DLG SET`** statement, specify this configuration name as an argument, e.g:

```
[UseModules]
DefaultsFile = /opt/eloquence/lib/module/defaults.eq

DLG SET ".driver","motif UseModules"
```

Alternatively, you can change the *DefaultsFile* setting in the default dialog server section to use the modular *defaults.eq* file:

- **HP-UX:** `DefaultsFile=/opt/eloquence/lib/defaults.eq`
- ***change to*:** `DefaultsFile=/opt/eloquence/lib/module/defaults.eq`
- **Windows:** `DefaultsFile=C:\DLG\DEFAULTS.EQ`
- ***change to*:** `DefaultsFile=C:\DLG\MODULE\DEFAULTS.EQ`

Using Models

Dialog Manager models provide an efficient way to handle similar objects instead of defining a lot of individual objects.

Example:

```
model statictext YellowText
{
   .fgc ColBlack;
   .bgc ColYellow;
   .width 10;
   .text "YellowText";
}
...
child YellowText Yt1
{
   .xleft 30;
   .width 15;
   .ytop 0;
}
```

This defines the model *YellowText*. It is derived from the *statictext* object default, but defines its own attribute default values.

The instance *Yt1* of the *YellowText* model will inherit all attributes as defined by the *statictext* default and the *YellowText* model.

The resulting object will display "YellowText" with black text on a yellow background.

Models may also define new attributes for Dialog Manager objects and may have default rules associated with them.

The following example will define the model *MyMenuItem* which is derived from the default *menuitem* object type. *MyMenuItem* defines an *EqRule* attribute and a rule causing return from **`DLG DO`** statement to Eloquence when an instance of the *MyMenuItem* is selected and this instance has a nonzero *EqRule* value:

```
model menuitem MyMenuItem
{
   .text "MyMenuItem";
   integer EqRule := 0;
}
...
child menubox
{
   .title "&File";
   child MyMenuItem About
   {
      .text "&About";
      .EqRule := 1;
   }
   ...
   child MyMenuItem Exit
```

```
      {
         .text "&Exit";
         .EqRule := 2;
      }
   }
   ...
   on MyMenuItem select
   {
      if this.EqRule then
         EqExitEventLoop(this, this.EqRule);
      endif
   }
```

NOTE: Please refer to the *ISA Dialog Manager* documentation for details.

NOTE: When a new dialog object is dynamically created with the **`DLG NEW`** statement (where the *object path* and an *object type* must be provided), you may specify a Dialog Manager object model instead of the *object type*.

On-line Help

The Eloquence dialog server provides a mechanism which calls an external program to provide (context sensitive) on-line help.

The Eloquence dialog server relies on the popular Netscape WWW browser to provide on-line help. This makes it possible to have the on-line documentation in standard HTML format placed on a server system (running a HTTP server) or in a local directory.

As another benefit, the on-line documentation format is portable across the different systems such as HP-UX and Microsoft Windows which allows easier maintenance of the documentation files.

NOTE: This functionality requires Netscape revision 1.1 or above.

NOTE: The help mechanism built into former Eloquence dialog server releases worked different, depending on the dialog server. The motif dialog server called the HP VUE help subsystem (by executing *helpview*). The Microsoft Windows dialog server called the Microsoft Windows help subsystem. This implied different documentation formats from system to system. The help context passed to the external program was created from information contained in the dialog variable *HelpVolume* and an object specific help attribute.

NOTE: Netscape is not included with Eloquence. It can be purchased separately from *Netscape Communications Corp*. Netscape can either be obtained from a local distributor or may be downloaded from the anonymous ftp server of *Netscape Communications Corp.* at *ftp.netscape.com*.

The Microsoft Windows Platform

The Microsoft Windows dynamic data exchange (DDE) communication protocol is used to communicate with Netscape.

