E l m S c r i p t

Version 7.0

E s l O b j e c t R e f e r e n c e

ELMSOFT INC.
7954 Helmart Drive
Laurel Maryland 20723
USA

Written and designed at Elmsoft, Inc., 7954 Helmart Drive, Laurel,MD 20723, USA

Table of Contents

Chapter 1

Introduction

EslObjects are references to some entity. The value is just an ID number that uniquely identifies that entity. Sometimes these are called handles or pointers. These entities can be various types of things. An EslObject might refer a set of functions (such as eStr or eUtl). It might refer to a database connection (EDB) or to a Form (EForm). Access to the actual object is via properties and methods (functions or subroutines). EslObjects can sometimes trigger events.

There are several different types of objects, permanent objects, script permanent objects, and volatile objects. Permanent objects are always present. They are created when ElmScript starts and have fixed names. They are available to all running scripts. Examples of these are eSys (the system object, formerly ESystem) and eUtl (a set of utility functions). Script permanent objects are also always present, but there is a separate object for each running script. An example of this is the eDebug object. Most of the EslObjects are volatile objects. To use them, you must first create an object (using the New command or one of the utiltity functions). When you are finished with the object, you should use the Delete Object command to remove it.

Since EslObject values are only handles to an object, not the object itself, there may be many copies of the handle, but only one copy of the object itself. When you create an object you usually put the object value into a variable. If you assign the value of that variable to another variable (or pass it to a subroutine or function) it makes a copy of the handle, not the object. Furthermore, if you delete one of the variables, it only deletes the handle, not the object.

The following provides a summary of the various EslObjects. Subsequent chapters will explain them in detail.

Object Type Summary

eSys

The **`eSys`** object is a permanent object and it represents a set of system functions and properties. Use this object to perform various system functions. The former name of this object was ESystem.

> **Note:** The old method will still work (New ESystem), but it will return the same object as eSys. For new scripts the preferred method is to use eSys instead of ESystem.

eStr

The eStr object is a set of String related functions.

EArray, EVector, EStackList, ECollection

These objects represent arrays and collections of other data types.

EDateTime

The EDateTime object represents a date and time value. You can use this object to get the current date and time or create a specific date time of your own choosing.

ETimeSpan

The ETimeSpan object represents a Time Span. You can use this object to measure a time length.

EFileInfo

The EFileInfo object represents a information about a selected file. You can use this object to access the various information in the file label. You can also use this object to get information about a group (selection) of files.

EslSession

The EslSession object represents the values that you see in the Options dialog box called from the user interface which are also in the fscript.ini file. This object contains a list of read-only properties that represent those values.

eDebug

The eDebug object allows you to perform a trace (Output to the console or to a file) of commands or function calls.

eUtl

The eUtl object is a set of utility functions. The evaluate function allows you to evaluate an expression and returns its value.

EDB, EQuery

These EslObjects are database related. The EDB refers to a database connection and the EQuery is a represents a results set of a SQL query.

EForm and Controls

The EForm represents a custom form or dialog box. The controls are the various controls that you find on a form, such as edit boxes, labels, checkboxes, etc. These also include menus and menu items.

EActiveXObject

The EActiveXObject represents a Com server object that currently resides somewhere on your system.

Chapter 2

eSys Object

The eSys object represents access to general operating system information and operations. The eSys object is a permanent object. It is always present so you do not have to create or delete it. This is a replacement for the old ESystem object.

Note: The old method will still work (New ESystem), but it will return the same object as eSys. For new scripts the preferred method is to use eSys instead of ESystem.

eSys Object Properties

Table 1: eSys Properties

Property Name	Data Type	Property Description
`WindowsDir`	`String`	The directory where MS Windows is installed.
`CurrDir`	`String`	The current directory.
`ErrorCode`	`Integer`	The error code for the last error..
`ErrorMsg`	`String`	The text of the last error.
`UserName`	`String`	The logged on User name.
`ComputerName`	`String`	The name of the computer.
`ScreenHeight`	`Integer`	The height of the screen.
`ScreenWidth`	`Integer`	The width of the screen.
`ScreenClientHeight`	`Integer`	The height of the client area of the screen.
`ScreenClientWidth`	`Integer`	The width of the client area of the screen.
`ScreenCaptionAreaHeight`	`Integer`	The height of the caption area of a form.
`DefaultPrinter`	`String`	The name of the default printer for the MS Windows session. This value can only be set correctly in Windows 2000 or Windows XP.
`PrinterList`	`StringList`	A list of all the printer names currently defined.
`CurrentPrinter`	`String`	The name of the current FrameMaker printer. You can set this value to change the current printer that FrameMaker is using. This is especially useful for saving files as postscript for later PDFproduction. *Requires FrameMaker 6.0 or greater*

Table 1: eSys Properties

Property Name	Data Type	Property Description
CurrPrintOrientation	**String**	The orientation of the current printer. *Requires FrameMaker 6.0 or greater*. Possible values are: 'Portrait' or 'Landscape'
CurrPrintQuality	**String**	The quality value of the current printer. *Requires FrameMaker 6.0 or greater*. Possible values are: `'High', 'Medium', 'Low', or 'Draft'.`
CurrPrintColor	**String**	The color type of the current printer. *Requires FrameMaker 6.0 or greater*. Possible values are: `'Color' or 'Monochrome'.`
CurrPrintNumCopies	**Integer**	The number of copies to print. *Requires FrameMaker 6.0 or greater*
CurrPrintPaperSource	**String**	The source of the paper for the current printer. *Requires FrameMaker 6.0 or greater*. Possible values are: `'Auto', 'Cassette', 'Envelope', 'EnvManual', 'FormSource', 'LargeCapacity', 'LargeFmt', 'Lower', 'Manual', 'Middle', 'OnlyOne', 'Tractor', 'SmallFmt'.`
CurrPrintPaperSize	**String**	The paper size of the current printer. *Requires FrameMaker 6.0 or greater*. Possible values are: `'Letter', 'Legal', '10X14', '11X17', '12X11', 'A3', 'A3Rotated', 'A4', 'A4Rotated', 'A4Small', 'A5', 'A5Rotated', 'A6', 'A6Rotated', 'B4', 'B4JisRotated', 'B5', 'B5JisRotated', 'B6','B6JisRotated', 'CSheet', 'DblJapanesePostcard', 'DblJapanesePostcardRotated', 'DSheet', 'Env9', 'Env10', 'Env11', 'Env12', 'Env14', 'EnvC5', 'EnvC3', 'EnvC4', 'EnvC6', 'EnvC65', 'EnvB4', 'EnvB5', 'EnvB6', 'EnvDL', 'EnvItaly', 'EnvMonarch', 'EnvPersonal', 'EnvESheet', 'Executive', 'FanfoldUS', 'FanfoldStdGerman', 'FanfoldLglGerman', 'Folio', 'JapanesePostcardRotated', 'JEnvChou3', 'JEnvChou3Rotated', 'JEnvChou4', 'JEnvChou4Rotated', 'JEnvKaku2', 'JEnvKaku2Rotated', 'JEnvKaku3', 'JEnvKaku3Rotated', 'JEnvYou4', 'JEnvYou4Rotated', 'Last', 'Ledger', 'LetterRotated', 'LetterSmall', 'Note', 'P16K', 'P16KRotated', 'P32K', 'P32KRotated', 'P32KBig', 'P32KBigRotated', 'PEnv1', 'PEnv1Rotated', 'PEnv2', 'PEnv2Rotated', 'PEnv3', 'PEnv3Rotated', 'PEnv4', 'PEnv4Rotated', 'PEnv5', 'PEnv5Rotated', 'PEnv6', 'PEnv6Rotated', 'PEnv7', 'PEnv7Rotated', 'PEnv8', 'PEnv8Rotated', 'PEnv9', 'PEnv9Rotated', 'PEnv10', 'PEnv10Rotated', 'Quarto', 'Statement', 'Tabloid'`

Example:
The following script displays some of the properties of the system.

```
. . .
Write Console 'The current directory is '+eSys.CurrDir;
Write Console 'The Windows directory is '+eSys.WindowsDir;
Write Console 'My User name is '+eSys.UserName;
Write Console 'My Computer name is '+eSys.ComputerName;
Write Console 'My Monitor size is '+
   eSys.ScreenWidth+' by '+eSys.ScreenHeight;
Write Console 'The Client area of my monitor is '+
   eSys.ScreenClientWidth+' by '+eSys.ScreenClientHeight;
. . .
```

Example:
The following script displays a list of printers and allows the user to choose a new default printer. This is for Windows 2000 or Windows XP.

```
. . .
DialogBox Type(ScrollBox) List(eSys.PrinterList)
         Caption('Select a Printer')
         NewVar(gvSelString) Button(gvBtnVar);

If gvBtnVar = OKBUTTON
   Set eSys.DefaultPrinter = gvSelString;
EndIf;
. . .
```

eSys Object Methods

Table 2: eSys Methods

Method Name	Method Description
Environment Methods	
GetEnvVar	Gets the value of an environment variable.
SetEnvVar	Sets the value of an environment variable.
ExpandEnvStrings	Writes a value to a registry key.
Directory Methods	
CreateDirectory	Creates a directory.
DeleteDirectory	Deletes a directory
File Methods	
FileExist	Determines if a file exists.
MoveFile	Moves a file from one location to another.
CopyFile	Copies a file from one location to another.
DeleteFile	Deletes a file.

Table 2: eSys Methods

Method Name	Method Description
RenameFile	Renames a file.
GetFullPathName	Gets the full path name for a partial file name.
ExtractPathName	Gets the path name from a full path name.
ExtractFileName	Gets the file name from a full path name.
ExtractFileExtension	Gets the file extension from a full path name.
RemoveFileExtension	Removes the extension from a full path name.
Registry Methods	
CreateRegistrySubKey	Creates a key.
GetRegistryKeyValue	Gets the value of a registry key.
GetAllRegistrySubKeys	Gets a list (StringList) of sub keys under the specified key.
GetAllRegistryKeyValueNames	Gets a list (StringList) of sub keys under the specified key.
SetRegistryKeyValue	Sets the value of a registry key. If the key does not exist, the key will be created.
DeleteRegistrySubKey	Deletes registry key.
DeleteRegistryValueName	Deletes a value of a registry key.
INI File Methods	
GetINIKeyValue	Gets the value of an INI file key.
GetAllINISections	Gets a list (StringList) of sections for the specified INI file.
GetAllINIKeys	Gets a list (StringList) of sub keys under the specified INI file section.
SetINIKeyValue	Sets the value of an INI key. If the key does not exist, the key will be created.
DeleteINIKey	Deletes a key from the specified INI file and Section.
DeleteINISection	Deletes a section from the specified INI file.
Font Methods	
GetFontCharsetList	Returns a list of CharSet values for the specified font name.
GetFontInfo	Returns a list of font information for the specified font name or for all installed fonts.
GetDefFontEncoding	Returns the name of the default font encoding.
SetDefFontEncoding	Sets the default font encoding to the specified value.
GetSupportedEncodings	Returns a list of the supported font encodings.
Miscellaneous Methods	
RunProgram	Runs an external program and waits for completion.
Idle	This method is used to allow other messages to be processed in the Windows session. It is a good idea to call this function occasionally when doing long operations, where you want Windows to respond to events (such as Dialog messages).

Environment methods

GetEnvVar

The GetEnvVar method gets the value of a system environment variable.

Format:

```
Run eSys.GetEnvVar EnvName(envVarName) NewVar(envVarValue);
```

Table 3: GetEnvVar Options

Option Name	Option Description
EnvName	The name of the environment variable.
NewVar	The name of the ElmScript variable to hold the value of the environment variable.

Example:

The following script gets the values of the Path, OS and TEMP system environment variables.

```
. . .
Run eSys.GetEnvVar EnvName('Path') NewVar(pathValue);
Run eSys.GetEnvVar EnvName('OS') NewVar(OsValue);
Run eSys.GetEnvVar EnvName('TEMP') NewVar(TempFilePath);
. . .
```

See also

"ExpandEnvStrings" on page 8

SetEnvVar

The SetEnvVar command sets the value of a system environment variable. NOTE: This sets the environment variable for the current process only. It does not permanently set this environment variable

Format:

```
Run eSys.SetEnvVar EnvName(envVarName) Value(envVarValue);
```

Table 4: SetEnvVar Options

Option Name	Option Description
EnvName	The name of the environment variable.
Value	The new value of the environment variable.

Example:

The following script sets the value of the MyVar system environment variable to 'MyValue'.

```
. . .
Run eSys.SetEnvVar EnvName('MyVar') Value('MyValue');
. . .
```

See also

"GetEnvVar" on page 7

ExpandEnvStrings

The ExpandEnvStrings command gets the value of a system environment variable.

Format:

```
Run eSys.ExpandEnvStrings Value(inputString) NewVar(expandedString);
```

Table 5: ExpandEnvStrings Options

Option Name	Option Description
Value	The input string.
NewVar	The name of the ElmScript variable to hold the value of the expanded string.

Example:

The following script expands the input string replacing the %MYDIR% with the value in the environment string defined as MYDIR. You would have previously defined the MYDIR environment variable (using the SET command in a batch file or updating the system environment setttings).

```
. . .
Set testInput = '%MYDIR%\MyFile.txt';
Run eSys.ExpandEnvStrings Value(testInput) NewVar(fileNameValue);
. . .
```

See also

"GetEnvVar" on page 7

Directory Methods

CreateDirectory

The CreateDirectory command creates a new directory (or folder) on a disk volume.

Format:

```
Run eSys.CreateDirectory DirName(dirName);
```

Table 6: CreateDirectory Options

Option Name	Option Description
DirName	The name of the directory or folder to create.

Example:

The following script creates a new directory under the Windows Temp directory.

```
. . .
Run eSys.GetEnvVar EnvName('TEMP') NewVar(TempFilePath);
Run eSys.CreateDirectory DirName(TempFilePath+'\MyTempDir');
. . .
```

DeleteDirectory

The DeleteDirectory command deletes a directory (or folder) from a disk volume. The directory must be empty.

Format:

```
Run eSys.DeleteDirectory DirName(dirName);
```

Table 7: DeleteDirectory Options

Option Name	Option Description
DirName	The name of the empty directory or folder to delete.

Example:

The following script deletes the directory created in the previous example.

```
. . .
Run eSys.GetEnvVar EnvName('TEMP') NewVar(TempFilePath);
Run eSys.DeleteDirectory DirName(TempFilePath+'\MyTempDir');
. . .
```

File Methods

MoveFile

The MoveFile command moves a file from one location to another.

Format:

```
Run eSys.MoveFile From(fromFileName) To(toFileName)
    [FilesOnly] [SimpleProgress] [NoConfirm]
    [NoConfirmDir] [NoErrorUI] [Silent] [ProgressTitle(title)];
```

Table 8: MoveFile Options

Option Name	Option Description
From	The name of the file to move.
To	The name of the destination file.
FilesOnly	Perform the operation only if a wildcard (*.*) is specified.
SimpleProgress	Displays a progress dialog, but does not show the file names.
NoConfirm	Respond with a Yes to All for any dialog box that is displayed.
NoConfirmDir	Does not confirm the creation of a new directory if the operation causes a directory to be created.
NoErrorUI	No user interface will be displayed if an error occurs.
Silent	Does not display a progress dialog box.
ProgressTitle	If the SimpleProgress option is chosen, you can provide a title for the dialog box, using this option.

Example:
The following script moves a file from the directory C:\MyDir to the directory C:\MyOtherDir.

```
. . .
Run eSys.MoveFile From('C:\MyDir\MyFile.dat') To('C:\MyOtherDir\MyFile.dat');
. . .
```

CopyFile

The CopyFile command copies a file from one location to another.

Format:

```
Run eSys.CopyFile From(fromFileName) To(toFileName)
    [FilesOnly] [SimpleProgress] [NoConfirm]
    [NoConfirmDir] [NoErrorUI] [Silent] [ProgressTitle(title)];
```

Table 9: CopyFile Options

Option Name	Option Description
From	The name of the file to copy.
To	The name of the destination file.
FilesOnly	Perform the operation only if a wildcard (*.*) is specified.
SimpleProgress	Displays a progress dialog, but does not show the file names.
NoConfirm	Respond with a Yes to All for any dialog box that is displayed.
NoConfirmDir	Does not confirm the creation of a new directory if the operation causes a directory to be created.
NoErrorUI	No user interface will be displayed if an error occurs.
Silent	Does not display a progress dialog box.
ProgressTitle	If the SimpleProgress option is chosen, you can provide a title for the dialog box, using this option.

Example:
The following script copies a file from the directory C:\MyDir to the directory C:\MyOtherDir, while showing a progress dialog box.

```
. . .
Run eSys.CopyFile From('C:\MyDir\MyFile.dat') To('C:\MyOtherDir\MyFile.dat')
       ProgressTitle('Copying My Files') SimpleProgress;
. . .
```

RenameFile

The RenameFile command renames a file.

Format:

```
Run eSys.RenameFile From(fromFileName) To(toFileName)
    [SimpleProgress] [NoConfirm]
    [NoConfirmDir] [NoErrorUI] [Silent] [ProgressTitle(title)];
```

Table 10: RenameFile Options

Option Name	Option Description
From	The initial name of the file.
To	The renamed name of the file.
SimpleProgress	Displays a progress dialog, but does not show the file names.
NoConfirm	Respond with a Yes to All for any dialog box that is displayed.
NoConfirmDir	Does not confirm the creation of a new directory if the operation causes a directory to be created.
NoErrorUI	No user interface will be displayed if an error occurs.
Silent	Does not display a progress dialog box.
ProgressTitle	If the SimpleProgress option is chosen, you can provide a title for the dialog box, using this option.

Example:
The following script renames a file called C:\MyDir\MyFile.dat to the name C:\MyDir\MyFileNew.dat.

```
. . .
Run eSys.RenameFile From('C:\MyDir\MyFile.dat') To('C:\MyDir\MyFileNew.dat');
. . .
```

DeleteFile

The DeleteFile command deletes a file.

Format:

```
Run eSys.DeleteFile FileName(FileName)
    [AllowUndo] [FilesOnly] [SimpleProgress] [NoConfirm]
    [NoConfirmDir] [NoErrorUI] [Silent] [ProgressTitle(title)];
```

Table 11: DeleteFile Options

Option Name	Option Description
FileName	The name of the file.
AllowUndo	Keep Undo information, if possible. If the file must be a fully qualified file name for this option to be effective. This option sends the file to the recycle bin.
FilesOnly	Perform the operation only if a wildcard (*.*) is specified.
SimpleProgress	Displays a progress dialog, but does not show the file names.
NoConfirm	Respond with a Yes to All for any dialog box that is displayed.
NoConfirmDir	Does not confirm the creation of a new directory if the operation causes a directory to be created.
NoErrorUI	No user interface will be displayed if an error occurs.
Silent	Does not display a progress dialog box.
ProgressTitle	If the SimpleProgress option is chosen, you can provide a title for the dialog box, using this option.

Example:
The following script deletes a file called C:\MyDir\MyFile.dat and sends it to the recycle bin.

```
. . .
Run eSys.DeleteFile FileName('C:\MyDir\MyFile.dat') AllowUndo;
. . .
```

See also
"DeleteDirectory" on page 9

FileExist

The FileExist command determines if the specified file exists.

Format:

```
Run eSys.FileExist FileName(FileName) NewVar(existVar);
```

Table 12: FileExist Options

Option Name	Option Description
FileName	The name of the file.
NewVar	The name of the ElmScript variable to hold the True or False value.

Example:
The following script checks a file called C:\MyDir\MyFile.dat to see if it exists.

```
. . .
Run eSys.FileExist FileName('C:\MyDir\MyFile.dat') NewVar(statVar);
If statVar
   MSGBOX 'The file, C:\MyDir\MyFile.dat, Exists!!';
Else
   MSGBOX 'The file, C:\MyDir\MyFile.dat, DOES NOT Exist!!';
EndIf
. . .
```

See also
"DeleteDirectory" on page 9

GetFullPathName

The GetFullPathName command returns a full path name for a file name.

Format:

```
Run eSys.GetFullPathName FileName(FileName) NewVar(fullFileName);
```

Table 13: GetFullPathName Options

Option Name	Option Description
FileName	The name of the file.
NewVar	The name of the ElmScript variable to hold the full path name of the file.

Example:
The following script returns the full name of the file (’MyFile.dat’).

```
. . .
Run eSys.GetFullPathName FileName('MyFile.dat') NewVar(fullFileName);
. . .
```

ExtractPathName

The ExtractPathName method returns the path part of the full path name.

Format:

```
Run eSys.ExtractPathName FullPath(FileName) NewVar(Pathname);
```

Table 14: ExtractPathName Options

Option Name	Option Description
FullPath	The full path name of the file.
NewVar	The path part of the file name.

Example:
The following script returns the path part of the file name. The `pathName` variable will contain ’C:\MyDir’ after the command runs.

```
. . .
Run eSys.ExtractPathName FullPath('c:\MyDir\MyFile.dat') NewVar(pathName);
. . .
```

ExtractFileName

The ExtractFileName method returns the file part of the full path name.

Format:

```
Run eSys.ExtractFileName FullPath(FileName) NewVar(Filename);
```

Table 15: ExtractFileName Options

Option Name	Option Description
FullPath	The full path name of the file.
NewVar	The filename part of the full path name.

Example:
The following script returns the path part of the file name. The `fileName` variable will contain ’`MyFile.dat`’ after the command runs.

```
. . .
Run eSys.ExtractFileName FullPath('c:\MyDir\MyFile.dat') NewVar(fileName);
. . .
```

ExtractFileExtension

The ExtractFileExtension method returns the extension part of the full path name.

Format:

```
Run eSys.ExtractFileExtension FullPath(FileName) NewVar(ExtName);
```

Table 16: ExtractFileExtension Options

Option Name	Option Description
FullPath	The full path name of the file.
NewVar	The extension part of the full path name.

Example:
The following script returns the extension part of thefull file name. The `extName` variable will contain '`dat`' after the command runs.

```
. . .
Run eSys.ExtractFileExtension FullPath('c:\MyDir\MyFile.dat') NewVar(extName);
. . .
```

RemoveFileExtension

The RemoveFileExtension method returns the evertyhing except the extension part of the full path name.

Format:

```
Run eSys.RemoveFileExtension FullPath(FileName) [NewExt(ext)] NewVar(Filename);
```

Table 17: RemoveFileExtension Options

Option Name	Option Description
FullPath	The full path name of the file.
NewExt	The name of the replacement extension. This is optional. If not specified, then the returned value will have no extension.
NewVar	The full path name minus the extension, but optionally with a replacement file extension.

Example 1:
The following script returns the full path name minus the extension. The `fileName` variable will contain '`c:\MyDir\MyFile`' after the command runs.

```
. . .
Run eSys.RemoveFileExtension FullPath('c:\MyDir\MyFile.dat') NewVar(fileName);
. . .
```

Example 2:
The following script returns the full path name replacing the extension with '.new'. The `fileName` variable will contain '`c:\MyDir\MyFile.new`' after the command runs.

```
. . .
Run eSys.RemoveFileExtension FullPath('c:\MyDir\MyFile.dat')
   NewExt('new') NewVar(fileName);
. . .
```

Registry Methods

The system registry is a database which MS Windows (95 and older) maintains information about the operating system and other programs. This database is an essential part of Windows. Care should be taken when trying to update parts of this database.