NOTE: Please refer to section *The RUNSRV Utility, RUNSRV DDE Communication* prior in this chapter for details about Microsoft Windows dynamic data exchange.
Additional information is provided in chapter *Configuring Eloquence*, section *Customize the ELOQ.INI File on the PC Platform, Section [modules]*.

The Motif Platform

Eloquence uses the communication mechanism built in Netscape. It calls the Netscape executable passing any arguments on the command line.

Netscape will then pass the request to an already running Netscape process on the current display. If Netscape is not currently running on the display, it will be started. In order to locate the Netscape executable, the dialog server relies on the configuration item *Netscape* in section *[dmsrv]* of the eloq.ini file on the HP-UX system.

NOTE: Please refer to chapter *Configuring Eloquence*, section *Customize the ELOQ.INI File on the HP-UX System, Section [dmsrv]* for details.

Accessing the On-line Documentation

Before the on-line documentation can be accessed, two configurations are necessary:

1 The configuration item *HelpBaseURL* defines the base location of the on-line documentation. It is defined in the *[dlgsrv]* of the file **eloqcl.ini**.

Example:

```
HelpBaseURL = http://www/application/help/
```

You can think of *HelpBaseURL* as the on-line help document root for a specific Eloquence application. Below this root there should be separate hierarchy levels, one for each functional area inside the application.

NOTE: Please refer to chapters *Configuring Eloquence*, section *Customize the ELOQ.INI File on the HP-UX System, Section [dlgsrv]* and *Configuring Eloquence*, section *Customize the ELOQ.INI File on the PC Platform, Section [dlgsrv]* for details.

2 The *EqHelpPath* dialog variable defines the on-line document location relative to *HelpBaseURL*. This variable can either be set to a fixed value in your Dialog Manager dialog file, e.g.:

```
config variable string EqHelpPath := "accounting/Dialog.html";
```

Alternatively, it can be dynamically set in your Eloquence program using the **`DLG SET`** statement, e.g.:

```
DLG SET "EqHelpPath!value","accounting/Dialog.html"
```

You can think of *EqHelpPath* as the on-line help document hierarchy level for a specific functional area inside the Eloquence application. The objects inside this functional area should be each associated with a separate *help tag* in order to realize a context sensitive on-line help system.

NOTE: Please refer to section *Dialog Manager, Accessing the Dialog Manager Variables* later in this chapter for details.

If these prerequisites are fulfilled, a *help tag* can be defined for every dialog object using the **`.help`** attribute. This is normally done in your Dialog Manager dialog file, e.g.:

```
window My_window
{
    ...
    .help "#My_window";
    ...
}
```

Alternatively, it can be dynamically set in your Eloquence program using the **`DLG SET`** statement, e.g.:

```
DLG SET "My_window.help","#My_window"
```

With these settings, the final document address is composed by concatenating *HelpBaseURL*, *EqHelpPath* and *My_window.help*. The resulting document is:

```
http://www/application/help/accounting/Dialog.html#My_window
```

Pressing the F1 function key or triggering an *EqRule -1* will then automatically invoke the Netscape WWW browser, which in turn loads the resulting document named above.

NOTE: You can specify a different **`.help`** attribute for each dialog object in order to provide real context sensitive on-line help. However, if an object does not have a **`.help`** attribute, the parent object's **`.help`** attribute is used if present. This way, the object hierarchy is traced up to the root (*window*) object until a **`.help`** attribute is found.

NOTE: If you want to arrange your on-line help document hierarchy in the local file system rather than on a HTTP server, you simply define a different *HelpBaseURL*, e.g.:

```
HelpBaseURL = file://opt/application/help/
```

The DLG HELP Statement

The **`DLG HELP`** statement is now implemented with the dialog server. It provides access to a specific *help tag* as if the F1 function key had been pressed or an *EqRule -1* had been triggered.

Following the example in the former section, if you issue the following statement:

```
DLG HELP "#My_window"
```

Netscape will be invoked loading the same document as in the example above.

The EqHelpOnObject Function

This function is used internally to enable context sensitive on-line help when pressing the F1 function key or triggering an *EqRule -1*.

You can apply this function inside a Dialog Manager rule, e.g.:

```
on WINDOW help
{
   EqHelpOnObject(this);
}
```

It cannot be applied inside a Eloquence program, since it expects an *object identifier* argument which Eloquence cannot provide.

This function must be declared in your Dialog Manager dialog file if you want to apply it, the declaration should look like:

```
function boolean EqHelpOnObject(object input);
```

Please refer to the file **`/opt/eloquence/lib/defaults.eq`**, where you will find the function declaration as well as the **`on WINDOW help`** rule.

The EqHelpOnTag Function

This function is similar to the **`EqHelpOnObject`** function, except that an additional parameter must be provided which refers to a specific *help tag*.

You can apply this function inside a Dialog Manager rule to provide on-line help for a specific *help tag*, e.g.:

```
on Help_on_help_button select
{
   EqHelpOnTag(this,"#HelpOnHelp");
}
```

It cannot be applied inside a Eloquence program, since it expects an *object identifier* parameter which Eloquence cannot provide.

This function must be declared in your Dialog Manager dialog file if you want to apply it, the declaration should look like:

```
function boolean EqHelpOnObject(object input, string input);
```

Please refer to the file **`defaults.eq`**, where you will find the function declaration.

The EqHelpViewFile Function

You apply this function to view the contents of any text file using the Netscape WWW browser.

The configuration item *FileBaseURL* defines the base location of the files to be viewed. This is similar to the *HelpBaseURL* configuration item.

On the HP-UX system, you define *FileBaseURL* in section *[dmsrv]* of the eloq.ini file on the HP-UX system.

On the PC platform, you define *FileBaseURL* in section *[dlgsrv]* of the eloq.ini file on the PC.

Example:

```
FileBaseURL = http://www/application/documents/
```

You can think of *FileBaseURL* as the root directory containing any files related to a specific Eloquence application.

NOTE: Please refer to chapters *Configuring Eloquence*, section *Customize the ELOQ.INI File on the HP-UX System, Section [dlgsrv]* and *Configuring Eloquence*, section *Customize the ELOQ.INI File on the PC Platform, Section [dlgsrv]* for details.

The **`EqHelpViewFile`** function expects a *string* argument which is appended to *FileBaseURL* to compose the final document address.

Example:

```
DLG CALL FUNCTION "EqHelpViewFile"("order.lst")
```

This will invoke Netscape, which in turn will load the following document:

```
http://www/application/documents/order.lst
```

This function must be declared in your Dialog Manager dialog file if you want to apply it, the declaration should look like:

```
function boolean EqHelpViewFile(string input);
```

Please refer to the file **`defaults.eq`**, where you will find the function declaration.

NOTE: If you want to arrange your file hierarchy in the local file system rather than on a HTTP server, you simply define a different *FileBaseURL*, e.g.:

```
FileBaseURL = file://opt/application/documents/
```

The EqHelpManPage Function

You apply this function to view the contents of any manual page using the Netscape WWW browser.

The configuration item *ManBaseURL* should be set to the location of a *CGI script* providing the functionality to retrieve a manual page using a search expression.

On the HP-UX system, you define *ManBaseURL* in section *[dmsrv]* of the eloq.ini file on the HP-UX system.

On the PC platform, you define *ManBaseURL* in section *[dlgsrv]* of the eloq.ini file on the PC.

Example:

```
ManBaseURL = http://www/cgi-bin/man2html
```

NOTE: Please refer to chapters *Configuring Eloquence*, section *Customize the ELOQ.INI File on the HP-UX System, Section [dlgsrv]* and *Configuring Eloquence*, section *Customize the ELOQ.INI File on the PC Platform, Section [dlgsrv]* for details.

The **`EqHelpManPage`** function expects a *string* argument which is appended to *ManBaseURL* to compose the final man page address.