The registry also contains information about installed programs. Applications such as Microsoft Word and FrameMaker itself stores information in the registry that it wants to preserve from session to session. For example, most programs store their program options in the registry. ElmScript itself uses the registry to store the position of the dialog windows and keep a list of the most recently used scripts.

The registry is a hierarchical (tree) structure similar (in spirit) to the directory structure on a hard disk. Instead of directories, however, the nodes are called Keys. These keys can have one or more values (identified by a value name) as well as zero or more subkeys associated with them. The top of the hierarchy is divided into four Groups, 'Classes', 'CurrentUser', 'LocalMachine', and 'Users' (their names in the registry are HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE ane HKEY_USERS, respectively). All the other keys fall into one of these groups. The 'Classes' group is where Windows stores a variety of information about file associations and OLE class Ids. The 'CurrentUser' group, as its name implies, contains information about the currently logged on user and the 'LocalMachine' group, likewise, contains information about the entire machine, such as system settings and device drivers. The 'Users' group has the settings for new users.

For ElmScript script writers, we also have a fifth group called 'Esl', where scriptwriters can get and save their own information. This 'Esl' group is actually an alias for a key under the 'CurrentUser' group (CurrentUser\Software\Esl\RegistryGroup). Although you can use the following commands to read the value of any key in the registry, you can only update, delete or create keys in the 'Esl' group.

You can use the registry to save information that you can retrieve the next time you run the script.

> **IMPORTANT:** You should decide how you want to organize this 'Esl' group area. If you are writing scripts for yourself and only your scripts run on your machine, then it does not matter so much. But if you are using scripts that others write or if you are part of a large organization where scripts may come from various sources, then you should be aware that other scripts might also save information under the 'Esl' group in the registry. You might want to use a structure that will not interfere (or conflict) with other scripts, for example, you might want to use your own name (or organization name) as the top level key, then perhaps the script name as the second level key. This way you can your information for each of your scripts separate from others.

CreateRegistrySubKey

The CreateRegistrySubKey command creates a sub key under the specified key.

Format:

```
Run eSys.CreateRegistrySubKey [Group(groupName)] [Key(keyname)] SubKey(subkeyname);
```

Table 18: CreateRegistrySubKey Options

Option Name	Option Description
Group	The name of the Group. If not specified, the 'Esl' is assumed.
Key	The name of the key. Each subkey is separated by a backslash (\). This is optional. If not specified, the sub key is created just under the group.
SubKey	The name of the sub key to create.

Example:
The following script creates the subkey 'MyScriptStuff' under the ’Esl’ group then creates a subkey 'MyStuff' under that key.

```
. . .
Run eSys.CreateRegistrySubKey Group('Esl') SubKey('MyScriptStuff');
Run eSys.CreateRegistrySubKey Group('Esl') Key('MyScriptStuff')
   SubKey('MyStuff');
. . .
```

GetRegistryKeyValue

The GetRegistryKeyValue command returns a value of the specified key and value name.

Format:

```
Run eSys.GetRegistryKeyValue [Group(groupName)] [Key(keyname)] ValueName(valname)
   NewVar(keyValue);
```

Table 19: GetRegistryKeyValue Options

Option Name	Option Description
Group	The name of the Group. If not specified, the ’Esl’ is assumed.
Key	The name of the key. Each subkey is separated by a backslash (\). This is optional. If not specified, the key used is the group.
ValueName	The name of the value.
NewVar	The name of the ElmScript variable to hold the key value.

Example:
The following script gets the value of the key called `'JohnDoe\ThisScript\MyKey'` under the ’Esl’ group and also gets the directory of FrameMaker version 6.0 from the ’LocalMachine’ group.

```
. . .
Run eSys.GetRegistryKeyValue Group('Esl') Key('JohnDoe\ThisScript\MyKey')
   ValueName('MyValue') NewVar(keyValue);
Run eSys.GetRegistryKeyValue Group('LocalMachine')
   Key('SOFTWARE\Adobe\FrameMaker\6.0') ValueName('FMHome') NewVar(fmHomeDir);
. . .
```

See also
“GetINIKeyValue” on page 19

GetAllRegistrySubKeys

The GetAllRegistrySubKeys command returns a list of subkeys for the specified key.

Format:

```
Run eSys.GetAllRegistrySubKeys [Group(groupName)] [Key(keyname)] NewVar(keyList);
```

Table 20: GetAllRegistrySubKeys Options

Option Name	Option Description
Group	The name of the Group. If not specified, the 'Esl' is assumed.
Key	The name of the key. Each subkey is separated by a backslash (\). This is optional. If not specified, the key used is the group.
NewVar	The name of the ElmScript variable to hold the list of sub keys.

Example:
The following script returns gets all the sub keys of the key called `'JohnDoe\ThisScript'` under the 'Esl' group.

```
. . .
Run eSys.GetAllRegistrySubKeys Group('Esl') Key('JohnDoe\ThisScript')
   NewVar(keyList);
. . .
```

See also
"GetAllINIKeys" on page 20

GetAllRegistryKeyValueNames

The GetAllRegistryKeyValueNames command returns a list of value names for the specified key.

Format:

```
Run eSys.GetAllRegistryKeyValueNames [Group(groupName)] [Key(keyname)]
   NewVar(valueNameList);
```

Table 21: GetAllRegistrySubKeys Options

Option Name	Option Description
Group	The name of the Group. If not specified, the 'Esl' is assumed.
Key	The name of the key. Each subkey is separated by a backslash (\). This is optional. If not specified, the key used is the group.
NewVar	The name of the ElmScript variable to hold the list of value names.

Example:
The following script returns gets all the sub keys of the key called `'JohnDoe\ThisScript'` under the 'Esl' group.

```
. . .
Run eSys.GetAllRegistrySubKeys Group('Esl') Key('JohnDoe\ThisScript')
   NewVar(keyList);
. . .
```

See also
"GetAllINIKeys" on page 20

SetRegistryKeyValue

The SetRegistryKeyValue command returns the value of the specified key.

Format:

```
Run eSys.SetRegistryKeyValue [Group(groupName)] Key(keyname) ValueName(valName)
   Value(keyValue);
```

Table 22: SetRegistryKeyValue Options

Option Name	Option Description
Group	The name of the Group. This must be 'Esl' or left out completely for this method.
Key	The name of the key. Each subkey is separated by a backslash (\). This is optional. If not specified, the key used is the group.
ValueName	The name of the value.
Value	The new key value to store with the specified key.

Example:

The following script returns sets the value of 'MyValue' of the key called `'JohnDoe\ThisScript\MyKey'` under the 'Esl' group to 'XXXXXX'.

```
. . .
Run eSys.SetRegistryKeyValue Group('Esl') Key('JohnDoe\ThisScript\MyKey')
   ValueName('MyValue') Value('XXXXXX');
. . .
```

DeleteRegistrySubKey

The DeleteRegistrySubKey command deletes the specified sub key.

Format:

```
Run eSys.DeleteRegistrySubKey [Group(groupName)] [Key(keyname)] SubKey(subKeyName);
```

Table 23: DeleteRegistrySubKey Options

Option Name	Option Description
Group	The name of the Group. This must be 'Esl' or left out completely for this method.
Key	The name of the key to delete. Each subkey is separated by a backslash (\). This is optional. If not specified, the key used is the group.
SubKey	The name of the sub key to delete. .

Example:

The following script deletes the key called `'JohnDoe\ThisScript\MyKey'` under the 'Esl' group.

```
. . .
Run eSys.DeleteRegistrySubKey Group('Esl') Key('JohnDoe\ThisScript\MyKey');
. . .
```

DeleteRegistryValueName

The DeleteRegistryValueName command deletes the value name of the specified key.

Format:

```
Run eSys.DeleteRegistryValueName [Group(groupName)] [Key(keyname)]
    ValueName(Valname);
```

Table 24: DeleteRegistryValueName Options

Option Name	Option Description
Group	The name of the Group. This must be 'Esl' or left out completely for this method.
Key	The name of the key. Each subkey is separated by a backslash (\). This is optional. If not specified, the key used is the group.
ValueName	The value name of the value to delete.

Example:
The following script deletes the value name 'MyValue' from the key called `'JohnDoe\ThisScript\MyKey'` under the 'Esl' group.

```
. . .
Run eSys.DeleteRegistryValueName Group('Esl') Key('JohnDoe\ThisScript\MyKey')
    ValueName('MyValue');
. . .
```

INI File Methods

Similar to the registry, INI (initialization) files are used for saving information from one program session to another. Instead of one large database, each ini is a separate file. FrameMaker itself uses an ini file to store its program options. An INI file consists of one or more sections, which, in turn, contain one or more keys.

GetINIKeyValue

The GetINIKeyValue command returns a value of the specified INI file, Section and Key name.

Format:

```
Run eSys.GetINIKeyValue File(iniFileName) Section(sectionName) Key(keyName)
    [Default(defaultValue)] NewVar(keyValue);
```

Table 25: GetINIKeyValue Options

Option Name	Option Description
File	The file name of the INI file.
Section	The name of the section.
Key	The name of the key.
Default	The value to return, if the key is not found.
NewVar	The name of the ElmScript variable to hold the key value.

Example:
The following script returns gets the value of the key called 'LastFile' in the 'File' section in the INI file named 'C:\MyScripts\MyFile.ini'.

```
. . .
Run eSys.GetINIKeyValue File('C:\MyScripts\MyFile.ini') Section('File')
   Key('LastFile') Default('') NewVar(lastFileSaved);
. . .
```

See also
"CreateRegistrySubKey" on page 15

GetAllINISections

The GetAllINISections command returns a list of all keys in the specified INI file and Section.

Format:

```
Run eSys.GetAllINISections File(iniFileName) NewVar(sectList);
```

Table 26: GetAllINISections Options

Option Name	Option Description
File	The file name of the INI file.
NewVar	The name of the ElmScript variable to hold the list of sections.

Example:
The following script returns gets a list of all the section names in the INI file named 'C:\MyScripts\MyFile.ini'.

```
. . .
Run eSys.GetAllINISections File('C:\MyScripts\MyFile.ini')
   NewVar(allSections);
. . .
```

GetAllINIKeys

The GetAllINIKeys command returns a list of all keys in the specified section.

Format:

```
Run eSys.GetAllINIKeys File(iniFileName) Section(sectionName) NewVar(keyList);
```

Table 27: GetAllINIKeys Options

Option Name	Option Description
File	The file name of the INI file.
Section	The name of the section.
NewVar	The name of the ElmScript variable to hold the list of keys.

Example:
The following script returns gets a list of all the keys `in the 'File' section in the INI file named 'C:\MyScripts\MyFile.ini'.`

```
. . .
Run eSys.GetAllINIKeys File('C:\MyScripts\MyFile.ini') Section('File')
   NewVar(fileKeys);
. . .
```

SetINIKeyValue

The SetINIKeyValue command sets the value of the specified INI file, Section and Key name.

Format:

```
Run eSys.SetINIKeyValue File(iniFileName) Section(sectionName) Key(keyName)
   Value(keyValue);
```

Table 28: SetINIKeyValue Options

Option Name	Option Description
File	The file name of the INI file.
Section	The name of the section.
Key	The name of the key.
Value	The key value.

Example:
The following script returns sets the value of the key called `'LastFile' in the 'File' section in the INI file named 'C:\MyScripts\MyFile.ini' to 'C:\MyScripts\MyLastFilename.txt'.`

```
. . .
Run eSys.SetINIKeyValue File('C:\MyScripts\MyFile.ini') Section('File')
   Key('LastFile') Value('C:\MyScripts\MyLastFilename.txt');
. . .
```

DeleteINIKey

The DeleteINIKey command deletes a key from the specified INI file and Section.

Format:

```
Run eSys.DeleteINIKey File(iniFileName) Section(sectionName) Key(keyName);
```

Table 29: DeleteINIKey Options

Option Name	Option Description
File	The file name of the INI file.
Section	The name of the section.
Key	The name of the key to delete.

Example:
The following script returns deletes the key called `'LastFile'` from the `'File'` section in the INI file named `'C:\MyScripts\MyFile.ini'`.

```
. . .
Run eSys.DeleteINIKey File('C:\MyScripts\MyFile.ini') Section('File')
   Key('LastFile');
. . .
```

DeleteINISection

The DeleteINISection command deletes a section from the specified INI file.

Format:

```
Run eSys.DeleteINISection File(iniFileName) Section(sectionName);
```

Table 30: DeleteINISection Options

Option Name	Option Description
File	The file name of the INI file.
Section	The name of the section.

Example:
The following script returns deletes the entire `'File'` section from the INI file named `'C:\MyScripts\MyFile.ini'`.

```
. . .
Run eSys.DeleteINIKey File('C:\MyScripts\MyFile.ini') Section('File');
. . .
```

Font Methods

GetFontCharsetList

The GetFontCharsetList method returns a list of Charset names for the specified font.

Format:

```
Run eSys.GetFontCharsetList FontName(fontNameString)
   NewVar(charsetList);
or
Set charsetList = eSys.GetFontCharsetList{fontNameString};
```

Table 31: GetFontCharsetList Options

Option Name	Option Description
fontNameString	The name of the font.
charsetList	A StringList containing a name for each charset for the font.

Example:
The following script gets the list of Charsets from the Times New Roman font and writes them to the FrameMaker console.

```
. . .
Run eSys.GetFontCharsetList FontName('Times New Roman') NewVar(vCharsetList);
Write Console 'Charset information for Times New Roman';
Write Console vCharsetList;
. . .
The output should be something like the following:
Charset information for Times New Roman
STRINGLIST Count(9)
ANSI
HEBREW
ARABIC
GREEK
TURKISH
BALTIC
EASTEUROPE
RUSSIAN
VIETNAMESE

. . .
```

GetFontInfo

The GetFontInfo method returns a list of information for the specified fonts.

Format:

```
Run eSys.GetFontInfo [FontName(fontNameString)]
   NewVar(fontList);
or
   Set fontList = eSys.GetFontInfo{fontNameString};
```

Table 32: GetFontInfo Options

Option Name	Option Description
`fontNameString`	The name of the font (Optional). If not specified, then information about all the fonts are returned.
`fontList`	A StringList containing a information about the fonts specified.

Example:
The following script gets information about the Times New Roman font and writes it to the FrameMaker console.

```
. . .
Run eSys.GetFontInfo FontName('Times New Roman') NewVar(vFontList);
Write Console 'Font information for Times New Roman';
Write Console vFontList;
. . .
The output should be something like the following:
Font information for Times New Roman
STRINGLIST Count(34)
FontName=Times New Roman;CharSet=ANSI;Italic=False;Weight=NORMAL;Height=35;Width=13
FontName=Times New Roman;CharSet=HEBREW;Italic=False;Weight=NORMAL;Height=35;Width=13
FontName=Times New Roman;CharSet=ARABIC;Italic=False;Weight=NORMAL;Height=35;Width=13
FontName=Times New Roman;CharSet=GREEK;Italic=False;Weight=NORMAL;Height=35;Width=13
FontName=Times New Roman;CharSet=TURKISH;Italic=False;Weight=NORMAL;Height=35;Width=13
FontName=Times New Roman;CharSet=BALTIC;Italic=False;Weight=NORMAL;Height=35;Width=13
. . .
FontName=Times New Roman;CharSet=RUSSIAN;Italic=True;Weight=NORMAL;Height=35;Width=13
FontName=Times New Roman;CharSet=VIETNAMESE;Italic=True;Weight=NORMAL;Height=35;Width=13
. . .
```

Default Font Encoding

Default font encoding is used primarily for documents with Asian text. The standard value for non-Asian FrameMaker installations is 'FrameRoman'. Other values are described in the Font encoding table, see “Font Encoding Values” on page 25. The initial value is set when FrameMaker starts. If your documents contain Asian text and you are using a non-Asian version of FrameMaker, you should set the Default Font Encoding to the appropriate value in your script. This will ensure that the Find commands work correctly with double byte characters.

GetDefFontEncoding

The GetDefFontEncoding method returns the name of the current font encoding.

Format:

```
    Run eSys.GetDefFontEncoding NewVar(encNameVar);
or
    Set encNameVar = eSys.GetDefFontEncoding{};
```

Table 33: GetDefFontEncoding Options

Option Name	Option Description
`encNameString`	The name of the current default font encoding.

Example:
The following script gets the name of the default encoding and writes it to the console.

```
. . .
Set gvEncName = eSys.GetDefFontEncoding{};
Write Console 'Current Encoding is '+gvEncName;
Write Console vCharsetList;
. . .
The output should be something like the following:
Current Encoding is FrameRoman
. . .
```

Table 34: Font Encoding Values

Value:	Description
`FrameRoman`	Standard Roman text (default value for non-Asian installations)
`JISX0208.ShiftJIS (or Japanese)`	Japanese Text
`BIG5 (or TradChinese)`	Traditional Chinese
`GB2312-80.EUC (or SimpChinese)`	Simplified Chinese
`KSC5601-1992 (or Korean)`	Korean
`UTF-8`	UTF-8 unicode. *FrameMaker 8 or Greater*

SetDefFontEncoding

The SetDefFontEncoding method sets the current default font encoding to the specified value. The value must be one of the members in the Font Encoding Table (Table 34 on page 25). You may use the actual value or the descriptive name. For example, to set for Japanese encoding, you could use the encoding name (`'JISX0208.ShiftJIS'`) or the descriptive name (`'Japanese'`). The previous encoding name is returned.

IMPORTANT: The SetDefFontEncoding method sets the font encoding for the entire FrameMaker session, not just the current script. Setting this value will affect any script that is running or will subsequently run. The setting will remain in effect until it is changed by this or another script or by using the Options dialog. For FrameMaker 8, this method also determines the encoding of text returned from or sent to FrameMaker documents. If you set the encoding to UTF-8, then all strings will be returned in the UTF-8 encoding. This is known as unicode mode.

It is recommended that you choose one setting and do not change it. If you have strings with different encodings, then the compare methods and commands will not work correctly.

Format:

```
Run eSys.SetDefFontEncoding EncName(newEncNameString)
   NewVar(prevEncNameString);
or
Set prevEncNameString = eSys.SetDefFontEncoding{newEncNameString};
```

Table 35: SetDefFontEncoding Options

Option Name	Option Description
newEncNameString	The new encoding name.
prevEncNameString	The previous encoding name.

Example:
The following script sets the encoding to simplified chinese then searches the current paragraph for text contained in the variable (gvText). It then write a message to the console, saying whether it was found or not. It also writes the encoding information the console.

```
. . .
Set gvPrevFontEnc=eSys.SetDefFontEncoding{'SimpChinese'};
Write Console 'Old Enc-'+gvPrevFontEnc;
Write Console 'New Enc-'+eSys.GetDefFontEncoding{};
Write Console 'Supported Enc--'+eSys.GetSupportedEncodings{};
Write Console '';
Find String(gvText) InObject(CurrentPgf) ReturnRange(gvRange) ReturnStatus(gvFound);
If gvFound
   Write Console 'Text was Found'
Else
   Write Console 'Text was not Found'
EndIf;
. . .
The output should be something like the following:
Old Enc-FrameRoman
New Enc-GB2312-80.EUC
Supported Enc--STRINGLIST Count(5)
FrameRoman
JISX0208.ShiftJIS
BIG5
GB2312-80.EUC
KSC5601-1992

Text was Found
. . .
```

GetSupportedEncodings

The GetSupportedEncodings method returns a list of the supported font encoding names.

Format:

```
   Run eSys.GetSupportedEncodings NewVar(encNameList);
or
   Set encNameList = eSys.GetSupportedEncodings{};
```

Table 36: GetSupportedEncodings Options

Option Name	Option Description
encNameList	A StringList containing the returned encoding names.

Example:
The following script gets the name of the default encoding and writes it to the console.

```
. . .
Write Console 'Supported Enc--'+eSys.GetSupportedEncodings{};
. . .
The output should be something like the following:
Supported Enc--STRINGLIST Count(5)
FrameRoman
JISX0208.ShiftJIS
BIG5
GB2312-80.EUC
KSC5601-1992
. . .
```

Miscellaneous Methods

RunProgram

The RunProgram method runs an external program and optionally waits for it to complete.

Format:

```
    Run eSys.RunProgram CommandLine(commandLineString)
        [Wait(timeToWait)] [NewVar(exitCode)]
or
    Set exitCode = eSys.RunProgram{commandLineString[,timeToWait]};
```

Table 37: RunProgram Options

Option Name	Option Description
`commandLineString`	The text of the command line including any options desired.
`timeToWait`	The number of milliseconds to wait. If this is not specified, then this method waits until the external program completes. If this value is 0, then it returns immediately. If you use an integer value here, it will wait for the specified number of milliseconds unless the program terminates first.
`exitCode`	A integer exit code returned from the program. Not all programs return an exit code and some return only zero regardless of whether the program ran correctly or not. In a standard Dos program, the exit code appears in the exit(code) function.

Example 1:
The following script starts the windows notepad program. If it has not terminated in 100 seconds (100000 milliseconds) the command will return and the script will continue.

```
. . .
Run eSys.RunProgram CommandLine('notepad.exe MyTestfile.txt') Wait(100000);
. . .
```

Example 2:
The following script starts a windows program and returns the exit code.

```
. . .
Set vRc = eSys.RunProgram{'MyProg.exe'};
. . .
. . .
. . .
```

The following is a shell of a C program that sets an exit code.

```
/* This program is compiled and linked to MyProg.exe */
main()
{
   int exitCode = {Do some computation here}
   exit(exitCode);
   return 0;
}
```

Idle

The Idle method allows windows to process other messages. This is good to call when doing long operations.