Example:

```
DLG CALL FUNCTION "EqHelpManPage"("?man")
```

This will invoke Netscape, which in turn will load the following document:

```
http://www/cgi-bin/man2html?man
```

This function must be declared in your Dialog Manager dialog file if you want to apply it, the declaration should look like:

```
function boolean EqHelpManPage(string input);
```

Please refer to the file **`/opt/eloquence/lib/defaults.eq`**, where you will find the function declaration.

The Remote Exec Utility

The **EQEXEC** utility is provided for the Microsoft Windows platform. It makes it possible to start HP-UX processes on a remote host using the *rexec* protocol.

The EQEXEC remote exec utility is available on the 32bit Windows environment and is now named EQEXEC32. It uses the eqexec.ini configuration file located in the Windows installation directory.

EQEXEC enables you to start your HP-UX application without the necessity to login to your favorite HP-UX box while you don’t get in contact with the operat ing system at all.

EQEXEC enables you to start your HP-UX application without the necessity to login to your favorite HP-UX box while you don’t get in contact with the operating system at all.

EQEXEC is designed for end users to start their Eloquence applications on a PC running Microsoft Windows using the **DLGSRV** network dialog server. This way, Eloquence applications behave like native Microsoft Windows programs.

Probably, **EQEXEC** may be used in a more general way since the *rexec* protocol enables you to start any process on any remote host running a remote execution server.

NOTE: For details about the *rexec* protocol please refer to the *rexecd(1m)* HP-UX manual page.

EQEXEC Features

The **EQEXEC** utility

- is fully localizable by changing a configuration file
- supports multiple hosts, applications and users
- provides a command line per application (up to 500 characters)
- has a built-in configuration screen, so it is easy to use and maintain
- includes a log window for debugging purposes

As an additional benefit, if your Eloquence application has been migrated to Dialog Manager client-server dialogs, you don't need a HP-UX user license (since the HP-UX system is used as an application server only) and a costly terminal emulator such as Reflection in order to start your application.

The EQEXEC.INI File

The **`EQEXEC`** utility relies on the eqexec.ini configuration file located in your **`WINDOWS`** directory. Part of this file is automatically maintained by **`EQEXEC`**, part of it can be manually configured. The eqexec.ini file is fully documented and contains a complete description of the various configuration items.

Manual configuration includes

- the common runtime behavior (e.g. if a password is required generally)
- customizing the dialog caption and control element titles
- customizing the program error messages

Using EQEXEC

`EQEXEC` pops up with a dialog. It contains the following control elements:

Application: Select the application you want to execute.

NOTE: The *application* is saved for the next time you use **`EQEXEC`**.

Username: Enter your user name to log in on the remote host.

NOTE: The *username* is saved for the next time you use **`EQEXEC`**.

Password: Enter your password to log in on the remote host.

NOTE: The *password* is *not* saved for the next time you use **`EQEXEC`**.
It depends on the *PasswordRequired* configuration item in section *[common]* of the eqexec.ini configuration file if a password is required generally.

Log output: Check this option to show the host response in a log window. This is useful if a command does not execute as expected.

NOTE: The state of the *log output* check box is saved for the next time you use **`EQEXEC`**.

OK: Push this button to execute the selected application.

Cancel: Push this button to leave **`EQEXEC`**.

Settings: Push this button to open the configuration sub-dialog.

The configuration sub-dialog enables you to maintain the application list. It contains the following control elements:

Description: This is the application title to be displayed in the list.

Host: This is the name of the remote host where the application shall be executed.

NOTE: You should provide the appropriate entry in your **`HOSTS`** file. The location of your **`HOSTS`** file depends on your networking software.

Command: This is the command line to be executed on the remote host.

NOTE: Is is recommended to redirect *standard input* to the *null device*, this is done by appending **`</dev/null`** to the command line.
When the command is executed as expected, you should also redirect *standard output* and *standard error* to the *null device* since this reduces the network traffic. This is done by appending **`>/dev/null 2>&1`** to the command line.

Add: Push this button to put your changes to the application list.

Delete: Push this button to delete the selected entry from the application list.

NOTE: Since every control element title can be customized, the titles in this description may differ from the current **`EQEXEC`** settings.

The CVDLG Utility

`CVDLG` is a HP-UX utility which converts Eloquence dialog files into Dialog Manager dialog files.

Eloquence dialog files, if accessed by a driver, will be converted temporarily into Dialog Manager format each time a dialog is loaded. However, this is a time consuming process and you may also wish to optimize the dialog layout using the Dialog Manager graphical editor. You may use `CVDLG` to convert Eloquence dialog files into Dialog Manager dialog files.

Dialog File Name Extensions

The following file name extensions are recommended:

.dlg	Eloquence dialog file
.idm	Dialog Manager dialog file
.idc	compiled Dialog Manager dialog file

The default file name extensions may be re-defined using the eloq.ini file.

NOTE: Compiled Dialog Manager dialog files are platform specific.

Using CVDLG

`CVDLG` is used with the following command line:

`cvdlg [`*`arguments`*`] outfile infile [infile ...]`

outfile: This specifies the destination file name. An extension of `.idm` is recommended.

infile: This lists the Eloquence dialog files to be converted. At least one *infile* must be specified.

The optional *arguments* must be supplied in the following order:

1 **`-help`**
Show program usage.

2 **`-debug`*`n`***
Set the internal `CVDLG` debug level to ***`n`***.

3 **`-driver`** ***`driver arguments`***
Specify the dialog driver and additional driver arguments. This is equivalent to the parameter supplied to the `DLG SET ".driver"` statement. Please refer to section *Eloquence Dialog Drivers* prior in this chapter.

Example:

```
   -driver "motif myoptions -IDMtracefile /tmp/idmtrace"
```

This will start the motif driver, specifying the *user-defined section* named *[myoptions]* and an additional Dialog Manager argument which creates a trace file for debugging purposes.

NOTE: If **`-driver`** is not specified, the **`DRIVER`** environment variable is used instead. This variable should be set to ***`driver arguments`***, e.g.:

```
DRIVER = "motif myoptions -IDMtracefile /tmp/idmtrace"
```

NOTE: **`CVDLG`** uses Eloquence drivers to convert dialog files.
If using the network driver, the output files will be created on the system running the **`DLGSRV`** utility. So the destination file name must be specified according to the remote system requirements. If you specify a relative file name, the target file will be created relative to the directory specified in the eloq.ini file.

CVDLG Examples

Example 1:

```
cvdlg -driver "motif myoptions" sample.idm sample.dlg
```

This example will convert a Eloquence dialog file named *sample.dlg* into a Dialog Manager dialog file named *sample.idm* using the motif driver. The *user-defined section* named *[myoptions]* is used for driver configuration.

Example 2:

```
cvdlg -driver @client c:\\dlg\\sample.idm c:\\dlg\\sample.dlg
```

This example will convert a Eloquence dialog file named *sample.dlg* into a Dialog Manager dialog file named *sample.idm*. The network dialog driver **`DLGSRV`** is used to perform this conversion, running on the remote PC named *client*.

Any *backslash* path separator character must be doubled since the shell uses a *backslash* as an escape character. You may use the regular HP-UX *slash* character instead, it will be mapped to *backslash* by the dialog driver.

The Conversion Process

A Eloquence dialog file is translated in a sequence of **`DLG NEW`** and **`DLG SET`** statements which are executed by the driver process.

The following sequence of Eloquence statements reflect the internal operation of **`CVDLG`**:

```
10 DLG SET ".driver","motif"
20 DLG LOAD "sample.dlg"
```

```
   30 DLG CALL RULE "EqSaveDialog","EqDialog"("sample.idm")
   40 DLG STOP
```

The following example script will show a possible usage of the **CVDLG** utility using the motif driver. It will convert **.dlg** files to **.idm** files. The dialog files are afterwards post-processed by *sed* (the dialogs are renamed in this step).