Format:

```
Run eSys.Idle;
```

Example:
The following script is an example of displaying a progress dialog box. The DisplayWaitForm function displays a Form with a progress bar and returns the form object. The Loop has the eSys.Idle method run each time the loop iterates. The information on the form is updated with a message and the progress bar value is updated with each iteration. Script command(s) that do the actual work should be placed where the DO SOMETHING HERE line is. When the loop finishes, the label on the button on the form is changed from 'Cancel' to 'Done' and the Label's caption is changed to **'Finished!'** or **'Canceled!'** depending on whether the loop finished normally or if the user pressed the cancel button.

```
Set gvLimit = 1000;

Set gvForm = DisplayWaitForm{'Test Wait Form',
                             'Please Wait...',
                             gvLimit,
                             'AbortRun'};

Set gvAbortFlag = False;
Loop While(i<gvLimit) LoopVar(i) InitVal(1) Incr(1)
   Run eSys.Idle;
   If gvAbortFlag
      LeaveLoop;
   EndIf
   // DO SOMETHING HERE
   Set gvRemaining = gvLimit - i;
   Set gvForm['WaitLabel'].Caption = 'Please wait '+gvRemaining+' seconds ...';
   Set gvForm['WaitBar'].CurrentValue = i;
EndLoop
```

```
Set gvForm['WaitCancelBtn'].Caption = 'Done';
If gvAbortFlag
   Write console 'Aborted!!';
   Set gvForm['WaitLabel'].Caption = 'Canceled!';
Else
   Write console 'Finished!!';
   Set gvForm['WaitLabel'].Caption = 'Finished!';
EndIf
Exec Wait Seconds(2);

Delete Object(gvForm);
Set gvForm = NULL;
```

```
Sub AbortRun

   Set gvAbortFlag = True;
   Write console 'In Abort Sub';

EndSub


//----------------------
Function DisplayWaitForm pvCaption pvLabel pvLimit pvCancelSubName

   Local lvLabel(pvLabel) lvCaption(pvCaption) lvLimit(pvLimit);

   If lvCaption = NULL
      Set lvCaption = 'Processing...';
   EndIf
   If lvLabel = NULL
      Set lvLabel = 'Processing...';
   EndIf
   If lvLimit = NULL
      Set lvLimit = 100;
   EndIf
   New EForm NewVar(Result) Caption(lvCaption)
      Top(eSys.ScreenHeight/3) Left((eSys.ScreenWidth-300)/2)
      Width(300) Height(140);
   New ELabel Form(Result) Name('WaitLabel')
      Caption(lvLabel) Top(12) Left(10);
   New EProgressBar Form(Result) Name('WaitBar')
      Top(37) Left(10) Width(275) Height(21)
      MinValue(0) CurrentValue(0) MaxValue(lvLimit);
   New EButton Form(Result) OnClick('AbortRun')
      Name('WaitCancelBtn') Caption('Cancel')
      Top(80) Left(140);
   Set Result.FirstFocus = 'WaitCancelBtn';
   Run Result.ShowModeless;
EndFunc

. . .
```

Chapter 3

String Functions Object(eStr)

The eStr object represents a set of string oriented functions.

Note: The methods (functions) of the eStr object are accessed the same way you access methods of any other object, by using the name followed by the dot (.) operator. The eStr methods can be accessed in a alternate manner. They can be accessed as methods of the string value itself. The first parameter of the eStr functions is always the source string. Use these methods as methods of the source string for a shortcut. Assuming that the variable gvMySourceString has a string value, the following two lines are equivalent.

```
Set gvStrVal = eStr.ToUpperCase{gvMySourceString};
Set gvStrVal = gvMySourceString.ToUpperCase{};
```

eStr Object Methods

Table 38: eStr Methods

Method Name	Method Description
Compare Methods	
Equal	Compares the two strings for equality. This method returns True if the strings are equal and False if not. You can optionally include a character count and 'NoCase' string to do a case insensitive comparison.
Compare	Compares the two strings for equality. This method returns 0 if the strings are equal , 1 if the first string is greater than the second and -1 if the second string is greater than the first. You can optionally include a character count and 'NoCase' string to do a case insensitive comparison.
IsPrefix	Compares the second string to determine if it is a prefix of the first string (for equality). This method returns True if the second string is a prefix of the first string and false otherwise. You can optionally include a 'NoCase' string to do a case insensitive comparison.
IsSuffix	Compares the second string to determine if it is a suffix of the first string (for equality). This method returns True if the second string is a suffix of the first string and false otherwise. You can optionally include a 'NoCase' string to do a case insensitive comparison.
IsValidUTF8	Returns True if the string is a valid UTF-8 sequence of bytes. *FrameMaker 8 or greater*.
Location Methods	
FindString	Finds a string within a string. This method looks for an occurrence of the second string within the first string and returns the character position, if found, and it returns 0 if not found. You can optionally include a 'NoCase' string to do a case insensitive comparison, a 'Back' option for a search starting at the end of the string and a 'Word' option to find only whole words.

Table 38: eStr Methods

Method Name	Method Description
SubString	Returns a substring of the original string from a range of character positions.
Character Replacement Methods	
ToUpperCase	Returns the string with all of its characters converted to upper case.
ToLowerCase	Returns the string with all of its characters converted to lower case.
Reverse	Returns the string with all of its characters reversed.
ReplaceAll	Replaces all occurrences in the first string of the second string with the third string.
ReplaceFirst	Replaces the first occurrence in the first string of the second string with the third string.
ReplaceLast	Replaces the last occurrence in the first string of the second string with the third string.
PlatformToFrame	Returns a string with all the characters converted from the platform character set to the Frame character set. This function is not valid when you set the `PlatformEncodingMode` = `True`.
FrameToPlatform	Returns a string with all the characters converted from the Frame character set to the platform character set. This function is not valid when you set the `PlatformEncodingMode` = `True`.
ConvertText	Returns a string with all the characters converted from one specified encoding to another specified encoding. *FrameMaker 8 or greater*.
Insert and Remove Character Methods	
InsertString	Returns a new string with the character string inserted at the specified position.
RemoveChars	Returns a new string with the specified characters removed.
RemoveLeading	Returns a new string with the specified characters removed from the beginning of the string.
RemoveTrailing	Returns a new string with the specified characters removed from the end of the string.
Miscellaneous Methods	
CharCount	Returns the number of characters in the string.
LoadFromTextFile	Returns a string with the contents of a text file.
SaveToTextFile	Writes the value of the string to a text file and returns the length of the string.
ToStringList	Converts a string into a string list.

Compare Functions

Equal{}

The Equal method is a function that returns a integer value which is either True or False.

Format:

```
Set boolVal = eStr.Equal{string, cmpString [,count][,'NoCase']};
```

Table 39: Equal{} Options

Option Name	Option Description
string	The text of the first string to compare.
cmpString	The text of the second string to compare.
count	An optional count of the number of characters to compare. If not specified then the entire string is compared.
'NoCase'	An optional string specifying that the comparison is case insensitive.
boolVal	The return value from the function. True indicates that the strings are equal. False otherwise.

Example:
The following script illustrates the Equal function. The first If command should be False, since, although the strings have the same characters, some have different cases. The second test should be True, because the 'NoCase' option specifies a case insensitive comparison.

```
. . .
If eStr.Equal{'mystring','MyString'}
   Display 'The Strings are Equal';
Else
   Display 'The Strings are not Equal';
EndIf
If eStr.Equal{'mystring','MyString','NoCase'}
   Display 'The Strings are Equal';
Else
   Display 'The Strings are not Equal';
EndIf
. . .
```

Compare{}

The Compare function compares two strings and returns a value indicating the comparitive lexigraphical order of the strings.

Format:

```
Set intVal = eStr.Compare{string, cmpString [,count][,'NoCase']};
```

Table 40: Compare{} Options

Option Name	Option Description
string	The text of the first string to compare.
cmpString	The text of the second string to compare.
count	An optional count of the number of characters to compare. If not specified then the entire string is compared.
'NoCase'	An optional string specifying that the comparison is case insensitive.
intVal	The return value of the function: 0 if the strings are equal , 1 if the first string is greater than the second and -1 if the second string is greater than the first.

Example:
The following script compares two strings and displays a message based on the lexigraphical order.

```
. . .
Set cmpVal = eStr.Compare{'abc','abd'};
If cmpVal < 0
   Display 'The first string is less than the second string';
Else
   Display 'The first string is greater than or equal to the second string';
EndIf
. . .
```

IsPrefix{}

The IsPrefix function returns True if second string is a prefix of the first string.

Format:

```
Set BoolVal = eStr.IsPrefix{string, prefString[,'NoCase']};
```

Table 41: IsPrefix{} Options

Option Name	Option Description
`string`	The text of the first string to compare.
`prefString`	The text of the prefix string to test.
`'NoCase'`	An optional string specifying that the comparison is case insensitive.
`BoolVal`	The return value from the function. True indicates that the second string is a prefix of the first string. It is False otherwise.

Example:
The following script should display the string in the True case, since the second string is a prefix of the first.

```
. . .
If eStr.IsPrefix{'MyString','MyStr'}
   Display 'The second string is a prefix';
Else
   Display 'The second string is not a prefix';
EndIf
. . .
```

IsSuffix{}

The IsSuffix function returns True if second string is a suffix of the first string.

Format:

```
Set BoolVal = eStr.IsSuffix{string, suffString[,'NoCase']};
```

Table 42: IsSuffix{} Options

Option Name	Option Description
string	The text of the first string to compare.
suffString	The text of the suffix string to test.
'NoCase'	An optional string specifying that the comparison is case insensitive.
BoolVal	The return value from the function. True indicates that the second string is a suffix of the first string. It is False otherwise.

Example:
The following script should display the string in the False case, since the second string is not a suffix of the first.

```
. . .
If eStr.IsSuffix{'MyString','MyStr'}
   Display 'The second string is a Suffix';
Else
   Display 'The second string is not a Suffix';
EndIf
. . .
```

IsValidUTF8{}

The IsValidUTF8 function returns True if the string is a valid UTF-8 string.

Format:

```
Set BoolVal = eStr.IsValidUTF8{string};
```

Table 43: IsValidUTF8{} Options

Option Name	Option Description
string	The text of the first string to test.
BoolVal	The return value from the function. True indicates that the string is a valid UTF-8 string. It is False otherwise.

Example 1:
The following script should display the string in the True case, since these characters are valid for UTF-8.

```
. . .
If eStr.IsValidUTF8{'MyString'}
   Display 'The string is VALID';
Else
   Display 'The string is not VALID';
EndIf
. . .
```

Example 2:
The following script should display the string in the False case, since the character represented by the integer value 200 does not result in a valid for UTF-8.

```
. . .
If eStr.IsValidUTF8{200S}
   Display 'The string is VALID';
Else
   Display 'The string is not VALID';
EndIf
. . .
```

Location Functions

FindString{}

The FindString function finds a substring within a string. This method looks for an occurrence of the second string within the first string and returns the character position, if found. It returns 0 if not found.

Format:

```
Set charPos = eStr.FindString{string, findString
   [,'NoCase'][,'Back'][,'Word'][,startPos]};
```

Table 44: FindString{} Options

Option Name	Option Description
`string`	The text of the source string to compare.
`findString`	The text of the substring to search for.
`'NoCase'`	An optional string specifying that the search is case insensitive.
`'Back'`	An optional string specifying that the search starts from the end of the string.
`'Word'`	An optional string specifying that the search will check for whole words only.
`startPos`	The character position to start the search. The default value is 1.
`charPos`	The character position returned. The value will be 0 if the substring is not found .

Example:
The following script searches for the substring 'is the' in the first string. It should display the number 6 as the character position.

```
. . .
Set gvCharPos = eStr.FindString{'This is the string to check','is the'};
Display 'The character position is '+gvCharPos;
. . .
```

SubString{}

The SubString function picks off a substring by character position.

Format:

```
Set strVal = eStr.SubString{string [[,startPos] [,endPos]]};
```

Table 45: SubString{} Options

Option Name	Option Description
string	The source string.
startPos	The character position of the first character to retrieve. If not specified, then 1 is assumed.
endPos	The character position of the last character to retrieve. If not specified, then the last character in the string is assumed.
strVal	The returned substring.

Example:

The following script should pick off the characters in positions 6 through 11, which should be 'is the'.

```
. . .
Set gvStrVal = eStr.SubString{'This is the string to check',6,11};
Display 'The sub string is '+gvStrVal;
. . .
```

Character Replacement Functions

ToUpperCase{}

The ToUpperCase function changes all the characters in the string to upper case.

Format:

```
Set upperString = eStr.ToUpperCase{string};
```

Table 46: ToUpperCase{} Options

Option Name	Option Description
string	The source string.
upperString	The returned upper cased string.

Example:

The following script returns the upper case version of 'MyString' which should be 'MYSTRING'.

```
. . .
Set gvStrVal = eStr.ToUpperCase{'MyString'};
Display 'Upper case string is '+gvStrVal;
. . .
```

ToLowerCase{}

The ToLowerCase function changes all the characters in the string to lower case.

Format:

```
Set lowerString = eStr.ToLowerCase{string};
```

Table 47: ToLowerCase{} Options

Option Name	Option Description
string	The source string.
lowerString	The returned lower cased string.

Example:
The following script returns the lower case version of 'MyString' which should be 'mystring'.

```
. . .
Set gvStrVal = eStr.ToLowerCase{'MyString'};
Display 'Lower case string is '+gvStrVal;
. . .
```

Reverse{}

The Reverse function reverses the direction of the string characters.

Format:

```
Set revString = eStr.Reverse{string};
```

Table 48: Reverse{} Options

Option Name	Option Description
string	The source string.
revString	The returned reversed string.

Example:
The following script returns the reversed version of 'MyString' which should be 'gnirtSyM'.

```
. . .
Set gvStrVal = eStr.Reverse{'MyString'};
Display 'The reversed string is '+gvStrVal;
. . .
```

ReplaceAll{}

The ReplaceAll function replaces all occurrences of the specified substring with another substring.

Format:

```
Set strVal = eStr.ReplaceAll{string, repString, withString};
```

Table 49: ReplaceAll{} Options

Option Name	Option Description
string	The source string.
repString	The string to find.
withString	The string to replace the found string.
strVal	The returned string.

Example:
The following script displays the string after all occurrences of the substring 'is' are replaced by 'was'. The resulting string should be 'That was the week that was'.

```
. . .
Set gvStrVal = eStr.ReplaceAll{'That is the week that is','is','was'};
Display 'The new string is '+gvStrVal;
. . .
```

ReplaceFirst{}

The ReplaceFirst function replaces the first occurrence of the specified substring with another substring.

Format:

```
Set strVal = eStr.ReplaceFirst{string, repString, withString};
```

Table 50: ReplaceFirst{} Options

Option Name	Option Description
string	The source string.
repString	The string to find.
withString	The string to replace the found string.
strVal	The returned string.

Example:
The following script displays the string after replacing the first occurrence of the substring 'is' with 'was'. The resulting string should be 'That was the week that is'.

```
. . .
Set gvStrVal = eStr.ReplaceFirst{'That is the week that is','is','was'};
Display 'The new string is '+gvStrVal;
. . .
```

ReplaceLast{}

The ReplaceLast function replaces the last occurrence of the specified substring with another substring.

Format:

```
Set strVal = eStr.ReplaceLast{string, repString, withString};
```

Table 51: ReplaceLast{} Options

Option Name	Option Description
string	The source string.
repString	The string to find.
withString	The string to replace the found string.
strVal	The returned string.

Example:
The following script displays the string after replacing the last occurrence of the substring 'is' with 'was'. The resulting string should be 'That is the week that was'.

```
. . .
Set gvStrVal = eStr.ReplaceLast{'That is the week that is','is','was'};
Display 'The new string is '+gvStrVal;
. . .
```

PlatformToFrame{}

The PlatformToFrame function changes all the characters in the string from the platform character set to the FrameMaker character set. The source string should be in the platform character set. The lower ascii characters should be equivalent. The upper ascii characters are different especially on the MS Windows platform.

Format:

```
Set strVal = eStr.PlatformToFrame{string};
```

Table 52: PlatformToFrame{} Options

Option Name	Option Description
string	The source string.
strVal	The returned converted string.

Example:
The following script displays the converted string, before inserting it into the currently active document at the current insertion point. The upper ascii Î character needs to be converted to the Frame character set before inserting it into a document.

```
. . .
Set gvStrVal = eStr.PlatformToFrame{'MyStrÎng'};
Display 'The Converted string is '+gvStrVal;
New Text gvStrVal;
. . .
```

FrameToPlatform{}

The FrameToPlatform function changes all the characters in the string from the FrameMaker character set to the platform character set. The source string should be in the FrameMaker character set. The lower ascii characters should be equivalent. The upper ascii characters are different especially on the MS Windows platform.

Format:

```
Set strVal = eStr.FrameToPlatform{string};
```

Table 53: FrameToPlatform{} Options

Option Name	Option Description
string	The source string.
strVal	The returned converted string.

Example:
The following script gets the text from the paragraph in the paragraph object (in the variable gvPgfVar), converts it to the platform character set, before writing it to a text file.

```
. . .
Set gvSourceStr = gvPgfVar.Text;
Set gvStrVal = eStr.FrameToPlatform{gvSourceStr};
New TextFile NewVar(gvFilevar) File('outFile.txt');
Write object(gvFilevar) gvStrVal;
Close Textfile Object(gvFilevar);
. . .
```

ConvertText{}

The ConvertText function changes all the characters in the string from the specified encoding to the other specified encoding. The source string encoding is not specified, the current encoding is assumed. *FrameMaker 8 or greater*

Format:

```
Set strVal = eStr.ConvertText{string, toEncoding[,fromEncoding]};
```

Table 54: ConvertText{} Options

Option Name	Option Description
string	The source string.
toEncoding	The encoding to which to convert the string. Table 55 on page 41.
fromEncoding	The encoding of source string. Table 55 on page 41. *Optional* The current encoding is assumed if this option is not specified.
strVal	The returned converted string.

Table 55: Convert Text Encodings

Encoding Name	Description
"ASCII"	
"ANSI"	MS Windows Character Set
"JIS7"	Japanese
"ShiftJIS"	Japanese

Table 55: Convert Text Encodings

Encoding Name	Description
"JIS8_EUC"	Japanese
"Big5"	Traditional Chinese
"CNS_EUC"	Traditional Chinese
"GB8_EUC"	Simplified Chinese
"HZ"	Simplified Chinese
"KSC8_EUC"	Korean
"SpecialSymbol"	
"SpecialZapfDingbats"	
"SpecialWingdings"	
"UTF8"	Unicode
"FrameRoman"	Standard FrameMaker Character Set

Example 1:
The following script gets the text from the paragraph in the paragraph object (in the variable gvPgfVar), converts it from the current text encoding to UTF-8.

```
. . .
Set gvSourceStr = gvPgfVar.Text;
Set gvUtf8StrVal = eStr.ConvertText{gvSourceStr,'UTF8'};
. . .
```

Example 2:
The following script gets the text from the paragraph in the paragraph object (in the variable gvPgfVar), converts it from the current text encoding to UTF-8, then writes it out to a text file with Utf-8 byte order mark.

```
. . .
Set gvSourceStr = gvPgfVar.Text;
Set gvUtf8StrVal = eStr.ConvertText{gvSourceStr,'UTF8'};
New TextFile NewVar(gvFilevar) File('outUtf8.txt') Utf8Bom(1);
Write Object(gvFilevar) gvUtf8StrVal;
Close Textfile Object(gvFilevar);
. . .
```

Example 3:
The following script gets opens the text file and reads the first line. If the text file has a UTF-8 byte order mark, then it is converted to the Ansi encoding.

```
. . .
```

```
Open TextFile NewVar(gvFilevar) File('inUtf8.txt') BomVar(gvBomType);
Read File(gvFilevar) NewVar(gvTextLine);
If gvBomType=1
   Set gvTextLine=eStr.ConvertText{gvTextLine,'Ansi','UTF8'};
EndIf
Close Textfile Object(gvFilevar);
. . .
```

Insert and Remove Character Functions

InsertString{}

The InsertString function inserts one string into another string at the specified character position.

Format:

```
Set strVal = eStr.InsertString{string, insString, charPos};
```

Table 56: InsertString{} Options

Option Name	Option Description
`string`	The source string.
`insString`	The string to insert.
`charPos`	The character position before which to insert the new string. If this value is missing or less than 1, then it will insert the new string at the beginning of the source string.
`strVal`	The returned string.

Example:

The following script inserts the string 'New' into 'MyString' at position 3 which should result in 'MyNewString'.

```
. . .
Set gvStrVal = eStr.InsertString{'MyString','New',3};
Display 'The new string is '+gvStrVal;
. . .
```

RemoveChars{}

The RemoveChars function removes all of the specified characters from the source string.

Format:

```
Set strVal = eStr.RemoveChars{string, remChars};
```

Table 57: RemoveChars{} Options

Option Name	Option Description
`string`	The source string.

Table 57: RemoveChars{} Options

Option Name	Option Description
remChars	The string consisting of a set of characters to remove.
strVal	The returned string.

Example:
The following script removes all the vowels from the source string 'MyNewString' which should result in 'MNwStrng'.

```
. . .
Set gvStrVal = eStr.RemoveChars{'MyNewString','yei'};
Display 'The new string is '+gvStrVal;
. . .
```

RemoveLeading{}

The RemoveLeading function removes the specified characters from the beginning of the source string.

Format:

```
Set strVal = eStr.RemoveLeading{string, remChars};
```

Table 58: RemoveLeading{} Options

Option Name	Option Description
string	The source string.
remChars	The string consisting of a set of characters to remove from the beginning of the string.
strVal	The returned string.

Example:
The following script removes the leading spaces and asterisks from the source string' ***MyNewString*** 'which should result in 'MyNewString*** '.

```
. . .
Set gvStrVal = eStr.RemoveLeading{'   ***MyNewString***   ',' *'};
Display 'The new string is '+gvStrVal;
. . .
```

RemoveTrailing{}

The RemoveTrailing function removes the specified characters from the end of the source string.

Format:

```
Set strVal = eStr.RemoveTrailing{string, remChars};
```

Table 59: RemoveTrailing{} Options

Option Name	Option Description
string	The source string.

Table 59: RemoveTrailing{} Options

Option Name	Option Description
remChars	The string consisting of a set of characters to remove from the end of the string.
strVal	The returned string.

Example:
The following script removes the trailing spaces and asterisks from the source string ' ***MyNewString*** ' which should result in ' ***MyNewString'.

```
. . .
Set gvStrVal = eStr.RemoveTrailing{'   ***MyNewString***   ',' *'};
Display 'The new string is '+gvStrVal;
. . .
```

Example:
Putting the last two functions together, the following script removes the leading and trailing spaces and asterisks from the source string ' ***MyNewString*** ' which should result in 'MyNewString'.

```
. . .
Set gvStrVal = eStr.RemoveLeading{
       eStr.RemoveTrailing{'   ***MyNewString***   ',' *'},
       ' *'};
Display 'The new string is '+gvStrVal;
. . .
```

Miscellaneoous Functions

CharCount{}

The CharCount function returns the number of characters in the string.

Format:

```
Set intVal = eStr.CharCount{string};
```

Table 60: CharCount{} Options

Option Name	Option Description
string	The source string.
intVal	The returned integer number of characters.

Example:
The following script displays the number of characters in the source string 'MyString', which should be 8.

```
. . .
Set gvIntVal = eStr.CharCount{'MyString'};
Display 'The number of characters is '+gvIntVal;
. . .
```

LoadFromTextFile{}

The LoadFromTextFile function reads the entire contents of a text file into the string. When the text is read from the file all line separators become a LF (or linefeed) character on all platforms.

Format:

```
Set strVal = eStr.LoadFromTextFile{filename};
```

Table 61: LoadFromTextFile{} Options

Option Name	Option Description
fileName	The name of the text file.
strVal	The returned string with the contents of the text file.

Example:
The following script reads the contents of the 'MyTextFile.txt' file located in the product directory.

```
. . .
Set gvStrVal = eStr.LoadFromTextFile{ClientDir + DIRSEP + 'MyTextFile.txt'};
. . .
```

SaveToTextFile{}

The SaveToTextFile function writes the string to a text file. When the text is written to the file the line separator character(LF) becomes the appropriate line separator(s) for the platform (CR LF for MS Windows, CR for the Macintosh and LF for Unix).

Format:

```
Set intVal = eStr.SaveToTextFile{string,filename[,'APPEND']};
```

Table 62: SaveToTextFile{} Options

Option Name	Option Description
string	The string to save.
fileName	The name of the output text file.
'APPEND'	An optional parameter indicating that the string is to be appended to the file instead of overwriting it.
intVal	The returned integer value containing the length of the text written.