If no file names are given on command line, this script will convert all **.dlg** files in the current directory which have no corresponding **.idm** files. If any file name is specified on the command line, the corresponding **.idm** files are overwritten.

```
#! /bin/sh
echo "mkidm.sh"
force=n
defaults=defaults.eq

if [ -z "$DISPLAY" ]
then
   echo "Motif display required!"
   exit 1
fi

if [ $# != 0 ]
then
   list=$*
   force=y
else
   list="*.dlg"
fi

for i in $list
do
   base=‘basename $i .dlg‘
   idm=$base.idm
   idc=$base.idc

   if [ ! -f $idm -o $force = y ]
   then
      echo "\\n$i -> $idm"

      rm -f $idm $idc

      /usr/eloquence/cvdlg -driver motif +arg $defaults $idm $i
      if [ $? != 0 ]
      then
         echo "failed!!"
         exit
      fi

      sed -e "s/EqDialog/Dlg_$base/g" %<$idm >$$
      if [ $? != 0 ]
      then
         echo "sed failed!!"
         exit
      fi
      mv $$ $idm
   fi
done
```

Advanced Dialog Manager usage

Overview

In some cases it is necessary to use some not standard functions, to implement certain tasks. The following specials perhaps can help you to manage it.

Accessing Dialog Manager Variables

Dialog Manager variables may be set using the **DLG SET** and retrieved using the **DLG GET** statement. The dialog name, the variable name and the *.value* attribute must be specified.

Example:

```
dialog YourDialog
...
variable string StrVal;
variable integer IntVal;
```

You may access these variables from Eloquence using the following statements:

```
DLG GET "YourDialog.StrVal!value",A$
```

This will retrieve the value of the Dialog Manager variable *StrVal* into the Eloquence variable *A$*.

```
DLG SET "YourDialog.IntVal!value",123
```

This will set the Dialog Manager variable *IntVal* to the constant value *123*.

NOTE: You should use the exclamation mark ('**!**') to delimit variable path and attribute because *native* Dialog Manager attributes are accessed. Please refer to *Accessing Dialog Manager Objects* later in this chapter.

NOTE: Please refer to the *ISA Dialog Manager* documentation for details.

Dialog Manager Records

Eloquence supports access to Dialog Manager *record objects*.

Dialog manager record objects are the equivalent of *structures* in *C language* and may be defined globally to the dialog or locally to any object.

This is an efficient method to exchange multiple data with Dialog Manager at once. Using the Eloquence **`XPACK`** and **`XUNPACK`** statements, you may link Eloquence variables to Dialog Manager record members.

Dialog Manager record members may either be defined as shadow objects, such that each access to a record member will affect the associated object attribute, or they may contain data which may be used in Dialog Manager rules or functions.

The **`DLG SET`** and **`DLG GET`** statement mappings of the dialog driver have been enhanced to support Dialog Manager record objects.

The DLG SET Statement

The **`DLG SET`** statement may be used to transfer a buffer, packed by the **`XPACK`** statement, into equivalent record members of a Dialog Manager record object.

```
DLG SET "Record.@",Buf$
```

NOTE: The .@ attribute must be specified.

Example:

```
window CusWin
{
   ...
   child record CusRec
   {
      string Cus_no shadows CusWin.Cus_number.content;
      string Cus_name shadows CusWin.Cus_name.content;
      boolean Cus_flag shadows CusWin.active;
      ...
   }
}

Cus_no$="1234"
Cus_name$="Customer Name"
Cus_flag=1
XPACK Buf$ FROM Cus_no$,Cus_name$,Cus_flag
DLG SET "CusWin.CusRec.@",Buf$
```

This will transfer the Eloquence variables *Cus_no$*, *Cus_name$* and *Cus_flag* into the Dialog Manager record object *CusWin.CusRec*.

The DLG GET Statement

The **`DLG GET`** statement may be used to transfer the contents of a Dialog Manager record object into Eloquence program variables.

If a .@ attribute is specified, all record members are transferred. If a member name is specified, only the given record member is transferred:

```
DLG GET "Record.@",Buf$
```

This transfers all members of *Record* into *Buf$*.

```
DLG GET "Record.Array",Buf$
```

This transfers all elements of *Record* member *Array* into *Buf$*.

```
DLG GET "Record.Array[1]",Buf$
```

This transfers the first element of *Record* member *Array* into *Buf$*.

Example:

```
DLG GET "CusWin.CusRec.@",Buf$
XUNPACK Buf$
```

This will transfer all members of the Dialog Manager record object *CusWin.CusRec* into equivalent Eloquence program variables.

```
DLG GET "CusWin.CusRec.Cus_flag",Buf$
XUNPACK Buf$
```

This will transfer the field *CusWin.CusRec.Cus_flag* into the Eloquence program variable *Cus_flag*.

Programming Considerations

In order to use Dialog Manager record objects, the following considerations should be followed:

- The Dialog Manager does not support floating point variables.
- All record member names must be valid Eloquence variable names.
- The Dialog Manager record members must have the appropriate type. You may use the Dialog Manager *boolean*, *integer* and *string* data types.
- The Dialog Manager record members and Eloquence program variables should have an equivalent type.
- While it is possible to transfer Eloquence numeric variables into Dialog Manager *string* record members, the opposite is not possible.

  However, there is a workaround implemented into Eloquence which allows to overcome this limitation:

A Dialog Manager record member with a trailing ’**N**’ (uppercase) appended to its name is considered numeric, regardless of its real data type. The corresponding Eloquence variable name does *not* include the trailing ’**N**’. This way, numeric data can be transferred from any Dialog Manager record member data type into Eloquence numeric variables.

Example:

```
window MyWin
{
   child record MyRec
   {
      string Num_strN "123";
   }
}


INTEGER Num_str
DLG GET "MyWin.MyRec.@",Buf$
XUNPACK Buf$
```

This will transfer the numeric value *123* from the Dialog Manager record member *MyWin.MyRec.Num_strN* (*string* data type) into the Eloquence variable *Num_str* (*INTEGER* data type).

NOTE: If such Dialog Manager record members contain non-numeric text, *0* (zero) is transferred into the corresponding Eloquence variables.

Dialog Manager Rules and Functions

Eloquence provides two statements which enable you to call Dialog Manager *rules* and *functions*.

Rules

Dialog Manager *rules* are implemented using the Dialog Manager script language and are stored in the dialog file. This makes it possible to provide a task-oriented interface to dialogs without having to care about the bits and pieces.

By writing your own rules you actually extend the dialog server functionality. The **DLG CALL RULE** statement allows you to trigger such extensions:

```
DLG CALL RULE "Rule","Object" [(arg,arg ...)] [,Retvar[$]] [;Err]
```

Example:

```
window MyWindow
{
   ...
   child image MyImage
   {
      ...
```

```
         .text "Image Name";
         .picture Default;
      }
      ...
   }
   ...
   rule void LoadGif( string Gif input, string Name input )
   {
      this.picture := Gif;
      this.text := Name;
   }

   DLG CALL RULE "LoadGif","MyImage"("sample.gif","Sample Image")
```

This will trigger the **`LoadGif()`** rule. The object which the rule is applied to (*this*) is *MyImage*, the GIF image to be loaded is *sample.gif* and the image title is set to *"Sample Image"*.

The **`DLG CALL RULE`** statement is mapped to a **`DM_CallRule()`** Dialog Manager function call.

If a return variable is present, the value returned by the rule will be assigned to it. If no return variable is specified, the return value will be ignored.

If the error return variable is present, no runtime error is returned, but the error variable is set with the error number.