Example:
The following script writes the contents of the specified string to the text file 'MyTextFile.txt' file located in the product directory.

```
. . .
Set gvIntVal = eStr.SaveToTextFile{'String To Write',
       ClientDir + DIRSEP + 'MyTextFile.txt'};
. . .
```

ToStringList{}

The ToStringList function converts a string into a stringlist value using the specified separation character.

Format:

```
Set stringList = eStr.ToStringList{string,sepChar};
```

Table 63: ToStringList{} Options

Option Name	Option Description
string	The string to convert.
sepChar	The character to use as a separator.
stringList	The returned stringlist value.

Example:

The following script converts the source string (using '|' as a separator) into a string list and writes the result to the FrameMaker console.

```
. . .
Set gvStringList = eStr.ToStringList{'String1|String2|String3|String4','|'};
Write Console gvStringList;
. . .
STRINGLIST Count(4)
String1
String2
String3
String4
. . .
```

Chapter 4

Array Objects

This chapter discusses the ElmScript array objects. These are EArray, EVector, EStackList, ECollection and EStructure. EArray is a fixed size array similar in spirit to a standard array in most computer languages. The EVector array type provides an extensible array. The EStackList provides a stack type array, where you push and pop members in a first in, last out method. An ECollection is a indexed linked list. Member can be accessed sequentially or, if an index is provided, accessed by index. The EStructure lets you build a structure that groups variables.

EArray Object

The EArray object represents a fixed array of data values. The bracket operators ([]) are used to access the members of the array. Attempting to access a member number outside the boundaries will cause an Out of Range error.

Creating the object

You create an instance of this object using the New command. There is also a utility function (under eUtl) that will create an EStructure object (See “EArray{}” on page 244.).

Format:

```
New EArray Count(numberOfMembers) NewVar(arrayVar);
```

Table 64: New EArray Options

Option Name	Option Description
`Count`	An integer value giving the number of members for the array.
`NewVar`	The name of the variable to hold the newly created object.

Example:
The following script creates an EArray object, fills it with some data, makes a computation, then dumps the array.

```
. . .
New EArray NewVar(gvArray) Count(10);
Set gvArray[1] = 100;
Set gvArray[2] = 'String Data';
Loop InitVal(3) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Set gvArray[gvIdx] = 1000;
EndLoop
Set gvArray[10] = gvArray[9] + gvArray[8];
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Write console 'Index-'+gvIdx+' Value-'+gvArray[gvIdx];
EndLoop
. . .
```

EArray Object Methods

Table 65: EVector Methods

Method Name	Method Description
FindPos	Finds the position of a data value.
Sort	Sorts the array in place by data values.
SortIndirect	Sorts the array by data values retuning the sorting order into an IntList value, which gives the order of the items as integer indexes into the array. This lets get the sorting order of the array without actually changing anything.

EArray Object Method Descriptions

FindPos

The FindPos method finds the position.

Format:

```
Run eArrayVar.FindPos Value(value) NewVar(position);
or
Set position = eArrayVar.FindPos{value};
```

Table 66: FindPos Options

Option Name	Option Description
Value	The value for which to search.
NewVar	The name of the variable to hold the position number of the found item. Zero (0) if not found.

Example:
The following script builds an array of values then searches for number 1005. It should return the integer value 5.

```
. . .
New EArray NewVar(gvArray) Count(10);
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Set gvArray[gvIdx] = 1000+gvIdx;
EndLoop
Set gvPos = gvArray.FindPos{1005};
. . .
```

Sort

The Sort method sorts the array in place. The position of the members change to accommodate the sorting options specified.

> **IMPORTANT:** The EArray object can have members with different data types, but sorting will only work if all the members have the same data type. Otherwise, the sort will fail and return a negative error code. Also, even if all the data types in the array are the same, the data types must be sortable, that is, there must be some logic to comparing one item to see if it is less than or greater than another one. Some data types such as StringList, Point, and Tab only have limited ways to compare them.

Format:

```
Run eArrayVar.Sort [Option(sortOption1)] [Option(sortOption2)] NewVar(errCode);
or
Set errCode = eArrayVar.Sort{[sortOption1][,sortOption2]};
```

Table 67: Sort Options

Option Name	Option Description
`sortOptionI`	A sort option specifier: 'A' for ascending, 'D' for descending, 'C' for case sensitive, 'N' for not case sensitive. the 'N' and 'C' options only apply to string type sorts. It is ignored otherwise.
`NewVar`	The name of the variable to store errorcode returned. If the sorting is successful this value should be 0. Otherwise it is an error code. Also, you will need to check the session variable ErrorCode to make sure that the sort worked correctly.

Example:
The following script builds an array of values then sorts them in ascending order.

```
. . .
New EArray NewVar(gvArray) Count(10);
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Set gvArray[gvIdx] = 1000-gvIdx;
EndLoop
/* The array should be in the following order
999 998 997 996 995 994 993 992 991 990
*/
Set gvStatus = gvArray.Sort{};
/* The array should now be in the following order
990 991 992 993 994 995 996 997 998 999
*/
. . .
```

Example:
The following script builds an array of string values using the utility function.

```
. . .
Set gvArray = eUtl.EArray{'harpo','Frodo','Samwise',
                          'Merry','Pippin','Bilbo','Gandalf'};
Set gvStatus = gvArray.Sort{};
Set gvCount = gvArray.Count;
Write console 'After Case Sensitive Sort';
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=gvCount)
   Write console 'Index-'+gvIdx+' Value-'+gvArray[gvIdx];
EndLoop
/* The array should now be in the following order
'Bilbo' 'Frodo' 'Gandalf' 'Merry' 'Pippin' 'Samwise' 'harpo'

   Note that the 'harpo' string appears at the end because the lower case characters
have higher character codes than upper case characters.
*/
Set gvStatus = gvArray.Sort{'N'};
Set gvCount = gvArray.Count;
Write console 'After Case InSensitive Sort';
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=gvCount)
   Write console 'Index-'+gvIdx+' Value-'+gvArray[gvIdx];
EndLoop
/* The array should now be in the following order
'Bilbo' 'Frodo' 'Gandalf' 'harpo' 'Merry' 'Pippin' 'Samwise'

   Note that the 'harpo' string now appears in the middle because the sorting is now
case insensitive. The lower case characters are now equivalent to upper case
characters for sorting purposes.
*/
Set gvStatus = gvArray.Sort{'D','N'};
Set gvCount = gvArray.Count;
Write console 'After Case InSensitive, Descending Sort';
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=gvCount)
   Write console 'Index-'+gvIdx+' Value-'+gvArray[gvIdx];
EndLoop
/* The array should now be in the following order
'Samwise' 'Pippin' 'Merry' 'harpo' 'Gandalf' 'Frodo' 'Bilbo'
*/
. . .
```

SortIndirect

The SortIndirect method sorts the array and places the indexes of the sorted items into an IntList value. The position of the members in the array itself do *not* change. If the sort is successful, the IntList return value will contain a list of indexes into the array that will specify the correct order, according to the sorting options. For example, if the return value is stored in a variable called gvSortList, then gvSortList[1] will contain the index for the first sorted item. So if gvArray is the array that was sorted, then gvArray[gvSortList[1]] will give you the first item in the sorted list.

Use the SortIndirect method or function, when you want a sorted order but do not want to change the original array.

> **IMPORTANT:** The EArray object can have members with different data types, but sorting will only work if all the members have the same data type. Otherwise, the sort will fail and return a negative error code. Also, even if all the data types in the array are the same, the data types must be sortable, that is, there must be some logic to

comparing one item to see if it is less than or greater than another one. Some data types such as StringList, Point, and Tab only have limited ways to compare them.

Format:

```
Run eArrayVar.SortIndirect [Option(sortOption1)] [Option(sortOption2)]
          NewVar(intListVar);
or
Set intListVar = eArrayVar.SortIndirect{[sortOption1][,sortOption2]};
```

Table 68: SortIndirect Options

Option Name	Option Description
sortOptionI	A sort option specifier: 'A' for ascending, 'D' for descending, 'C' for case sensitive, 'N' for not case sensitive. the 'N' and 'C' options only apply to string type sorts. It is ignored otherwise.
NewVar	The name of the variable to store the returned IntList value containing the sorted indexes. If the sorting is *not* successful this value should be NULL. Also, you will need to check the session variable ErrorCode to make sure that the sort worked correctly. If ErrorCode is 0, then the sort worked.

Example:

The following script builds an array of values then sorts them in ascending order putting the result in gvSortedList.

```
. . .
New EArray NewVar(gvArray) Count(10);
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Set gvArray[gvIdx] = 1000-gvIdx;
EndLoop
/* The array should be in the following order
999 998 997 996 995 994 993 992 991 990
*/
Set ErrorCode = 0;
Set gvSortedList = gvArray.SortIndirect{};
If ErrorCode = 0
   /* The array should still be in the following order
   999 998 997 996 995 994 993 992 991 990
   and the gvSortedList should have the following members
   10 9 8 7 6 5 4 3 2 1
   */
   Set gvCount = gvSortedList.Count;
   Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx<=gvCount);
      Write console '#-'+ ' Index-'+gvSortedList[gvIdx]+' Value-'+
                  gvVector[gvSortedList[gvIdx]];
   EndLoop
Else
   Display 'An error has occurred while sorting-'+ErrorMsg;
EndIf
. . .
```

EVector Object

The EVector object represents an extensible array of data values. The bracket operators ([]) are used to access the members of the array. The first member is at index 1. The last member can be determined by the Count property. Attempting to access a member number outside the boundaries will cause an Out of Range error. You can also remove members with the PopFront and PopBack methods and add them with the PushFront and PushBack methods.

Creating the object

You create an instance of this object using the **`New`** command. This command always creates an empty EVector object. Most of the time you can ignore the InitCount or Incr options. These are present for performance reasons only. When you create an EVector object, it allocates a number of places to store possible members. These are called slots. These slots make it very efficient to add new members at the end of the list (PushBack method). It just uses one of these previously allocated slots. When it runs out of slots, it allocates more (the number is based on the Incr option). This takes a little more time. If you know that you are going to have a large array (many members added), then you might want to use the InitCount option to specify a large number of initial slots for data members. The EVector will always add more slots for members as needed so the Push commands will always work. It will just take a little longer if it has to keep allocating more slots. There is also a utility function (under eUtl) that will create an EStructure object (See "EVector{}" on page 244.).

Format:

```
New EVector {InitCount(nMember)] [Incr(incrCount)] NewVar(vectorVar);
```

Table 69: New EVector Options

Option Name	Option Description
`InitCount`	An optional value specifying the initial allocation size for the array. When you add members, it starts out with this initial allocation of empty slots. The default value is 10 slots.
`Incr`	An optional value specifying the subsequent allocation size for the array. When the initial allocation of empty slots is used, it adds more in groups of this amount. The default value is new 10 slots.
`NewVar`	The name of the variable to hold the newly created object.

Example:
The following script creates an EVector object, fills it with some data, makes a computation, then dumps the array. This is similar to the EArray example, except in this case, you can continue to add and remove members to and from the list.

```
. . .
New EVector NewVar(gvVector);
Run gvVector.PushBack Value(100);
Run gvVector.PushBack Value('String Data');
Loop InitVal(3) Incr(1) LoopVar(gvIdx) While(gvIdx <=10);
   Run gvVector.PushBack Value(1000);
EndLoop
Set gvCount = gvVector.Count;
Set gvVector[gvCount] = gvVector[9] + gvVector[8];
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=gvCount);
   Write console 'Index-'+gvIdx+' Value-'+gvVector[gvIdx];
EndLoop
// Add some strings
Run gvVector.PushBack Value('String1');
Run gvVector.PushBack Value('String2');
Run gvVector.PushBack Value('String3');
Set gvCount = gvVector.Count
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=gvCount);
   Write console 'Index-'+gvIdx+' Value-'+gvVector[gvIdx];
EndLoop
. . .
```

Creating an EVector using the join operator

An alternate way to create an EVector object is to use the join (comma (,)) operator. The join operator takes the two operands and makes an EVector of those values. For example,

```
Set gvVector = 10,20;
```

this will create an EVector with two members. You can use this operator and the parentheses to build complex array structures. The following will create an EVector with an EVector as a member.

```
Set gvVector = (10, (50,60),20);
```

The parentheses makes the middle numbers a separate vector. You could access the middle values as follows:

```
Set gvValue1 = gvVector[2][1];
Set gvValue2 = gvVector[2][2];
```

gvValue1 should have the value 50 and gvValue2 should be 60.

A more practical example follows:

```
Set gvLookUpTable = (
      ('Frodo', 'Hobbit', 50),
      ('Samwise', 'Hobbit', 46),
      ('Aragorn', 'Dunedain', 87),
      ('Gandalf', 'Istari', 2500),
      ('Elrond', 'Elf', 3500),
      ('Boromir', 'Human', 40)
   );
```

Here is an EVector build to be a look up table. This EVector's members are all EVectors themselves consisting of a set of records with three values, a name, a type, and an age. The following function can be used to search the structure by name.

```
Function LookupByName using pvTable pvName
   Result = NULL;
   If pvTable.ObjectName = 'EVector'
      Local lvIdx(0);
      Loop LoopVar(lvIdx) InitVal(1) Incr(1) While(lvIdx<=pvTable.Count)
         If pvName = pvTable[lvIdx][1]
            Set result = pvTable[lvIdx];
            LeaveSub;
         Endif
      EndLoop
   EndIf
EndFunc
```

The function returns NULL if the name is not found. Here is how you would call the search function:

```
Set gvRecord = LookupByName{gvLookUpTable,'Gandalf');
```

This will return the EVector object corresponding to the name 'Gandalf'.

EVector Properties

Table 70: EVector Properties

Property Name	Data Type	Property Description
`Count` *(Read-Only)*	`Integer`	The number of members in the vector. The last member of the array can be retrieved or set using the bracket operator and this property as follows: `Set gvLastMember = gvVector [gvVector.Count];`
`SlotCount`	`Integer`	The number of slots currently allocated.
`SlotIncr`	`Integer`	The number of slots to allocate whenever more slots are needed.

EVector Object Methods

Table 71: EVector Methods

Method Name	Method Description
PushBack	Adds one or more members to the end of the array.
PushFront	Adds one or more members to the beginning of the array.
PopBack	Returns and removes the last data member.
PopFront	Returns and removes the first data member.
Insert	Insert a new member at the specified position.
Remove	Remove a specified member from the array
FindPos	Finds the position of a data value.
Sort	Sorts the array in place by data values.
SortIndirect	Sorts the array by data values retuning the sorting order into an IntList value, which gives the order of the items as integer indexes into the array. This lets get the sorting order of the array without actually changing anything.

EVector Object Method Descriptions

PushBack

The PushBack method adds a new data value or list of data values to the end of the array.

Format:

```
Run eVectorVar.PushBack Value(value) {Value(value2)} ... [Value(valueN)];
or
eVectorVar.PushBack{value[,value2] ... [,valueN]};
```

Table 72: PushBack Options

Option Name	Option Description
ValueI	The value(s) to add to the end of the array.

Example:

The following script creates an EVector object and fills it with some data. The first value is a Frame Object (ActiveDoc) followed by a string, then 10 numbers are added to the end in order 1001 through 1010.

```
. . .
New EVector NewVar(gvVector);
Run gvVector.PushBack Value(ActiveDoc) Value('String Data'); // Add two values
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Run gvVector.PushBack Value(1000+gvIdx);
EndLoop
. . .
/* The vector should be in the following order
ActiveDoc 'String Data' 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010
*/
```

PushFront

The PushFront method adds a new data value or list of data values to the beginning of the array. Note that adding items to the beginning is much less efficient than adding them to the end.

Format:

```
Run eVectorVar.PushFront Value(value) {Value(value2)} ... [Value(valueN)];
or
eVectorVar.PushFront{value[,value2] ... [,valueN]};
```

Table 73: PushFront Options

Option Name	Option Description
`ValueI`	The value(s) to add to the beginning of the array. Each of these values are added as a group so they will be in the same order in the list.

Example:

The following script creates an EVector object and fills it with some data. The loop adds 10 integers to the front of the array. Note that they go in in reverse order from the PushBack example. The end result will have 1010 as the first member in the array and 1001 as the 10 th member. This is because each successive number goes into the first position and pushes the rest up in the list.

```
. . .
New EVector NewVar(gvVector);
Run gvVector.PushBack Value(ActiveDoc) Value('String Data'); // Add two values
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <= 10)
   Run gvVector.PushFront Value(1000+gvIdx);
EndLoop
. . .
/* The vector should be in the following order
1010 1009 1008 1007 1006 1005 1004 1003 1002 1001 ActiveDoc 'String Data'
*/
```

Example:

The following script creates an EVector object and fills it with some data. It adds ten integer members similar to those of the last example, but this time they go in the order given because they all go in at once.

```
. . .
New EVector NewVar(gvVector);
Run gvVector.PushBack Value(ActiveDoc) Value('String Data'); // Add two values
gvVector.PushFront{1001,1002,1003,1004,1005,1006,1007,1008,1009,1010};
. . .
/* The vector should be in the following order
1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 ActiveDoc 'String Data'
*/
```

PopBack

The PopBack method removes a data value from the end of the array.

Format:

```
Run eVectorVar.PopBack NewVar(varName);
or
Set varName = eVectorVar.PopBack{};
```

Table 74: PopBack Options

Option Name	Option Description
NewVar	The name of the variable to put the removed value.

Example:
The following script creates an EVector object and fills it with some data. The first value is a Frame Object (ActiveDoc) followed by a string, then 10 numbers are added to the end in order 1001 through 1010. It then pops a few values from the end of the array.

```
. . .
New EVector NewVar(gvVector);
Run gvVector.PushBack Value(ActiveDoc) Value('String Data'); // Add two values
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Run gvVector.PushBack Value(1000+gvIdx);
EndLoop
/* The vector should be in the following order
ActiveDoc 'String Data' 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010
*/
Set gvNum1 = gvVector.PopBack{};
Set gvNum2 = gvVector.PopBack{};
Set gvNum3 = gvVector.PopBack{};
/* The vector should now be in the following order
   ActiveDoc 'String Data' 1001 1002 1003 1004 1005 1006 1007
and
   gvNum1 is 1010, gvNum2 is 1009 and gvNum3 is 1008
*/
. . .
```

PopFront

The PopFront method removes a data value from the beginning of the array. Note that removing items from the beginning is much less efficient than removing them from the end.

Format:

```
Run eVectorVar.PopFront NewVar(varName);
or
Set varName = eVectorVar.PopFront{};
```

Table 75: PopFront Options

Option Name	Option Description
NewVar	The name of the variable to put the removed value.

Example:
The following script creates an EVector object and fills it with some data. The first value is a Frame Object (ActiveDoc) followed by a string, then 10 numbers are added to the end in order 1001 through 1010. It then pops a few values from the end of the array.

```
. . .
New EVector NewVar(gvVector);
Run gvVector.PushBack Value(ActiveDoc) Value('String Data'); // Add two values
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Run gvVector.PushBack Value(1000+gvIdx);
EndLoop
/* The vector should be in the following order
ActiveDoc 'String Data' 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010
*/
Set gvNum1 = gvVector.PopFront{};
Set gvNum2 = gvVector.PopFront{};
Set gvNum3 = gvVector.PopFront{};
/* The vector should now be in the following order
   1002 1003 1004 1005 1006 1007 1008 1009 1010
and
   gvNum1 is ActiveDoc, gvNum2 is String Data' and gvNum3 is 1001
*/
. . .
```

Insert

The Insert method adds a new data value to a specified position in the array.

Format:

```
Run eVectorVar.Insert Value(value) [Before(number)];
or
Set status = eVectorVar.Insert{value[,number]};
```

Table 76: Insert Options

Option Name	Option Description
Value	The value to insert.
number	The number of the member before which to add the new value. If not given then the new value will be inserted at the beginning.

Example:
The following script creates an EVector object and fills it with some data. Then inserts some other numbers in the middle.

```
. . .
New EVector NewVar(gvVector);
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Run gvVector.PushBack Value(1000+gvIdx);
EndLoop
/* The vector should be in the following order
1001 1002 1003 1004 1005 1006 1007 1008 1009 1010
*/
Run gvVector.Insert Value(2000) Before(5);
Run gvVector.Insert Value(2001) Before(10);
/* The vector should now be in the following order
1001 1002 1003 1004 2000 1005 1006 1007 1008 2001 1009 1010
*/
. . .
```

Remove

The Remove method adds a new data value to a specified position in the array.

Format:

```
Run eVectorVar.Remove Number(position);
or
Set status = eVectorVar.Remove{position};
```

Table 77: Remove Options

Option Name	Option Description
`position`	The number of the member to remove.

Example:
The following script creates an EVector object and fills it with some data. Then inserts some other numbers in the middle, then removes some others.

```
. . .
New EVector NewVar(gvVector);
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Run gvVector.PushBack Value(1000+gvIdx);
EndLoop
/* The vector should be in the following order
1001 1002 1003 1004 1005 1006 1007 1008 1009 1010
*/
Run gvVector.Insert Value(2000) Before(5);
Run gvVector.Insert Value(2001) Before(10);
/* The vector should now be in the following order
1001 1002 1003 1004 2000 1005 1006 1007 1008 2001 1009 1010
*/
Run gvVector.Remove Number(3);
Run gvVector.Remove Number(11);
/* The vector should now be in the following order
1001 1002 1004 2000 1005 1006 1007 1008 2001 1009
*/
. . .
```

FindPos

The FindPos method finds the position.

Format:

```
Run eVectorVar.FindPos Value(value) NewVar(position);
or
Set position = eVectorVar.FindPos{value};
```

Table 78: FindPos Options

Option Name	Option Description
NewVar	The name of the variable to hold the position number of the found item. Zero (0) if not found.

Example:
The following script builds an array of values then searches for number 1005. It should return the integer value 5.

```
. . .
New EVector NewVar(gvVector);
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Run gvVector.PushBack Value(1000+gvIdx);
EndLoop
Set gvPos = gvVector.FindPos{1005};
. . .
/* The vector should be in the following order
   1001 1002 1003 1004 1005 1006 1007 1008 1009 1010
and
   gvPos should be 5
*/
```

Sort

The Sort method sorts the array in place. The position of the members change to accommodate the sorting options specified.

> **IMPORTANT:** The EArray object can have members with different data types, but sorting will only work if all the members have the same data type. Otherwise, the sort will fail and return a negative error code. Also, even if all the data types in the array are the same, the data types must be sortable, that is, there must be some logic to comparing one item to see if it is less than or greater than another one. Some data types such as StringList, Point, and Tab only have limited ways to compare them.

Format:

```
Run eVectorVar.Sort [Option(sortOption1)] [Option(sortOption2)] NewVar(errCode);
or
Set errCode = eVectorVar.Sort{[sortOption1][,sortOption2]};
```

Table 79: Sort Options

Option Name	Option Description
sortOptionI	A sort option specifier: 'A' for ascending, 'D' for descending, 'C' for case sensitive, 'N' for not case sensitive. the 'N' and 'C' options only apply to string type sorts. It is ignored otherwise.
NewVar	The name of the variable to store errorcode returned. If the sorting is successful this value should be 0. Otherwise it is an error code. Also, you will need to check the session variable ErrorCode to make sure that the sort worked correctly.