NOTE: In the example above, the GIF file will be read from the local system (where the dialog server is executed). If you are using the network dialog server on the PC platform, this file is expected on the PC. This behavior is subject to change in a subsequent release.

Functions

Dialog Manager *functions* are implemented in *C language* and are linked to the dialog server executable. The functions are bound to the Dialog Manager using the **`DM_BindFunctions()`** Dialog Manager function. In order to use such functions they must be declared in the dialog file.

By writing your own functions you actually extend the dialog server functionality. The **`DLG CALL FUNCTION`** statement allows you to trigger such extensions:

```
DLG CALL FUNCTION "Function" [(arg,arg ...)] [,Retvar[$]] [;Err]
```

Example:

```
   function boolean EqHelpViewFile(string input);

   DLG CALL FUNCTION "HelpViewFile"("sample.txt")
```

This will trigger the **`HelpViewFile()`** function. The file to be viewed is *sample.txt*.

The **`DLG CALL FUNCTION`** statement is mapped to a **`DM_CallFunction()`** Dialog Manager function call. The maximum number of arguments is 8.

If a return variable is present, the value returned by the rule will be assigned to it. If no return variable is specified, the return value will be ignored.

If the error return variable is present, no runtime error is returned, but the error variable is set with the error number. If no dialog server is active, an error 1004 is returned.

NOTE: Please refer to section *Customizing the Dialog Server* prior in this chapter for details about extending the dialog server.

Programming Notes

This section covers solutions to common problems concerning Dialog Manager programming issues.

Avoiding Edit Text Exit Rule

Problem: **`on EDITTEXT deselect`** is triggered at **`PUSHBUTTON select`**.

If a *pushbutton* or *image* button is selected, this will involve a focus change which will trigger an **`EDITTEXT deselect`** rule. As a result, it is required to select the *pushbutton* twice.

This behavior can also result in a situation where a “Cancel” *pushbutton* would be unusable if an **`EDITTEXT deselect`** rule performs a field validation and re-sets the focus on itself again.

This problem can be solved in the following manner:

1 Initialize an *integer* variable in your *pushbutton* object or in the object model or default. Do this in every object (model, default) that should have “Cancel” *pushbutton* characteristics, e.g.:

```
model pushbutton Cancel_btn
{
   ...
   integer EqRuleOverride := 1;
}
```

2 Change the object (model, default) rule that normally is triggered **`on EDITTEXT deselect`**.

If this rule is e.g.:

```
on EDITTEXT deselect
{
   if this.EqRule then
      EqExitEventLoop(this, this.EqRule);
   endif
}
```

Change this rule to:

```
on EDITTEXT deselect
{
   variable integer RuleOverride := 0;
   if (typeof(this.window.focus.EqRuleOverride)=integer) then
      RuleOverride := this.window.focus.EqRuleOverride;
   endif
   if RuleOverride=0 then
      if this.EqRule then
         EqExitEventLoop(this, this.EqRule);
      endif
   endif
}
```

The basic idea is to obtain the newly focused object (*this.window.focus* which might be the “Cancel” *pushbutton* object) and check if this object has an *EqRuleOverride* attribute. If this is true and *EqRuleOverride* is set the **`EqExitEventLoop()`** function will not be triggered and the **`EDITTEXT deselect`** rule will not perform any action.

Setting the Focus to Microsoft Windows Radio Button Object

Previously, a **`DLG SET "Radiobutton.focus",1`** statement always *activated* the chosen *radiobutton* when applied with the Microsoft Windows network dialog server. This behavior sometimes had side effects if a rule was associated with the activation of this *radiobutton* and the *radiobutton* had not previously been selected.

The current implementation sets the focus to the *currently selected radiobutton within the same group* whenever a **`DLG SET`** statement is used as shown above. Since the focused *radiobutton* has been selected previously, any activation rule will *not* be triggered.

To achieve the former behavior, you can access the *native* Dialog Manager attribute, e.g. **`DLG SET "Radiobutton!focus",1`**