Example:

The following script builds an array of values then sorts them in ascending order.

```
. . .
New EVector NewVar(gvVector);
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Run gvVector.PushBack Value(1000-gvIdx);
EndLoop
/* The array should be in the following order
999 998 997 996 995 994 993 992 991 990
*/
Set gvStatus = gvVector.Sort{};
/* The array should now be in the following order
990 991 992 993 994 995 996 997 998 999
*/
. . .
```

Example:
The following script builds an array of string values using the utility function.

```
. . .
Set gvVector = eUtl.EVector{'harpo','Frodo','Samwise',
                        'Merry','Pippin','Bilbo','Gandalf'};
Set gvStatus = gvVector.Sort{};
Set gvCount = gvVector.Count;
Write console 'After Case Sensitive Sort';
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=gvCount)
   Write console 'Index-'+gvIdx+' Value-'+gvVector[gvIdx];
EndLoop
/* The array should now be in the following order
'Bilbo' 'Frodo' 'Gandalf' 'Merry' 'Pippin' 'Samwise' 'harpo'

   Note that the 'harpo' string appears at the end because the lower case characters
have higher character codes than upper case characters.
*/
Set gvStatus = gvVector.Sort{'N'};
Set gvCount = gvVector.Count;
Write console 'After Case InSensitive Sort';
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=gvCount)
   Write console 'Index-'+gvIdx+' Value-'+gvVector[gvIdx];
EndLoop
/* The array should now be in the following order
'Bilbo' 'Frodo' 'Gandalf' 'harpo' 'Merry' 'Pippin' 'Samwise'

   Note that the 'harpo' string now appears in the middle because the sorting is now
case insensitive. The lower case characters are now equivalent to upper case
characters for sorting purposes.
*/
Set gvStatus = gvVector.Sort{'D','N'};
Set gvCount = gvVector.Count;
Write console 'After Case InSensitive, Descending Sort';
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=gvCount)
   Write console 'Index-'+gvIdx+' Value-'+gvVector[gvIdx];
EndLoop
/* The array should now be in the following order
'Samwise' 'Pippin' 'Merry' 'harpo' 'Gandalf' 'Frodo' 'Bilbo'
*/
. . .
```

SortIndirect

The SortIndirect method sorts the array and places the indexes of the sorted items into an IntList value. The position of the members in the array itself do *not* change. If the sort is successful, the IntList return value will contain a list of indexes into the array that will specify the correct order, according to the sorting options. For example, if the return value is stored in a variable called gvSortList, then gvSortList[1] will contain the index for the first sorted item. So if gvArray is the array that was sorted, then gvArray[gvSortList[1]] will give you the first item in the sorted list.

Use the SortIndirect method or function, when you want a sorted order but do not want to change the original array.

> **IMPORTANT:** The EArray object can have members with different data types, but sorting will only work if all the members have the same data type. Otherwise, the sort will fail and return a negative error code. Also, even if all the data types in the array are the same, the data types must be sortable, that is, there must be some logic to

comparing one item to see if it is less than or greater than another one. Some data types such as StringList, Point, and Tab only have limited ways to compare them.

Format:

```
Run eVectorVar.SortIndirect [Option(sortOption1)] [Option(sortOption2)]
          NewVar(intListVar);
or
Set intListVar = eVectorVar.SortIndirect{[sortOption1][,sortOption2]};
```

Table 80: SortIndirect Options

Option Name	Option Description
sortOptionI	A sort option specifier: 'A' for ascending, 'D' for descending, 'C' for case sensitive, 'N' for not case sensitive. the 'N' and 'C' options only apply to string type sorts. It is ignored otherwise.
NewVar	The name of the variable to store the returned IntList value containing the sorted indexes. If the sorting is *not* successful this value should be NULL. Also, you will need to check the session variable ErrorCode to make sure that the sort worked correctly. If ErrorCode is 0, then the sort worked.

Example:

The following script builds an array of values then sorts them in ascending order putting the result in gvSortedList.

```
. . .
New EVector NewVar(gvVector);
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Run gvVector.PushBack Value(1000-gvIdx);
EndLoop
/* The array should be in the following order
999 998 997 996 995 994 993 992 991 990
*/
Set ErrorCode = 0;
Set gvSortedList = gvVector.SortIndirect{};
If ErrorCode = 0
   /* The array should still be in the following order
   999 998 997 996 995 994 993 992 991 990
   and the gvSortedList should have the following members
   10 9 8 7 6 5 4 3 2 1
   */
   Set gvCount = gvSortedList.Count;
   Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx<=gvCount);
      Write console '#-'+ ' Index-'+gvSortedList[gvIdx]+' Value-'+
                  gvVector[gvSortedList[gvIdx]];
   EndLoop
Else
   Display 'An error has occured while sorting-'+ErrorMsg;
EndIf
. . .
```

EStackList Object

The EStackList object represents a stack array of data values. This is a simplified array type. Access is from the top of the stack only. You can get the number of values on the stack with the Count property and you can use the top value on the stack with the Peek property. You use the Push method or function to add members to the top of the stack and you use the Pop method or function to remove values.

Creating the object

You create an instance of this object using the **New** command. This command always creates an empty EStackList object. The creation of this object is similar to the EVector object. Most of the time you can ignore the InitCount or Incr options. These are present for performance reasons only. When you create an EStackList object, it allocates a number of places to store possible members. These are called slots. These slots make it very efficient to add new members (Push method). It just uses one of these previously allocated slots. When it runs out of slots, it allocates more (the number is based on the Incr option). This takes a little more time. If you know that you are going to have a large array (many members added), then you might want to use the InitCount option to specify a large number of initial slots for data members. The EStackList will always add more slots for members as needed so the Push commands will always work. It will just take a little longer if it has to keep allocating more slots. There is also a utility function (under eUtl) that will create an EStructure object (See "EStackList{}" on page 244.).

Format:

```
New EStackList {InitCount(nMember)] [Incr(incrCount)] NewVar(stackVar);
```

Table 81: New EStackList Options

Option Name	Option Description
InitCount	An optional value specifying the initial allocation size for the stack. When you add members, it starts out with this initial allocation of empty slots. The default value is 10 slots.
Incr	An optional value specifying the subsequent allocation size for the stack. When the initial allocation of empty slots is used, it adds more in groups of this amount. The default value is new 10 slots.
NewVar	The name of the variable to hold the newly created object.

Example:
The following script creates an EStackList object and fills it with some data. It then pops some values off.

```
. . .
New EStackList NewVar(gvStack);
Run gvStack.Push Value(100) Value('String Data');
Loop InitVal(3) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Run gvStack.Push Value(1000+gvIdx);
EndLoop
/* The stack should be in the following order
ActiveDoc 'String Data' 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 <--TOP
*/
// Pop some values
Set gvNum1 = gvStack.Pop{};
Set gvNum2 = gvStack.Pop{};
/* The stack should now be in the following order
   ActiveDoc 'String Data' 1001 1002 1003 1004 1005 1006 1007 1008 <--TOP
and
   gvNum1 is 1010 and gvNum2 is 1009
*/
. . .
```

EStackList Properties

Table 82: EStackList Properties

Property Name	Data Typ;e	Property Description
Count ***(Read-Only)***	**Integer**	The number of members in the vector.
IsEmpty ***(Read-Only)***	**Integer**	True if the stack is empty, False otherwise.
SlotCount	**Integer**	The number of slots currently allocated.
SlotIncr	**Integer**	The number of slots to allocate whenever more slots are needed.

EStackList Object Methods

Table 83: EStackList Methods

Method Name	Method Description
Push	Adds one or more members to the top of the stack.
Pop	Returns and removes the data value at the top of the stack.

EStackList Object Method Descriptions

Push

The Push method adds a new data value or list of data values to the top of the stack.

Format:

```
Run stackVar.Push Value(value) {Value(value2)} ... [Value(valueN)];
or
stackVar.Push{value[,value2] ... [,valueN]};
```

Table 84: Push Options

Option Name	Option Description
ValueI	The value(s) to add to the top of the stack.

Example:

The following script creates an EStackList object and fills it with some data. The first value is a Frame Object (ActiveDoc) followed by a string, then 10 numbers are added to the top of the stack in order 1001 through 1010.

```
. . .
New EStackList NewVar(gvStack);
gvStack.Push Value(ActiveDoc, 'String Data'); // Add two values
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Run gvStack.Push Value(1000+gvIdx);
EndLoop
. . .
/* The stack should be in the following order
ActiveDoc 'String Data' 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 <--TOP
*/
```

Pop

The Pop method removes a data value from the end of the array.

Format:

```
Run stackVar.Pop NewVar(varName);
or
Set varName = stackVar.Pop{};
```

Table 85: Pop Options

Option Name	Option Description
NewVar	The name of the variable to put the removed value.

Example:
The following script creates an EStackList object and fills it with some data. It then pops a few values from the end of the stack.

```
. . .
New EStackList NewVar(gvStack);
gvStack.Push{ActiveDoc,'String Data'}; // Add two values
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <= 10)
   Run gvStack.Push Value(1000+gvIdx);
EndLoop
/* The vector should be in the following order
ActiveDoc 'String Data' 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 <--TOP
*/
Set gvNum1 = gvStack.Pop{};
Set gvNum2 = gvStack.Pop{};
Set gvNum3 = gvStack.Pop{};
/* The vector should now be in the following order
   ActiveDoc 'String Data' 1001 1002 1003 1004 1005 1006 1007 <--TOP
and
   gvNum1 is 1010, gvNum2 is 1009 and gvNum3 is 1008
*/
. . .
```

ECollection Object

The **ECollection** object represents a linked list (optionally indexed) of **ECollectionMember** objects. The **ECollectionMember** represents a data value. In addition to the value, it has a set of properties for list navigation. The bracket operators ([]) can be used to create and access the indexed members of the list. Members are added to the end and beginning of the list using the PushBack and PushFront methods (or functions), respectively. You can remove members (from the front and back) with the PopFront and PopBack methods (or functions). The list is transversed using the FirstMember/LastMember properties of the **ECollection** object and the NextMember/PrevMember properties of the **ECollectionMember** object. Members can also be added to any place in the list by setting the FirstMember/LastMember properties of the ECollection object and the NextMember/PrevMember properties of the **ECollectionMember** object.

Creating the object

You create an instance of this object using the **New** command. This command creates an ECollection object. You can add members at the time it is created using a set of Value options. There is also a utility function (under eUtl) that will create an EStructure object (See “ECollection{}” on page 245.).

Format:

```
New ECollection NewVar(collVar) [Value(value1)] ... [Value(valueN)];
```

Table 86: New ECollection Options

Option Name	Option Description
ValueI	The value(s) to add to the list.
NewVar	The name of the variable to hold the newly created object.

Example:
The following script creates an ECollection object and fills it with some data.

```
. . .
New ECollection NewVar(gvCollVar) Value(100) Value('String Data');
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Run gvCollVar.PushBack Value(1000+gvIdx);
EndLoop
Set gvCollVar['Frodo'] = 50;
Set gvCollVar['Samwise'] = 38;
Set gvCollVar['Merry'] = 36;
Set gvCollVar['Pippin'] = 29;
. . .
```

ECollection Properties

Table 87: ECollection Properties

Property Name	Data Typ;e	Property Description
Count ***(Read-Only)***	**Integer**	The number of members in the list.
FirstMember	**EslObject**	The object representing the first member in the list. This is a **ECollectionMember** object. This is NULL if list is empty. Setting this property adds a new member to the list at the beginning.
LastMember	**EslObject**	The object representing the last member in the list. This is a **ECollectionMember** object. This is NULL if list is empty. Setting this property adds a new member to the list at the end.

ECollectionMember Properties

Table 88: ECollectionMember Properties

Property Name	Data Typ;e	Property Description
Value	**<Any>**	The value of the member.
Index	**<Any>**	The value of the index. NULL if no index present for this member.
NextMember	**EslObject**	The object representing the next member in the list. This is a **ECollectionMember** object. This is NULL if this is the last member in the list. Setting this property adds a new member to the list after this member.
PrevMember	**EslObject**	The object representing the previous member in the list. This is a **ECollectionMember** object. This is NULL if this is the first member in the list. Setting this property adds a new member to the list, before this member.
Parent	**EslObject**	The object representing the parent of this member. This is a **ECollection** object.

Traversing the List

A list is scanned using the FirstMember property of the ECollection object and the NextMember property of the ECollectionMember object.

Example:
The following script creates an ECollection object and fills it with some data, then scans through each member and writes the value of each to the FrameMaker console. It finishes by doing the same thing backwards.

```
New ECollection NewVar(gvCollVar) Value(100) Value('String Data');
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Run gvCollVar.PushBack Value(1000+gvIdx);
EndLoop
Set gvMember = gvCollVar.FirstMember;
Loop While (gvMember)
   write console 'Value is '+gvMember.Value;
   Set gvMember = gvMember.NextMember;
EndLoop
/* Now do it backwards */
Set gvMember = gvCollVar.LastMember;
Loop While (gvMember)
   write console 'Value is '+gvMember.Value;
   Set gvMember = gvMember.PrevMember;
EndLoop
```

Using Indexes

Indexed members and non-indexed members can co-exist in the same list. Non-indexed members simply have an index of NULL. When you add members using the PushBack and PushFront methods, a member is created without an index. For a member to have an index, you must create it using the bracket operator ([]). You can access these members by the same method.

When you set a member's value using the bracket operators, a new member is created if it does not already exist. See the following example:

Example:
The following script creates an ECollection object and fills it with some data using indexing. The list also contains two non-indexed items. Here we use the name of several fictional characters as the index and their ages at the time of the story as the data value. We access the first one directly by index and write the age of the character to the console. Then we loop through the entire collection and, if the member has an index, we write the name and the age to the console.

```
. . .
New ECollection NewVar(gvCollVar) Value(100) Value('String Data');
Set gvCollVar['Frodo'] = 50;
Set gvCollVar['Samwise'] = 38;
Set gvCollVar['Merry'] = 36;
Set gvCollVar['Pippin'] = 29;
. . .
Write console 'The Age of Frodo is '+gvCollVar['Frodo'].Value;
Set gvMember = gvCollVar.FirstMember;
Loop While (gvMember)
   If gvMember.Index
      Write console 'The Age of '+gvMember.Index+' is '+gvMember.Value;
   EndIf
   Set gvMember = gvMember.NextMember;
EndLoop
. . .
```

ECollection Object Methods

Table 89: ECollection Methods

Method Name	Method Description
PushBack	Adds one or more members to the end of the array.
PushFront	Adds one or more members to the beginning of the array.
PopBack	Returns and removes the last data member.
PopFront	Returns and removes the first data member.
FindMember	Finds the position of a data value.

ECollection Object Method Descriptions

PushBack

The PushBack method adds a new data value or list of data values to the end of the list.

Format:

```
Run eCollVar.PushBack Value(value) {Value(value2)} ... [Value(valueN)];
or
eCollVar.PushBack{value[,value2] ... [,valueN]};
```

Table 90: PushBack Options

Option Name	Option Description
ValueI	The value(s) to add to the end of the array.

Example:

The following script creates an ECollection object and fills it with some data. The first value is a Frame Object (ActiveDoc) followed by a string, then 10 numbers are added to the end in order 1001 through 1010.

```
. . .
New ECollection NewVar(gvCollVar);
Run gvCollVar.PushBack Value(ActiveDoc) Value('String Data'); // Add two values
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Run gvCollVar.PushBack Value(1000+gvIdx);
EndLoop
/* The list should be in the following order
ActiveDoc 'String Data' 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010
*/
. . .
```

PushFront

The PushFront method adds a new data value or list of data values to the beginning of the list.

Format:

```
Run eCollVar.PushFront Value(value) {Value(value2)} ... [Value(valueN)];
or
eCollVar.PushFront{value[,value2] ... [,valueN]};
```

Table 91: PushFront Options

Option Name	Option Description
ValueI	The value(s) to add to the beginning of the list. Each of these values are added as a group so they will be in the same order in the list.

Example:
The following script creates an ECollection object and fills it with some data. The loop adds 10 integers to the front of the array. Note that they go in in reverse order from the PushBack example. The end result will have 1010 as the first member in the array and 1001 as the 10 th member. This is because each successive number goes into the first position and pushes the rest up in the list.

```
. . .
New ECollection NewVar(gvCollVar);
Run gvCollVar.PushBack Value(ActiveDoc) Value('String Data'); // Add two values
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <= 10)
   Run gvCollVar.PushFront Value(1000+gvIdx);
EndLoop
. . .
/* The list should be in the following order
1010 1009 1008 1007 1006 1005 1004 1003 1002 1001 ActiveDoc 'String Data'
*/
```

PopBack

The PopBack method removes a data value from the end of the list.

Format:

```
Run eCollVar.PopBack NewVar(varName);
or
Set varName = eCollVar.PopBack{};
```

Table 92: PopBack Options

Option Name	Option Description
NewVar	The name of the variable to put the removed value.

Example:
The following script creates an ECollection object and fills it with some data. The first value is a Frame Object (ActiveDoc) followed by a string, then 10 numbers are added to the end in order 1001 through 1010. It then pops a few values from the end of the array.

```
. . .
New ECollection NewVar(gvCollVar);
Run gvCollVar.PushBack Value(ActiveDoc) Value('String Data'); // Add two values
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <= 10)
   Run gvCollVar.PushBack Value(1000+gvIdx);
EndLoop
/* The vector should be in the following order
ActiveDoc 'String Data' 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010
*/
Set gvNum1 = gvCollVar.PopBack{};
Set gvNum2 = gvCollVar.PopBack{};
Set gvNum3 = gvCollVar.PopBack{};
/* The vector should now be in the following order
   ActiveDoc 'String Data' 1001 1002 1003 1004 1005 1006 1007
and
   gvNum1 is 1010, gvNum2 is 1009 and gvNum3 is 1008
*/
. . .
```

PopFront

The PopFront method removes a data value from the beginning of the list.

Format:

```
Run eCollVar.PopFront NewVar(varName);
or
Set varName = eCollVar.PopFront{};
```

Table 93: PopFront Options

Option Name	Option Description
NewVar	The name of the variable to put the removed value.

Example:
The following script creates an ECollection object and fills it with some data. The first value is a Frame Object (ActiveDoc) followed by a string, then 10 numbers are added to the end in order 1001 through 1010. It then pops a few values from the end of the array.

```
. . .
New ECollection NewVar(gvCollVar);
Run gvCollVar.PushBack Value(ActiveDoc) Value('String Data'); // Add two values
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Run gvCollVar.PushBack Value(1000+gvIdx);
EndLoop
/* The vector should be in the following order
ActiveDoc 'String Data' 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010
*/
Set gvNum1 = gvCollVar.PopFront{};
Set gvNum2 = gvCollVar.PopFront{};
Set gvNum3 = gvCollVar.PopFront{};
/* The vector should now be in the following order
   1002 1003 1004 1005 1006 1007 1008 1009 1010
and
   gvNum1 is ActiveDoc, gvNum2 is String Data' and gvNum3 is 1001
*/
. . .
```

FindMember

The FindPos method finds the member whose data is equal to value.

Format:

```
Run eCollVar.FindMember Value(value) NewVar(member);
or
Set member = eCollVar.FindMember{value};
```

Table 94: FindMember Options

Option Name	Option Description
`value`	The data value to search for.
`NewVar`	The name of the variable to hold the ECollectionMember found. NULL if not found.

Example:
The following script builds an array of values then searches for the member with data value 1005.

```
. . .
New ECollection NewVar(gvCollVar);
Loop InitVal(1) Incr(1) LoopVar(gvIdx) While(gvIdx <=10)
   Run gvCollVar.PushBack Value(1000+gvIdx);
EndLoop
Set gvMember = gvCollVar.FindMember{1005};
. . .
/* The vector should be in the following order
   1001 1002 1003 1004 1005 1006 1007 1008 1009 1010
and
   gvPos should be the member whose value is 1005
*/
```

EStructure Object

The **EStructure** object represents a structure or group of named members. The **EStructure** object represents a typical structure (or group of named members) where the properties are the data names. This is similar to a struct in C or a Record in Pascal. Of course, with ElmScript you do not define the structure ahead of time. You may add member names and values as you like.

Creating the object

You create an instance of this object using the **New** command. This command creates an EStructure object. You can add members at the time it is created using a set of named options. There is also a utility function (under eUtl) that will create an EStructure object (See "EStructure{}" on page 245.).

Format:

```
New EStructure NewVar(structVar) [Name1(value1)] ... [NameN(valueN)];
```

Table 95: New EStructure Options

Option Name	Option Description
NameI	The member name (s) to add to the structure.
ValueI	The value(s) to associate with the corresponding member name.
NewVar	The name of the variable to hold the newly created object.

Example:

The following script creates an EStructure object and adds some member names to it with some string data.

```
. . .
New EStructure NewVar(myStruct);
Set myStruct.FirstName = 'John';
Set myStruct.LastName = 'Doe';
Set myStruct.Address = '345 Park Avenue';
Set myStruct.City = 'San Jose';
Set myStruct.State = 'California';
. . .
```

EStructure Properties

Table 96: EStructure Properties

Property Name	Data Type	Property Description
Count	**Integer**	The number of members in the structure.
MemberNameList	**StringList**	A string list containing the names of each member name defined in the structure.
MemberName	**Various**	A name of a member in the structure. This property returns the current value of the member.

EStructure Object Methods

Table 97: EStructure Methods

Method Name	Method Description
AddMember	Adds one or more members to the structure.
RemoveMember	Removes one or more members from the structure.

EStructure Object Method Descriptions

AddMember

The AddMember method adds one or more new members to the structure.

Format:

```
Run eStructVar.AddMember FieldName1(value1) {FieldName2(value2)}
      ... [FieldNameN(valueN)];
```

Table 98: AddMember Options

Option Name	Option Description
FieldNameI	The name (s) of the new member to add to the structure.
ValueI	The value(s) to add to the corresponding member name.

Example:

The following script creates an EStructure object and adds some member names to it with some string data, then it adds some members using the AddMember method.

```
. . .
New EStructure NewVar(myStruct);
Set myStruct.FirstName = 'Jonathan';
Set myStruct.LastName = 'Doe';
Set myStruct.Address = '345 Park Avenue';
Set myStruct.City = 'San Jose';
Set myStruct.State = 'California';
Run myStruct.AddMember ZipCode('99999') NickName('Jon');
. . .
```

RemoveMember

The RemoveMember method removes one or more members from the structure.

Format:

```
Run eStructVar.RemoveMember FieldName1 {FieldName2} ... [FieldNameN];
```

Table 99: RemoveMember Options

Option Name	Option Description
FieldNameI	The name (s) of the members to delete from the structure.

Example:
The following script creates an EStructure object and adds some member names to it with some string data, then it adds some members using the AddMember method, then removes the NickName member later on.

```
. . .
New EStructure NewVar(myStruct);
Set myStruct.FirstName = 'Jonathan';
Set myStruct.LastName = 'Doe';
Set myStruct.Address = '345 Park Avenue';
Set myStruct.City = 'San Jose';
Set myStruct.State = 'California';
Run myStruct.AddMember ZipCode('99999') NickName('Jon');
. . .
Run MyStruct.RemoveMember NickName;
```

EStructure Index Access

You can use the index operator to access members of a structure. Each member is numbered from 1 to the number of members in the structure. Indexed access will only give you the values, not the associated member names. For example:

```
Set myValue = myStruct[4];
```

gets the value of the fourth member of the structure.

The following subroutine writes member information for each member of a structure to the FrameMaker console.

```
RUN DumpStruct pvMsg('Dump Structure ') ByRef pvStrVar(MyStruct);
   ...

Sub DumpStruct using pvMsg pvStrVar
   Local lvCount lvFieldNum;

   Write console 'Dump EStructure-'+pvMsg;
   Set lvCount = pvStrVar.Count;
   Write console 'Count('+lvCount+')';
   Loop LoopVar(lvFieldNum) Incr(1) InitVal(1) While(lvFieldNum <= lvCount)
      Write console 'Field('+lvFieldNum+')  Value('+pvStrVar[lvFieldNum]+')';
   EndLoop

EndSub
```

Chapter 5

EDateTime Object

The EDateTime object represents a date and time value. This is a volatile object. You must create an instance of this object to use it. This object can be used to do date and time calculations.

Creating the object

You create an instance of this object using the NEW command. You supply the value of the date EDateTime object using the Value option.

Format:

```
New EDateTime NewVar(edtVar) [Value(stringValue)];
```

Table 100: NEW EDateTime Options

Option Name	Option Description
`NewVar`	The name of the variable to hold the newly created EDateTime object.
`Value`	The value of the object. This can be a string value, an integer value or another EDateTime object. An integer value indicates only the year. A string value can be 'CurrTime' or any standard string representation of a date-time value such as 'Jan 4, 1989 09:12:33'

Example:

The following script command creates an EDateTime object with the current date and time as its value.

```
. . .
New EDateTime NewVar(edtVar) Value('CurrTime');
. . .
```

Example:

The following script creates an EDateTime object with a specified date time value, then creates another EDateTime object using the first as its value.

```
. . .
New EDateTime NewVar(edtVar) Value('Jul 22, 1999');
New EDateTime NewVar(edt2Var) Value(edtVar);
. . .
```

Deleting the object

To delete an instance of this object use the DELETE Object command. Delete the object when you do not need it anymore.

Delete Object

The Delete method deletes the `EDateTime` object.

Format:

```
Delete Object(edtVar);
```

EDateTime Properties

Table 101: EDateTime Properties

Property Name	Data Type	Property Description
ErrorCode	**Integer**	The ElmScript error code for the last method.
ErrorMsg	**String**	The Text of the last error.
Value	**String**	The string value in the format YYYYMMDDHHmmSS, where YYYY is the year, MM is the month number, DD is the day number, HH is the hour, mm is the minute, and SS is the second..
Year	**Integer**	The year value.
Month	**Integer**	The Month number.
Day	**Integer**	The day number within the month.
Hour	**Integer**	The hour number.
Minute	**Integer**	The minute number.
Second	**Integer**	The second number.
DayOfWeek	**Integer**	The number of the day of the week (1-Sunday).
DayOfYear	**Integer**	The day of the year.
Status	**Integer**	The validity status, 0-valid, 1-Invalid, -1-NULL.
DateString	**String**	A formatted version of the date-time value..

EDateTime Methods

Table 102: EDateTime Methods

Method Name	Method Description
New	Creates the EDateTime object.
Delete	Deletes the EDateTime object.
SetCurrTime	Sets the value of the object to the current date and time.
Format	Returns a customized formatted string of the EDateTime value.

Table 102: EDateTime Methods

Method Name	Method Description
Add	Adds an ETimeSpan value to the EDateTime value.
Subtract	Subtracts another EDateTime object and produces an ETimeSpan object.
SetFromFileName	Sets the value of this EDateTime to a date from a file name.

EDateTime Method Descriptions

SetCurrTime

The SetCurrTime command sets the value of the EDateTime object to the current time.

Format:

```
Run edtVar.SetCurrTime;
```

Example:

The following script creates an `EDateTime` object with a specified date, then changes it to the current date and time.

```
. . .
New EDateTime NewVar(edtVar) Value('Jul 22, 1999');
Run edtVar.SetCurrTime;
. . .
```

Format

The Format command formats the value of the EDateTime object into a customized formatted string. You can specify what items to include and the order in which they appear.

Format:

```
Run edtVar.Format FormatStr(fmtString) [Lang(langString)] NewVar(strValue);
```

Table 103: Format Options

Option Name	Option Description
FormatStr	A string value specifying the format of the output string. This string can contain coded symbols (those beginning with %), which represent parts of the date-time value. Characters that do not begin with a % character are copied to the output string as they are. .See the next table labeled “Coded Date Time Characters” on page 82. You can optionally include a special flag (#) between the % and the code to modify the output. See the table for more information.
Lang	An optional language string for international date time formats. See the table labeled “Language Strings” on page 82 for the possible values (use Column 3 as one of the values for this option)..
NewVar	The formatted output string.

Table 104: Coded Date Time Characters

Option Name	Option Description
`%a`	The abbreviated weekday name. .
`%A`	The full weekday name.
`%b`	The abbreviated month name. .
`%B`	The full month name.
`%c or %#c`	Locale specific date and time representation. The # indicates a long date/time representation.
`%d or %#d`	The day number of the month. The # removes leading zeroes.
`%H or %#H`	Hour (in 24-hour format, 0-23) The # removes leading zeroes.
`%I or %#I`	Hour (in 12-hour format, 1-12). The # removes leading zeroes.
`%j or %#j`	Day of the year (1-366) The # removes leading zeroes.
`%m or %#m`	Month number (1-12). The # removes leading zeroes.
`%M or %#M`	Minute number (0-59). The # removes leading zeroes.
`%p`	Current locale's A.M./P.M. indicator for 12-hour clock.
`%S or %#S`	Second Number (0-59). The # removes leading zeroes.
`%U or %#U`	Week of the year (0-53) with Sunday as the first day of the week. The # removes leading zeroes.
`%w or %#w`	Weekday number (0-6), Sunday is 0. The # removes leading zeroes.
`%W or %#W`	Week of the year (0-53) with Monday as the first day of the week. The # removes leading zeroes.
`%x or %#x`	Date representation for the current locale. The # option indicates a long date representation.
`%X`	Time representation for the current locale.
`%y or %#y`	Year number without century (0-99). The # removes leading zeroes.
`%Y or %#Y`	Year number with century. The # removes leading zeroes.
`%z or %Z`	Time zone name or abbreviation, if applicable. The # removes leading zeroes.
`%%`	Inserting the % character into the output string.

Table 105: Language Strings

Primary Language	Sub Language	Language String
Chinese	Chinese	'chinese'
Chinese	Chinese (simplified)	'chinese-simplified' or 'chs'
Chinese	Chinese (traditional)	'chinese-traditional' or 'cht'
Czech	Czech	'csy' or 'czech'
Danish	Danish	'dan'or 'danish'

Table 105: Language Strings

Primary Language	Sub Language	Language String
Dutch	Dutch (Belgian)	'belgian', 'dutch-belgian', or 'nlb'
Dutch	Dutch (default)	'dutch' or 'nld'
English	English (Australian)	'australian', 'ena', or 'english-aus'
English	English (Canadian)	'canadian', 'enc', or 'english-can'
English	English (default)	'english'
English	English (New Zealand)	'english-nz' or 'enz'
English	English (UK)	'eng', 'english-uk', or 'uk'
English	English (USA)	'american', 'american english', 'american-english', 'english-american', 'english-us', 'english-usa', 'enu', 'us', or 'usa'
Finnish	Finnish	'fin' or 'finnish'
French	French (Belgian)	'frb' or 'french-belgian'
French	French (Canadian)	'frc' or 'french-canadian'
French	French (default)	'fra'or 'french'
French	French (Swiss)	'french-swiss' or 'frs'
German	German (Austrian)	'dea' or 'german-austrian'
German	German (default)	'deu' or 'german'
German	German (Swiss)	'des', 'german-swiss', or 'swiss'
Greek	Greek	'ell' or 'greek'
Hungarian	Hungarian	'hun' or 'hungarian'
Icelandic	Icelandic	'icelandic' or 'isl'
Italian	Italian (default)	'ita' or 'italian'
Italian	Italian (Swiss)	'italian-swiss' or 'its'
Japanese	Japanese	'japanese' or 'jpn'
Korean	Korean	'kor' or 'korean'
Norwegian	Norwegian (Bokmal)	'nor' or 'norwegian-bokmal'
Norwegian	Norwegian (default)	'norwegian'
Norwegian	Norwegian (Nynorsk)	'non' or 'norwegian-nynorsk'
Polish	Polish	'plk' or 'polish'

Table 105: Language Strings

Primary Language	Sub Language	Language String
Portuguese	Portuguese (Brazilian)	'portuguese-brazilian' or 'ptb'
Portuguese	Portuguese (default)	'portuguese' or 'ptg'
Russian	Russian (default)	'rus' or 'russian'
Slovak	Slovak	'sky' or 'slovak'
Spanish	Spanish (default)	'esp' or 'spanish'
Spanish	Spanish (Mexican)	'esm' or 'spanish-mexican'
Spanish	Spanish (Modern)	'esn' or 'spanish-modern'
Swedish	Swedish	'sve' or 'swedish'
Turkish	Turkish	'trk' or 'turkish'

Example 1:
The following script creates an `EDateTime` object, then writes the date and time out in various formats to the FrameMaker console.

```
. . .
NEW EDateTime NewVar(edtVar) Value('Jul 22, 1999 9:56:12');
write console 'Date String-'+edtVar.DateString;
write console 'Status-'+edtVar.Status+' Value-'+edtVar.Value;
RUN edtVar.Format FormatStr('%A %B %d, %Y at %H:%M:%S %Z') NewVar(oStr);
Write Console oStr; // This will write the following to the frame console:
// Date String-7/22/1999 9:56:12 AM
// Status-0 Value-19990722095612
// Thursday July 22, 1999 at 09:56:12 Eastern Daylight Time
. . .
```

Example 2:
The following script creates an `EDateTime` object from the current date, then writes the date and time out in various international formats to the FrameMaker console.

```
. . .
New EDateTime NewVar(gvDate) Value('CurrTime');
Run gvDate.Format FormatStr('%#c') Lang('German') NewVar(gvStrDate);
Write Console 'German DateTime(Long)-'+gvStrDate;
Run gvDate.Format FormatStr('%B--%c') Lang('esn') NewVar(gvStrDate);
Write Console 'Spanish Month and DateTime(Short)-'+gvStrDate;
Run gvDate.Format FormatStr('%A') Lang('portuguese') NewVar(gvStrDate);
Write Console 'Portuguese Day-'+gvStrDate;
Run gvDate.Format FormatStr('%B et %A') Lang('french') NewVar(gvStrDate);
Write Console 'French Month and Day-'+gvStrDate;
//     Here is the output.
// German DateTime(Long)-Donnerstag, 14. Oktober 2004 12:01:00
// Spanish Month and DateTime(Short)-octubre--14/10/2004 12:01:00
// Portuguese Day-quinta-feira
// French Month and Day-octobre et jeudi
```

Add

The Add command adds a time span value to the date.

Format:

```
Run edtVar.Add Value(timeSpanValue);
```

Table 106: Add Options

Option Name	Option Description
Value	An expression for a time span. See "ETimeSpan Properties" on page 90 for more information. If you specify an integer value, it will be a number of days.

Example:

The following script creates an `EDateTime` object, then adds fives days to the initial date before writing the value to the FrameMaker console.

```
. . .
NEW EDateTime NewVar(edtVar) Value('Jul 22, 1999 9:56:12');
RUN edtVar.Add Value(5);
write console 'Date String-'+edtVar.DateString;
// Date String-7/27/1999 9:56:12 AM
. . .
```

Example:

The following script creates an `EDateTime` object, then adds a time span of 21 hours and 10 minutes to the initial date/time before writing the value to the FrameMaker console.

```
. . .
NEW EDateTime NewVar(edtVar) Value('Jul 22, 1999 9:56:12');
NEW ETimeSpan NewVar(espanVar) Value('21:10:00');
RUN edtVar.Add Value(espanVar);
write console 'Date String-'+edtVar.DateString;
// Date String-7/23/1999 7:06:12 AM
. . .
```

Subtract

The Subtract command subtracts a time span value from the date. The second format of the command subtracts another `EDateTime` object from the current object and produces an ETimeSpan object.

Format:

```
Run edtVar.Subtract Value(timeSpanValue);
or
Run edtVar.Subtract Value(dateTimeValue) NewVar(timeSpanVar);
```

Table 107: Subtract Options

Option Name	Option Description
Value	An expression for a time span. See "ETimeSpan Properties" on page 90 for more information. If you specify an integer value, it will be a number of days.
NewVar	For the second format above, you specify the variable name for the resulting ETimeSpan object.

Example:
The following script creates an `EDateTime` object, then adds fives days to the initial date before writing the value to the FrameMaker console.

```
. . .
NEW EDateTime NewVar(edtVar) Value('Jul 22, 1999 9:56:12');
RUN edtVar.Subtract Value(5);
write console 'Date String-'+edtVar.DateString;
// Date String-7/17/1999 9:56:12 AM
. . .
```

Example:
The following script creates an `EDateTime` object, then subtracts a time span of 21 hours and 10 minutes from the initial date/time before writing the value to the FrameMaker console.

```
. . .
NEW EDateTime NewVar(edtVar) Value('Jul 22, 1999 9:56:12');
NEW ETimeSpan NewVar(espanVar) Value('21:10:00');
RUN edtVar.Subtract Value(espanVar);
write console 'Date String-'+edtVar.DateString;
// Date String-7/21/1999 12:46:12 PM
. . .
```

Example:
The following script creates two `EDateTime` objects, then subtracts one from another, before writing the resulting ETimeSpan value to the FrameMaker console.

```
. . .
NEW EDATETIME NewVar(edtVar) Value('Jul 22, 1999 9:56:12');
NEW EDATETIME NewVar(edt2Var) Value('Aug 22, 1999 9:56:12');
RUN edtVar.Subtract Value(edt2Var) NewVar(espanVar);
WRITE console '   Time String-->'+espanVar.TimeString;
// Time String-->30(19:28:47)
. . .
```

SetFromFileName

The SetFromFileName command sets the date and time object from the one of the date-time values in the file label of the specified file name.

Format:

```
Run edtVar.SetFromFileName FileName(filenameStr) DateType(typeString);
```

Table 108: SetFromFileName Options

Option Name	Option Description
`FileName`	The filename that provides the source for the date information.

`DateType`	This is a string value which indicates which of the three date fields from the file label to use.
	`'CREATE' - The file creation date`
	`'ACCESS' - The file last access date.`
	`'MOD' - The file last modification date`

Example:
The following script creates an `EDateTime` object, then sets the date information from the creation date of the specified file name, before writing the value to the FrameMaker console.

```
. . .
NEW EDateTime NewVar(edtVar);
RUN edtVar.SetFromFileName FileName('c:\MyDir\MyFile.dat')
   DateType('CREATE');
write console 'File Creation Date String-'+edtVar.DateString;
. . .
```

Chapter 6

ETimeSpan Object

The ETimeSpan object represents a time value, not an absolute time, but a length of time. You can use ETimeSpan objects by themselves or in combination with EDateTime objects. This is a volatile object. To use these methods, you must first create an ETimeSpan object using the NEW ETimeSpan command.

Creating the object

Using the NEW command to craete an instance of the ETimeSpan object. You supply the value of the date ETimeSpan object using the Value option.

Format:

```
New ETimeSpan NewVar(espanVar) Value(stringValue);

or

New ETimeSpan NewVar(espanVar) Days(dayCount) Hours(hourCount) Minutes(minuteCount)
       Seconds(secondCount);
```

Table 109: NEW ETimeSpan Options

Option Name	Option Description
NewVar	The name of the variable to hold the newly created ETimeSpan object.
Value	The value of the object. This can be a string value, or another ETimeSpan object. A string value can be 'Clear' or any standard string representation of a time value such as '11:21:54'. To set the Day count, you have to use the second form of the command
Days	The number of Days.
Hours	The number of Hours.
Minutes	The number of Minutes.
Seconds	The number of Seconds.

Example:
The following script command creates an ETimeSpan object with a zero value.

```
. . .
New ETimeSpan NewVar(espanVar) Value('Clear');
. . .
```

Example:
The following script creates an ETimeSpan object with a specified time span value, then creates another ETimeSpan object using the first as its value.

```
. . .
New ETimeSpan NewVar(espanVar) Value('11:43:05');
New ETimeSpan NewVar(span2Var) Value(espanVar);
. . .
```

Example:
The following script creates an ETimeSpan object by specifying the number of days, hours, minutes and seconds of the time span.

```
. . .
New ETimeSpan NewVar(espanVar) Days(4) Hours(15) Minutes(33) Seconds(22);
. . .
```

Deleting the object

The DELETE Object command deletes an instance of this object. Delete the object when you do not need it anymore.

Delete

The Delete method deletes the `ETimeSpan` object.

Format:

```
Delete Object(espanVar);
```

ETimeSpan Properties

l

Table 110: ETimeSpan Properties

Property Name	Data Type	Property Description
ErrorCode	**Integer**	The ElmScript error code for the last method.
ErrorMsg	**String**	The Text of the last error.
Value	**String**	The string value in the format DDHHMMSS, where DD is the day number, HH is the hour, MM is the minute, and SS is the second..
Days	**Integer**	The number of days
Hours	**Integer**	The hour number.
Minutes	**Integer**	The minute number.
Seconds	**Integer**	The second number.
TotalDays	**Integer**	The total number of days in the entire time span.
TotalHours	**Integer**	The total number of hours in the entire time span.
TotalMinutes	**Integer**	The total number of minutes in the entire time span.

Table 110: ETimeSpan Properties

Property Name	Data Type	Property Description
TotalSeconds	**Integer**	The total number of seconds in the entire time span..
Status	**Integer**	The validity status, 0-valid, 1-Invalid, -1-NULL.
TimeString	**String**	A formatted version of the time span value..

ETimeSpan Methods

Table 111: ETimeSpan Methods

Method Name	Method Description
New	Creates the ETimeSpan object.
Delete	Deletes the ETimeSpan object.
Clear	Clears the time span value.
Format	Returns a customized formatted string of the ETimeSpan value.
Add	Adds an ETimeSpan value to the ETimeSpan value.
Subtract	Subtracts another ETimeSpan object and produces an ETimeSpan object.

ETimeSpan Method Descriptions

Clear

The Clear command resets the value of the ETimeSpan object to zero.

Format:

```
Run espanVar.Clear;
```

Example:

The following script creates an `ETimeSpan` object with a specified time span, then clears the value to zero.

```
. . .
New ETimeSpan NewVar(espanVar) Value('11:21:00');
RUN espanVar.Clear;
. . .
```

Format

The Format command formats the value of the ETimeSpan object into a customized formatted string. You can specify what items to include and the order in which they appear.

Format:

```
Run espanVar.Format FormatStr(fmtString) NewVar(strValue);
```

Table 112: Format Options

Option Name	Option Description
FormatStr	A string value specifying the format of the output string. This string can contain coded symbols (those beginning with %),which represent parts of the time span value. Characters that do not begin with a % character are copied to the output string as they are. See the next table labeled "Coded Time Span Characters" on page 92. You can optionally include a special flag (#) between the % and the code to modify the output. See the table for more information.
NewVar	The formatted output string.

Table 113: Coded Time Span Characters

Option Name	Option Description
%D	The day number.
%H	Hour (in 24-hour format, 0-23).
%M	Minute number (0-59).
%S	Second Number (0-59).
%%	Inserting the % character into the output string.

Example:

The following script creates an `ETimeSpan` object, then writes the time span out in various formats to the FrameMaker console.

```
. . .
NEW ETimeSpan NewVar(espanVar) Days(55) Hours(11) Minutes(21) Seconds(44);
WRITE console '   Time Span Value-'+espanVar.Value;
WRITE console '   Formatted Time Span Value-'+espanVar.TimeString;
RUN espanVar.Format FormatStr('Days-%D The Rest-%H:%M:%S')
      NewVar(oStr);
write console oStr; // This will write the following to the frame console:
// Time Span Value-0055112144
// Formatted Time Span Value-55(11:21:44)
// Days-55 The Rest-11:21:44
. . .
```

Add

The Add command adds another time span value to the time span object.

Format:

```
Run espanVar.Add Value(timeSpanValue);
```

Table 114: Add Options

Option Name	Option Description
`Value`	An expression for a time span. If you specify an integer value, it will be a number of days.

Example:

The following script creates an `ETimeSpan` object, then adds five days to the initial value then adds a different time span to the resulting value, displaying the results of both to the FrameMaker console.

```
. . .
NEW ETimeSpan NewVar(espanVar) Days(55) Hours(11) Minutes(21) Seconds(44);
RUN espanVar.Add Value(5);
write console 'Time Span After Adding 5 Days--->'+espanVar.TimeString;
// Time Span After Adding 5 Days--->60(11:21:44)
NEW ETimeSpan NewVar(espan2Var) Days(5) Hours(11) Minutes(21) Seconds(44);
RUN espanVar.Add Value(espan2Var);
WRITE console '   Time Span After Add Time Span('+espan2Var.TimeString+
          ')--->'+espanVar.TimeString;
// Time Span After Add Time Span(5(11:21:44))--->65(22:43:28)
. . .
```

Subtract

The Subtract command subtracts a time span value another time span.

Format:

```
Run espanVar.Subtract Value(timeSpanValue);
```

Table 115: Subtract Options

Option Name	Option Description
`Value`	An expression for a time span. If you specify an integer value, it will be a number of days.

Example:

The following script creates an `ETimeSpan` object, then subtracts five days to the initial value then subtracts a different time span to the resulting value, displaying the results of both to the FrameMaker console.

```
. . .
NEW ETimeSpan NewVar(espanVar) Days(55) Hours(11) Minutes(21) Seconds(44);
RUN espanVar.Subtract Value(5);
WRITE console '   Time Span After Subtract 5 Days--->'+espanVar.TimeString;
RUN espanVar.Subtract Value(espan2Var);
WRITE console '   Time Span After Subtract Time Span('+espan2Var.TimeString+
          ')--->'+espanVar.TimeString;
. . .
```

Chapter 7

EFileInfo Object

The EFileInfo object represents a information in a file's label. This is a volatile object. To use these methods, you must first create an EFileInfo object using the NEW EFileInfo command.

Creating the object

You create an instance of this object using the NEW command. You supply the value of the date EFileInfo object using the Value option.

Format:

```
New EFileInfo NewVar(einfo) FileName(filePattern);
```

Table 116: New EFileInfo Options

Option Name	Option Description
NewVar	The name of the variable to hold the newly created EFileInfo object.
FileName	This identifies the file name or file group. You may specify a single file name or a file pattern with wildcards indicating a file group[. The GetNext method lets you go to the next file in the group.

Example 1:

The following script command creates an EFileInfo object with the current date and time as its value.

```
. . .
New EFileInfo NewVar(einfo) FileName(ClientDir+'\fscript.ini');
. . .
```

Example 2:

The following script creates an EFileInfo object for a group of files (all the files in the product client directory) and writes the name of each file to the FrameMaker console.

```
. . .
New EFileInfo NewVar(einfo) FileName(ClientDir+'\*.*');
Loop While(einfo.FilePresent)
   Write console 'File name-'+einfo.FileName;
   Run einfo.GetNext;
EndLoop
Delete Object (einfo);
. . .
```

Deleting the object

The DELETE Object command will delete the object. Delete the object when you do not need it anymore.

Delete Object

The Delete method deletes the `EFileInfo` object.

Format:

```
Delete Object(einfo);
```

EFileInfo Properties

Table 117: EFileInfo Properties

Property Name	Data Type	Property Description
ErrorCode	**Integer**	The ElmScript error code for the last method.
ErrorMsg	**String**	The Text of the last error.
FileName	**String**	The name of the file without path information.
FileLength	**Integer**	The length of the file in bytes.
FileTitle	**String**	Gets the title of the file with path information nor file extension.
FileRoot	**String**	The Root of the file name. The directory path without the file name
FilePath	**String**	The Full Path of the file.
FileURL	**String**	Gets the URL of the file name. The Full path of the file in the form of a URL
CreateTime	**EDateTime**	The date and time the file was created.
AccessTime	**EDateTime**	The date and time the file was last accessed.
ModTime	**EDateTime**	The date and time the file was last modified.
FilePresent	**Integer**	True if the file label is present, False otherwise.
IsDots	**Integer**	Determines if the name of the found file has the name "." or "..", indicating that is actually a directory. True if this is a dot directory, False otherwise.
IsValid	**Integer**	True if the file information is valid, False otherwise.
IsReadOnly	**Integer**	True if the file is read-only, False otherwise.
IsDirectory	**Integer**	True if this is a directory, False otherwise.
IsCompressed	**Integer**	True if the file is compressed, False otherwise.
IsSystem	**Integer**	True if this is a system file, False otherwise.
IsHidden	**Integer**	True if this is a hidden file, False otherwise.
IsTemporary	**Integer**	True if the file is a temporary file, False otherwise.

Table 117: EFileInfo Properties

Property Name	Data Type	Property Description
IsNormal	**Integer**	True if this is a normal file (no other flags set), False otherwise.
IsArchived	**Integer**	True if this is an archived file, False otherwise.

EFileInfo Methods

Table 118: EFileInfo Methods

Method Name	Method Description
New	Creates the EFileInfo object.
Delete	Deletes the EFileInfo object.
GetNext	Gets the next file in a file group.
Refresh	Reloads the information from the file label.

EFileInfo Method Descriptions

GetNext

The GetNext command gets the next file in the file group (if any). The FilePresent property indicates the end of the file group

Format:

```
Run eFileInfo.GetNext;
```

Example:

See “Example 2:” on page 95.

Refresh

The GetNext command refreshes the information in the current file label.

Format:

```
Run eFileInfo.Refresh;
```

Chapter 8

Database Objects

Introduction

These objects allows you to retrieve and update data in standard ODBC compliant databases via ElmScript scripts. You may use SQL commands to perform searches and get results set, from which you may access data variables the same way you would use standard ElmScript variables. You can also use the SqlCommand method (if the database is opened for update) to issue SQL commands directly to the database manager to insert, update and delete records from database tables.

eODBCInfo Object

The eODBCInfo object represents the ODBC system as a whole. This is a permanent object. You do not create nor delete it. This object provides information on the available drivers and data sources.

EDB Object

The EDB object represents a database connection. You use this object to identify, open, close and issue SQL commands to the database. You supply the object with a data source name (defined when you register your database to the ODBC manager (see below), and optionally a User ID and Password. Then you connect to the database using the Open method (a subroutine call). The Close method terminates the connection. You can also open a database by specifying a connection string. This allows you to supply database specific options.

EQuery Object

The EQuery object represents a query and search result. Each EQUERY object is associated with an EDB object. After creating an EQuery object, you set its Select property to the SQL search that you wish, then you run the RunQuery method to perform the search. You access the Fields property to get the data from the current record and you may use the GetNext method to move forward to the next record in the results set. You can also use the EQuery object to run special queries to get information about the tables in the database and the columns (Fields) in a table.

Connecting to a Database

We use the term ’Open a Database’ to get access to it, because it is a familiar concept. We open documents and text files all the time in various programs. With databases though a more accurate term would be ’Make a Database Connection’. You generally need to supply more than just the file name to access (generally speaking) a database. With the ODBC system, you need to specify the driver information as well.

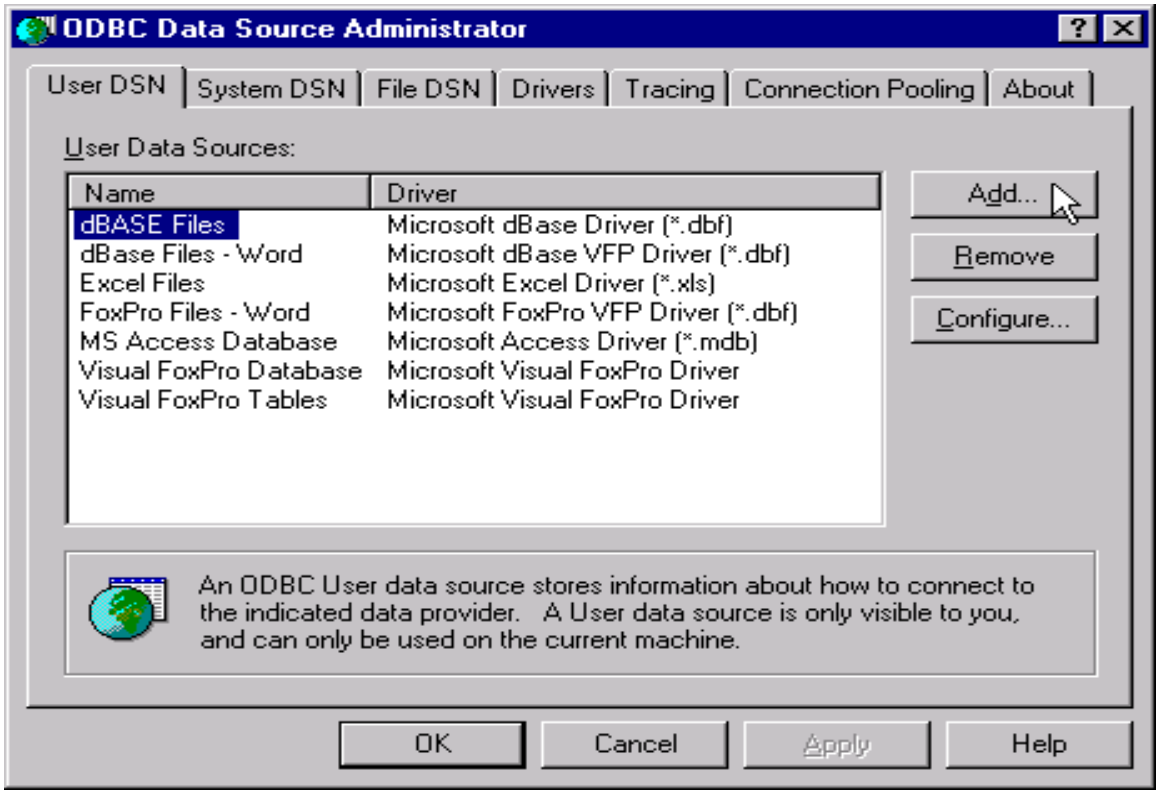
There are two ways to open a database (make a connection) with ODBC databases. The most common way is to register a database with the ODBC Administrator. Then you will be able to specify the Data Source name on the ElmScript EDB open command. The other way is to provide a connection string with all the information ODBC needs to make the connection. The connection string option is somewhat database type dependent. In this case, you would use the ConnectString property of the EDB object to specify all the parameters needed.

Registering a Database via ODBC

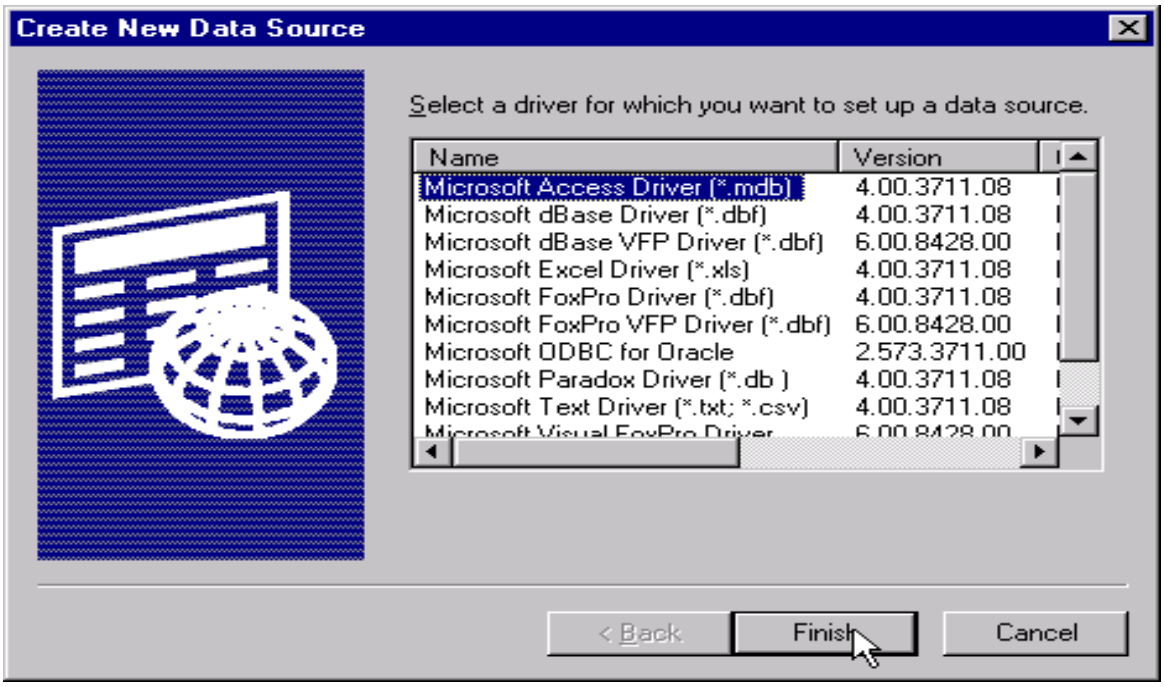
To register a database with the ODBC Manager you need to bring up the ODBC Data Sources dialog. As an example, use the following steps to register a data source for the Demo Database (MS Access) database (Issues.mdb):

1. Bring up the ODBC manager.

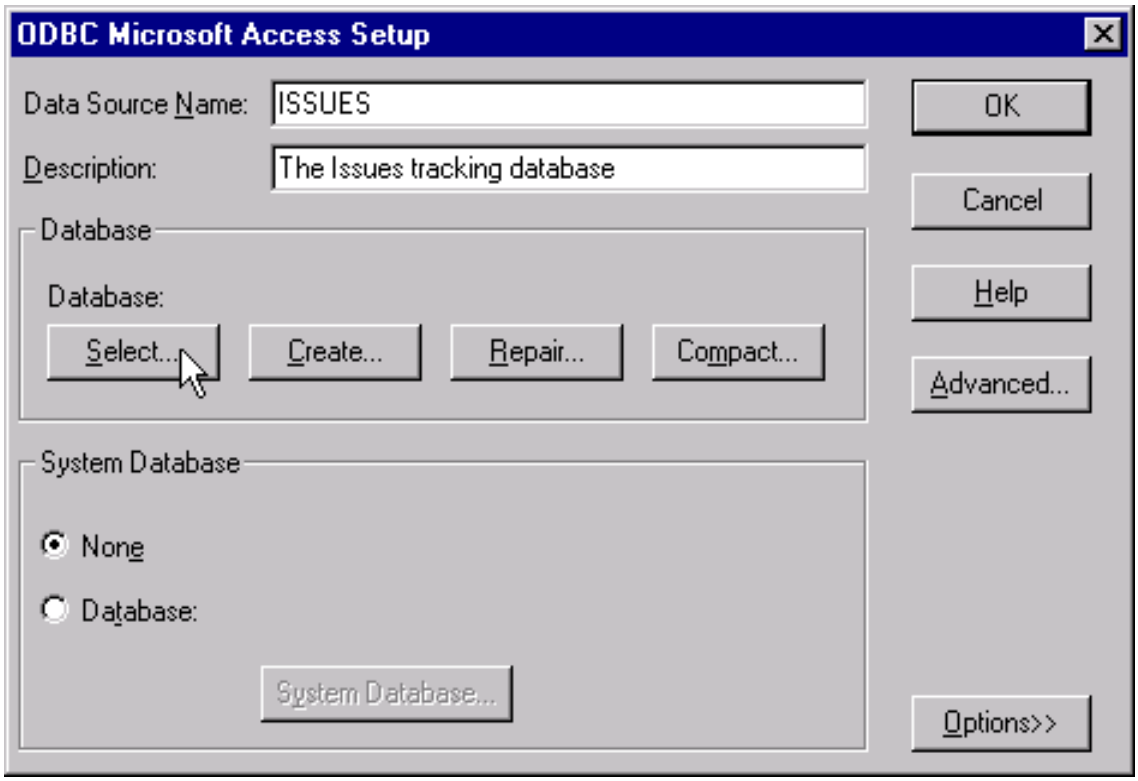
   Click on the Start->Control Panel. When the Control panel comes up, double click on the ODBC icon and the following window (or one very much like it) should appear. With Windows 2000 and XP, the ODBC icon appears under the Administrative Tools folder in the Control panel.



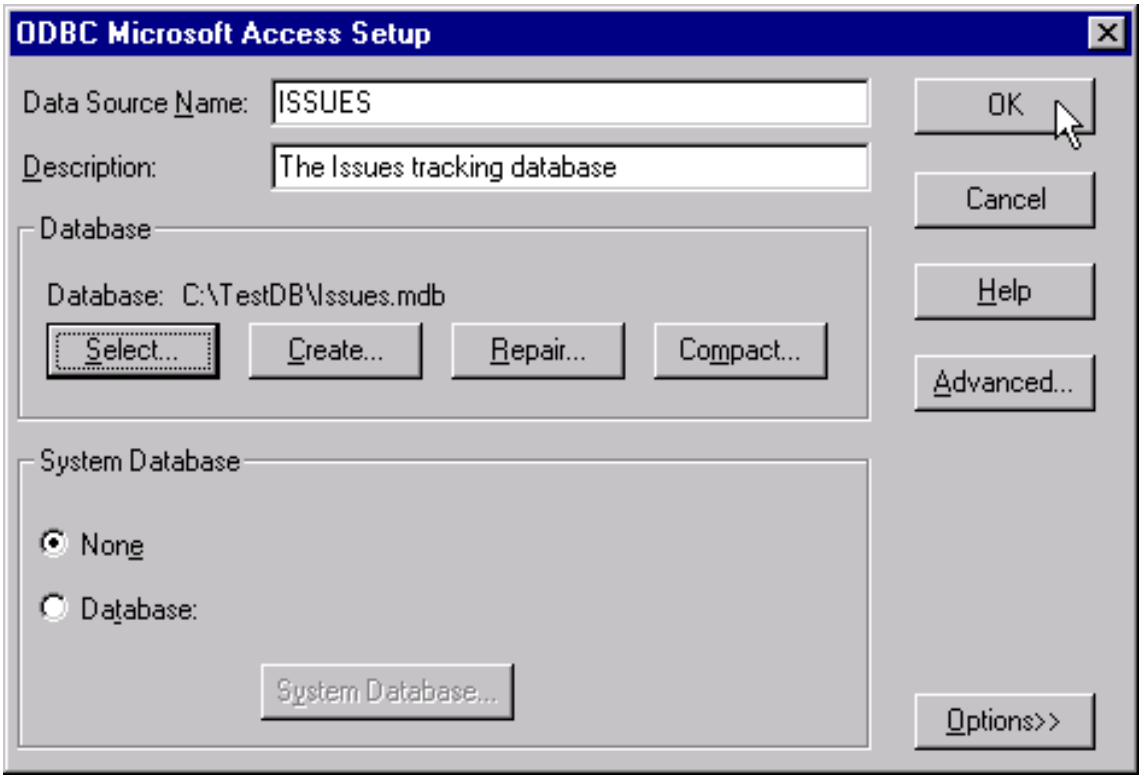
2. Click on the Add button to add a new data source. The following window should appear.



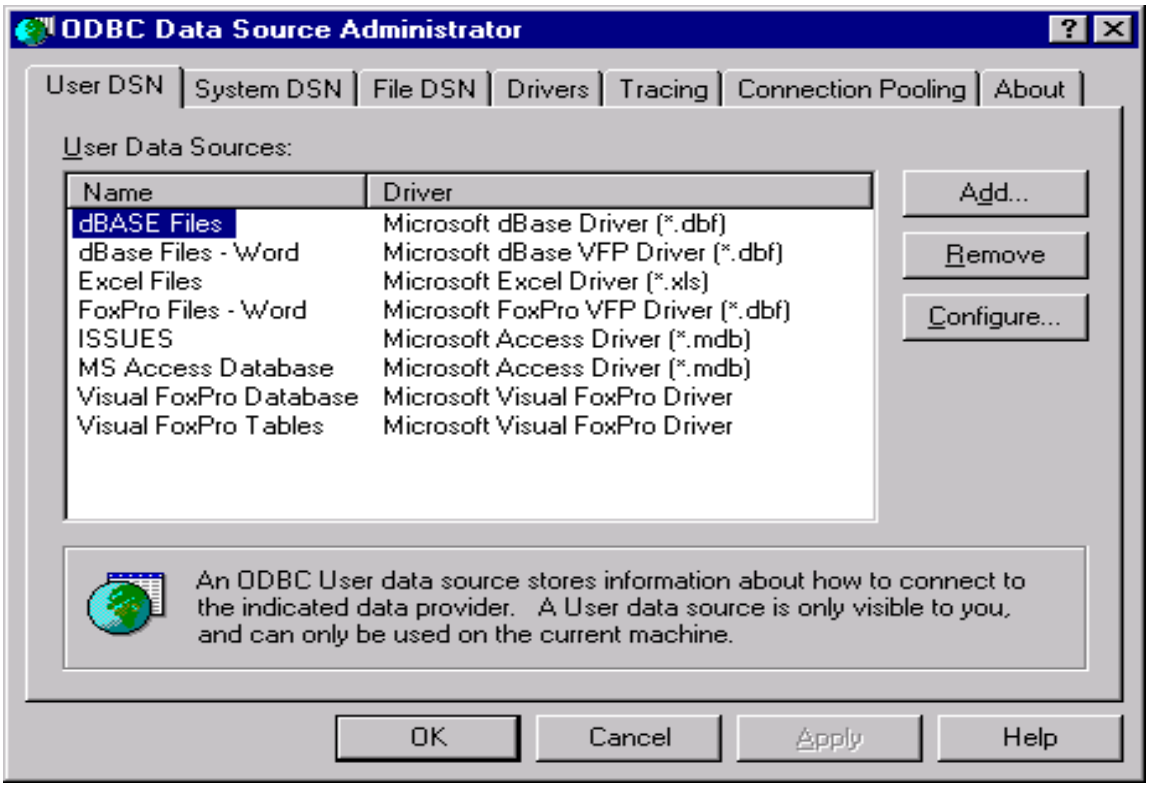
3. Select the driver for the database (in this case, it is Microsoft Access) and press Finish. The Microsoft Access Setup dialog will appear. Enter the name of the data source (in this case: ISSUES). This name is very important because it is used to identify the data source when you make a data connection in ElmScript. To run the demo database scripts make sure that you use the name ISSUES. The demo scripts expect a data source named ISSUES to exist. For your own databases, you can choose whatever name that you like. You may also optionally enter a description.



4. Press the Select button to choose the database file. The standard file selection dialog will appear. Go to the ElmScript directory and the DemoDatabase sub-directory and select the Issues.mdb file and press OK. The setup window should be updated as follows:



5. Press the OK to accept all the changes. The following window appears.



6. The issues database now appears in the main window. Press OK to finish.

You can now specify the ISSUES Data source name to access this database on the Open Command or New Command.

Example:

```
New EDB DataSource('ISSUES') NewVar(testdb);
Run testdb.Open;
```

Using the ConnectString option

The other way to make a database connection is to use the ConnectString option. Instead of giving a Data Source name, you need to specify the ODBC driver name along with some other database type specific attributes. For example, to open an MS Access database, you need to supply the driver name and the name of the database (.mdb) file. For example, if the demo database (issues.mdb) were located in the C:\TestDb\ directory, you could open it with the following commands:

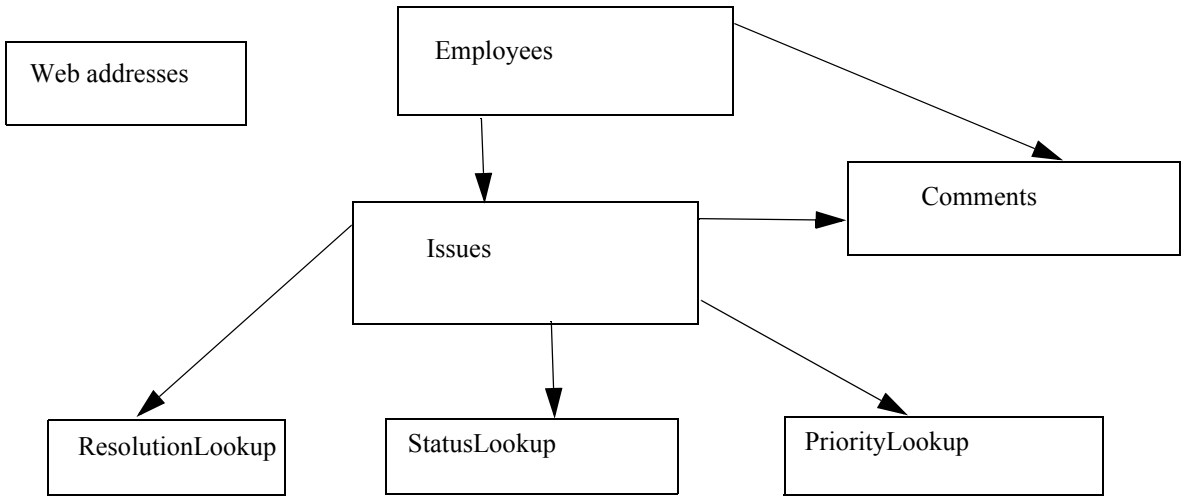
```
New EDB NewVar(issuesDB)
   ConnectString('Driver={Microsoft Access Driver (*.mdb);DBQ=C:\TestDb\issues.mdb');
Run issuesDB.Open;
. . .
```

A similar ConnectString will work for MS Excel spreadsheets. Other database types, though, require other options. You will have to check the Database system documentation for more information on those.

Demo Database

The Demo Microsoft Access database (Issues.mdb) is a simple issues tracking database. Employees enter problems or issues and other employees make comments on the issue. The Issues table contains information about each problem identified. The employee table contains a list of employees. A comment table contains the employee comments about each issue. There are also smaller lookup tables, such as Resolution, Status and Priority.

The following diagram illustrates the relationship between the various tables.



Demo database scripts

Now that the demo database has been registered to ODBC, you are ready to run some scripts that access that database.

The first script to run on a database is usually the general utiltity script called DBListFields.fsl. This script puts up a dialog box, which asks you to supply a Data source name and optionally a userid and password. When you click the OK button, the script builds a Frame document containing the names of all the tables for the data source and a list of fields for each table. It tells you what type of data the field contains, and, if you selected the data sample checkbox, it will include a line of data for each field (the first one in the table).

The following examples use the EDBUtils.fsl script that is found in the Lib directory. This is a set of database utilities.

DBTableAndFieldReport.fsl

To run this script on the demo database (or any database), do the following:

1. Start FrameMaker.
2. Run the script (ElmScript->Run...) in the SampleScripts directory called DBTableAndFieldReport.fsl
3. A dialog box should appear asking you to select a database. This is the standard Open Database dialog box (see below). Select the ISSUES data source name from the drop down box for the Demo Database (or select another data source if you wish). Leave the UserId and Password fields blank for the Demo database. You would need to use these if you choose another database which require a user and password.
4. Press the 'Connect to Database' button.

   A FrameMaker document should be created containing a list of tables and fields. This is a useful script that you can use to examine the structure of any ODBC compliant database.

DBIssuesReport.fsl

The DBIssuesReport.fsl script is a simple report script. It illustrates how to open a database and read data from doing multiple searches. It generates a FrameMaker document which contains a line for each issue recorded in the database and for each issue it displays a list of comments for that issue.

To run this demo, do the following:

1. Start FrameMaker.
2. Run the script (ElmScript->Run...) in the database demo directory called DBIssuesReport.fsl.

A FrameMaker document should appear containing the report described above.

DBIssues.fsl

The next demo script is much more complicated. It is an interactive script that fills in a table in the demo issues.fm document. The demo document (Issues.fm) is a form type document. It is designed to display the issues brought up by selected employee. You select an employee from a list of employees and the ElmScript script fills in the information in the table. This script also formats some of the data from the look up tables.

To run this demo, do the following:

1. Start FrameMaker.
2. **`Install`** the script (ElmScript->Install...) called Issues.fsl. When the dialog box appears select the issues.fsl script (using the browse button) and type in DBISSUES for the script name. Make sure that you install this script. Do not run it. It is an event script that stays around and responds to events. Also make sure that the script name is DBISSUES. The test document uses this name to communicate with the script via HyperText links.
3. Open the demo document located in the DatabaseDemo sub-directory called Issues.fm.
4. Click on the Select Employee button. A list of employees should appear. Select one and the screen will be updated with the new employee's information.
5. If you have Microsoft Access (97 or greater), you can open this database and make changes. Use the reload button to reload employee information.

DBTestUpdate.fsl

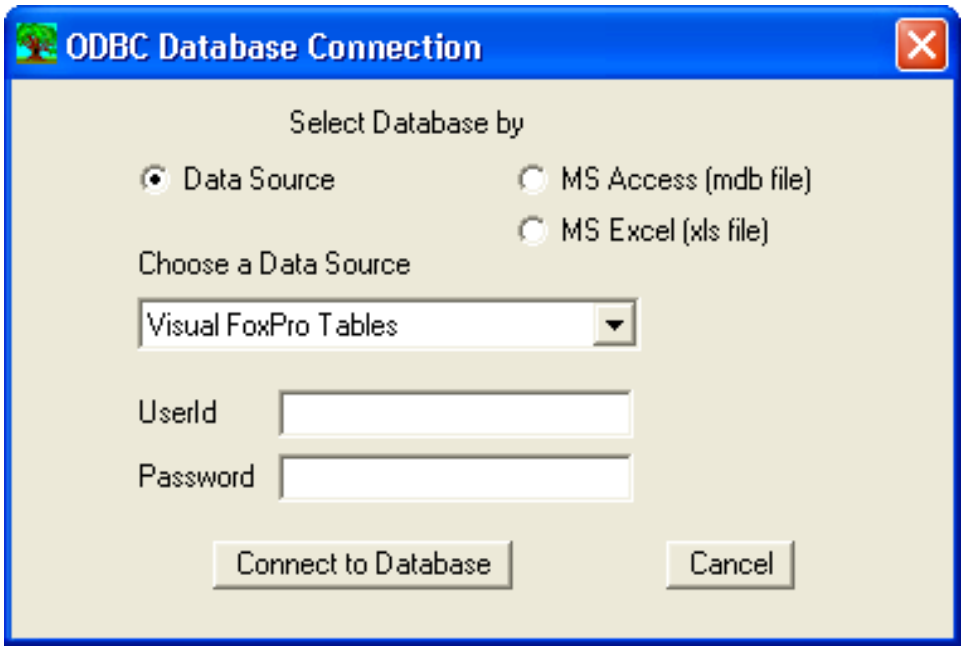
This script provides an example of updating a database. It will create a new table in the selected database, insert several records, update a few records, then delete one. To run this script on the demo database (or any database), do the following:

1. Start FrameMaker.
2. Run the script (ElmScript->Run...) in the SampleScripts directory called DBTestUpdate.fsl
3. A dialog box should appear asking you to select a database. This is the standard Open Database dialog box (see below). Select the ISSUES data source name from the drop down box for the Demo Database (or select another data source if you wish). Leave the UserId and Password fields blank for the Demo database. You would need to use these if you choose another database which require a user and password.
4. Press the 'Connect to Database' button.

A FrameMaker document should be created showing each step of the update process. It will first list all of the records in the newly created table. It will show this same table after the update is run, then after the delete is performed.

EDBConnectDlg.fsl

This is a utility script, located in the Lib directory that you can use to have a standard way of prompting the user for a data source/database to open. Examples of its use can be found in several of the above demo scripts. Of course, you can always write your own database open script, if this does not suit your needs. See below for a screen shot.


ODBC Database Connection
Select Database by
Data Source
MS Access (mdb file)
MS Excel (xls file)
Choose a Data Source
Visual FoxPro Tables
UserId
Password
Connect to Database
Cancel

Chapter 9

Database Object Reference

eODBCInfo Object

The eODBCInfo object represents the ODBC (Open Database Connectivity) system as a whole. It provides several useful ODBC properties that are not associated with a specific database. This is a permanent object It is always available. You do not need to create or to delete it.

eODBCInfo Properties

Table 119: eODBCInfo Properties

Property Name	Data Type	Property Description
DsnList ***(Read-Only)***	**StringList**	A list of the names of the currently registered Data Sources.
DsnDescList ***(Read-Only)***	**StringList**	A list of the descriptions of the currently registered Data Sources. There is one entry for each data corresponding source.
DriverList ***(Read-Only)***	**StringList**	A list of the available ODBC database drivers.
DriverAttrList ***(Read-Only)***	**StringList**	A list of attributes for the corresponding ODBC database driver. These are in the form of keyword and attribute pairs separated by a vertical bar(|).
ODBCVersion ***(Read-Only)***	**Integer**	The version number of the current ODBC installation.

Example:
The following script displays a list of Data Sources in the standard ScrollBox dialog and allows the user to select one.

```
. . .
DialogBox Type(ScrollBox) NewVar(vSelDsn) Button(vButton)
   List(eODBCInfo.DsnList) Title('Select a Data Source');
If vButton = OKBUTTON
...
EndIf
. . .
```

EDB Object

The EDB object represents a database. You must create an EDB object using the NEW EDB command before attempting to open or otherwise use the database. Some of the following properties should be set before a database is opened, such as the CanUpdate property. Others are only available after a database has been opened, such as `ODBCConnectString`.

Creating the object

You create an instance of this object using the NEW command.

Format:

```
New EDB DataSource(datasrcname) [User(username)] [PassWord(password)]
   NewVar(dbvar);
```

Table 120: New EDB Options

Option Name	Option Description
<<Any Property>>	You can supply any of the EDB updatable property names and values.
NewVar	The name of the variable to hold the newly created object.

Example:
The following script creates an EDB object referring to the datasource named 'Customers'. This database is unsecured with no user or password required.

```
. . .
New EDB DataSource('Customers') NewVar(custdb);
. . .
```

Example:
The following script creates an EDB object refering to the datasource named '`Northwind`'. The User name and password are supplied.

```
. . .
New EDB DataSource('Northwind') NewVar(northdb) User('MyUsername') Password('MyPass');
. . .
```

Example:
The following script creates an EDB object using a connect string instead of a pre-configured data source. The User name and password are also supplied.

```
. . .
New EDB NewVar(northdb)
   ConnectString('Driver={Microsoft Access Driver (*.mdb);DBQ=C:\TestDb\Test.mdb')
   User('MyUsername') Password('MyPass');
. . .
```

Example:
The following script creates an EDB object refering to the datasource named 'Customers'. This database is unsecured with no user or password required. Update access is requested.

```
. . .
New EDB DataSource('Customers') NewVar(custdb) CanUpdate(True);
. . .
```

EDB Properties

Table 121: EDB Properties

Property Name	Data Type	Property Description
IsOpen ***(Read-Only)***	**Integer**	True if a database is currently open; False otherwise.
DataSource	**String**	The name of the datasource specified on the open method. If used, it needs to be set before the database is opened. DSN is a synonym.
User	**String**	The user name specified on the open method. If used, it needs to be set before the database is opened.
Password	**String**	The password specified on the open method. If used, it needs to be set before the database is opened.
ConnectString	**String**	A formatted string giving the parameters to connect to a database. This is used in place of or in conjunction with the DataSource, User and Password options. See the discussion on database connections. If used, it needs to be set before the database is opened.
ODBCConnectString ***(Read-Only)***	**String**	A string containing a list of attributes (keywords and values) for the database connection returned by the ODBC driver. This is only available after a database is opened.
ErrorCode	**Integer**	The database error code for the last method.
ErrorMsg ***(Read-Only)***	**String**	The Text of the last error.
ODBCError ***(Read-Only)***	**Integer**	The ODBC error code for the last ODBC call.
DBMSName ***(Read-Only)***	**String**	The name of the database management system for the open database.
QuoteChar ***(Read-Only)***	**String**	The character used for quotes in identifiers for this database.
Translate	**Integer**	Set to True (default) to translate the characters from the platform character set to the Frame character set. Set to False to skip this translation.
CanUpdate	**Integer**	Set to True to allow database updates. Set to False(default) for Read-only access. If you wish to allow updates, you must set this property to True before opening the database.

EDB Methods

Table 122: EDB Methods

Method Name	Method Description
Open	Opens the database to be represented by the object.
Close	Closes the database to be represented by the object.
SqlCommand	This method allows you to send a SQL command directly to the Database connection. This is used for Adding, deleting, or updating tables. It can also be used to issue any other valid SQL command to the database. NOTE: Not all database systems support all SQL commands.

EDB Method Descriptions

Open

Format:

```
Run edbvar.Open
```

The Open method opens the database specified in the EDB object. You must open a database before you can read information from it.

Close

Format:

```
Run edbvar.Close
```

The Close method closes the database specified in the EDB object. You should close a database when you are finished using it.

SqlCommand

Format:

```
Run edbvar.SqlCommand Command(cmdString)
```

The SqlCommand method sends a SQL command to the corresponding database.

Table 123: SqlCommand Options

Option Name	Option Description
`Command`	A string variable or constant containing the text of the SQL command.

Example:
The following script creates opens a database identified by the Customers data source, then uses the SqlCommand method to create a new table, before adding a record to that new table.

```
. . .
New EDB DataSource('Customers') NewVar(custdb) User('MyUsername') Password('MyPass');
Run custdb.Open CanUpdate(True);
Local lvSqlString;
Set lvSqlString = 'CREATE TABLE MyTestTable (';
Set lvSqlString = lvSqlString + 'Field1 CHAR(30), ';
Set lvSqlString = lvSqlString + 'Field2 CHAR(30) ) ;';
Set custdb.errorcode = 0;
Run custdb.SqlCommand Command(lvSqlString);
If custdb.ErrorCode not = 0
   MsgBox 'Error on the Create SqlCommand-'+custdb.ErrorMsg+
   '  ODBC State'+custdb.ODBCERROR);
EndIf
Set lvSqlString = 'INSERT INTO MyTestTable ';
Set lvSqlString = lvSqlString + '(Field1, Field2) ';
Set lvSqlString = lvSqlString + 'VALUES (';
Set lvSqlString = lvSqlString + QUOTE + 'Value1' + QUOTE+ ', ';
Set lvSqlString = lvSqlString + QUOTE + 'Value1' + QUOTE+ ') ; ';

Run custdb.SqlCommand Command(lvSqlString);
If custdb.ErrorCode not = 0
   MsgBox 'Error on the Create SqlCommand-'+custdb.ErrorMsg+
   '  ODBC State-'+custdb.ODBCERROR);
EndIf

. . .
```

EQuery Object

The EQuery object represents a search and search result. You must create an EQuery object using the NEW EQuery command before attempting to run a search on the associated database.

Creating the object

You create an instance of this object using the NEW command.

Format:

```
New EQuery EDB(databaseobject) [Select(Selectstr)] [TableTypes(tableStr)]
   NewVar(dbvar);
```

The New method creates a new EQuery object. You must create an EQuery object before attempting to perform a search or try to access a result set. The parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 124: New EQuery Options

Option Name	Option Description
EDB	Identifies the EDB object associated with this query.
Select	The SQL select command for the search.
TableTypes	The list of table types for a SqlTables type query.
NewVar	The name of the variable to hold the newly created object.

Example:

The following script creates an `EQuery` object after creating a EDB object for the database.

```
. . .
New EDB DataSource('Customers') NewVar(custdb) User('MyUsername') Password('MyPass');
New EQuery EDB(custdb) Select('Select * From CustTable') NewVar(custquery));
. . .
```

EQuery Properties

Table 125: EQuery Properties

Property Name	Data Typ;e	Property Description
Fields	**EslObject**	An EslObject representing the fields in the current search set. You can get the information for a specific field in several ways. The first way is to specify the field name as a property of the Fields Object, as follows: `queryVar.Fields.FieldName` where `FieldName` is the name of a column in the database. You can also use the `Fields` object as an array and access the columns by number, as follows: `queryVar.Fields[1]` This will return the value of the first column in the results set. You can use an integer index from 1 up to the number of columns in the results set (`NumCols`). There is also a third way to access the column values. You can use the `Fields` object as an array indexed by a text string identifying the name of the column. This can be used when (and if) the database system allows spaces or special characters in the column names, as follows: `queryVar.Fields['Odd@@Column Name']` This is not a very common situation. You can access information about fields by getting the properties of the field in a similar fashion as above. `queryVar.Fields[1].ColName` returns the name of the column for the first field in the results set. See “Field Properties” on page 113 for a complete list of properties of each field.
CurrField	**Dependent**	This is an alternate way to get field information. Set the CurrFieldName property to the name of the field you wish, then the syntax `queryvar.CurrField` gives you the value of that field.
CurrFieldName	**String**	The name current field. This is used with `CurrField` above.
CurrFieldNum	**Integer**	The number of the current field. This is used with `CurrField` above. Set the CurrFieldNum to the number of the field that you wish and queryvar.CurrField gives the value of that field.

Table 125: EQuery Properties

Property Name	Data Typ;e	Property Description
DefStringDataType	**String**	The Default data type of character strings in the results set. This property should be set before the query is performed (e.g. RunQuery). The normal value is 'Ansi' for a standard Ansi string. Other values are 'Utf8' (for data in UTF-8 format) or 'UIntList' (where data is retrieved as Unicode (UTF-16) and converted to a list of unsigned integers. This property is primarily used when the data is stored in Unicode on the data source in order to convert it to a usable form, since FrameMaker does not support unicode in the UTF-16 format.
ErrorCode	**Integer**	The ElmScript error code for the last method.
ErrorMsg	**String**	The Text of the last error.
ODBCError	**Integer**	The ODBC error code for the last ODBC call.
QueryRun	**Integer**	True if a query has been run; False otherwise.
AtEOF	**Integer**	True if the current record pointer is at the end of the list of results; False otherwise.
Select	**String**	The SQL select string used to run the query.
EDB	**EDB**	The EDB database object associated with this query.
NumCols	**Integer**	The number of fields (or columns) in this result set.
TableTypes	**String**	The type of tables for the SQLTables query.

Table 126: Field Properties

Property Name	Data Typ;e	Property Description
ColName	**String**	The name of the field. (e.g. Query`.Fields.Name.ColName`).
ColNum	**Integer**	The number of the column.
SqlType	**Integer**	The integer SQL type of the field.
SqlTypeStr	**String**	The String SQL type of the field.
ColSize	**String**	The maximum size of the column.
Decimal	**Integer**	The number of decimal places (if applicable).
Nullable	**Integer**	True if the column can have a null value.
ConvertTo	**String**	If this field has character data, this property allows you to convert it to a different type of string. The default value is the value of the `DefStringDataType` property of the associated Query object. This is normally 'Ansi'. Other values are 'Utf8' (for data in UTF-8 format) or 'UIntList' (where data is retrieved as Unicode (UTF-16) and converted to a list of unsigned integers. If you wish to set this property, you must do it before data is retrieved. See the GetFirst option on the RunQuery command for more information. Use this property to convert individual fields. Use the `DefStringDataType` property (see above) to convert all the character string fields of the results set. This property is primarily used when the data is stored in Unicode on the data source in order to convert it to a usable form, since FrameMaker does not support unicode in the UTF-16 format.

EQuery Methods

Table 127: EQuery Methods

Method Name	Method Description
RunQuery	Runs a query.
RunSQLTables	Runs a query that produces a results set, one record for each table in the database.
RunSQLColumns	Runs a query that produces a results set, one record for each column in the selected Table.
GetNext	Makes the next record in the result available.
Close	Closes the query.

EQuery Method Descriptions

RunQuery

Format:

```
Run queryVar.RunQuery [GetFirst(getOption)];
```

The RunQuery method executes a query on the specified database and creates a result set. You must have previously specified a Select statement.

Table 128: RunQuery Options

Option Name	Option Description
GetFirst	When a query is performed, the first row is usually automatically fetched from the results set. Set this option to False to stop this. The default value is True. If you set this option to False, then you must run the GetNext method before attempting to access any field data. This option is set to False in order to set the ConvertTo property for a field.

Example 1:

The following script creates an `EQuery` object, runs a query and writes the value of each column in the first row.

```
. . .
New EDB DataSource('Customers') NewVar(custdb) User('MyUsername') Password('MyPass');
New EQuery EDB(custdb) NewVar(vQuery);
vQuery.Select = 'Select * From InfoTable';
Run vQuery.RunQuery;
Set vNumCols = vQuery.NumCols;
Loop LoopVar(vFieldNumber) Incr(1) InitVal(1) While(vFieldNumber<=vNumCols)
    Write Console 'Field '+vFieldNumber+
        ' Data=' +vQuery.Fields[vFieldNumber];
EndLoop

. . .
```

Example 2:
The following script creates an `EQuery` object, runs a query and writes the value of each column in the first row with all the character string fields converted to UTF-8.

```
. . .
New EDB DataSource('Customers') NewVar(custdb) User('MyUsername') Password('MyPass');
New EQuery EDB(custdb) NewVar(vQuery);
vQuery.Select = 'Select * From InfoTable';
Set vQuery.DefStringDataType = 'Utf8';
Run vQuery.RunQuery;
Set vNumCols = vQuery.NumCols;
Loop LoopVar(vFieldNumber) Incr(1) InitVal(1) While(vFieldNumber<=vNumCols)
   Write Console 'Field '+vFieldNumber+
       ' Data=' +vQuery.Fields[vFieldNumber];
EndLoop

. . .
```

Example 3:
The following script creates an `EQuery` object, runs a query and writes the value of each column in the first row with the first field converted to UTF-8.

```
. . .
New EDB DataSource('Customers') NewVar(custdb) User('MyUsername') Password('MyPass');
New EQuery EDB(custdb) NewVar(vQuery);
vQuery.Select = 'Select * From InfoTable';
Run vQuery.RunQuery GetFirst(False); // Do not read first row
Set vQuery.Fields[1].ConvertTo = 'Utf8';
Run vQuery.GetNext; // Get the first row in the results
Set vNumCols = vQuery.NumCols;
Loop LoopVar(vFieldNumber) Incr(1) InitVal(1) While(vFieldNumber<=vNumCols)
   Write Console 'Field '+vFieldNumber+
       ' Data=' +vQuery.Fields[vFieldNumber];
EndLoop

. . .
```

RunSqlTables

Format:

```
Run queryVar.RunSqlTables [TableName(nameOfATable]) [TableTypes('ListOfTablesTypes')]
       [GetFirst(getOption)];
```

The `RunSqlTables` method executes a query on the specified database and creates a result set that includes set of tables in the database. This query allows you to examine the tables and fields in a database. See “RunSqlTables Field Information” on page 116 for the fields returned in the results set for this query.

Table 129: RunSqlTables Options

Option Name	Option Description
TableName	A string with the name of the table. This is optional. If not specified the results set will contain all the tables in the database matching the types in the TableTypes option.

Table 129: RunSqlTables Options

Option Name	Option Description
TableTypes	A string with the list of table types to return. This is optional. If not specified the default value is 'TABLE,VIEW,SYSTEM TABLE'.
GetFirst	When a query is performed, the first row is usually automatically fetched from the results set. Set this option to False to stop this. The default value is True. If you set this option to False, then you must run the GetNext method before attempting to access any field data. This option is set to False in order to set the ConvertTo property for a field.

Example:
The following script creates an `EQuery` object and runs a RunSqlTables search.

```
. . .
New EDB DataSource('Customers') NewVar(custdb) User('MyUsername') Password('MyPass');
New EQuery EDB(custdb) NewVar(tableListQuery);
Run tableListQuery.RunSqlTables;
. . .
```

Table 130: RunSqlTables Field Information

Option Name	Option Description
TABLE_CAT	The catalog name of the Table, if applicable.
TABLE_SCHEM	The schema name of the Table, if applicable.
TABLE_NAME	The table name .
TABLE_TYPE	The Table Type ('TABLE', 'VIEW', 'SYSTEM TABLE', etc.)
REMARKS	User defined comments.

RunSqlColumns

Format:

```
Run queryVar.RunSqlColumns TableName(nameofTable) [ColumnName(colName)];
      [GetFirst(getOption)];
```

The `RunSqlColumns` method executes a query on the specified database and creates a result set that includes set of information about the columns for the specified table in the database. This query allows you to examine the column information for tables in a database. See "RunSqlColumns Field Information" on page 117 for the fields returned in the results set for this query.

Table 131: RunSqlColumns Options

Option Name	Option Description
TableName	A string with the name of the table.
ColumnName	A string with the name of the column. This is optional. If not specified the results set will contain all the columns in the table.
GetFirst	When a query is performed, the first row is usually automatically fetched from the results set. Set this option to False to stop this. The default value is True. If you set this option to False, then you must run the GetNext method before attempting to access any field data. This option is set to False in order to set the ConvertTo property for a field.

Example:
The following script creates an `EQuery` object and runs a RunSqlColumns search.

```
. . .
New EDB DataSource('Customers') NewVar(custdb) User('MyUsername') Password('MyPass');
Run custdb.Open;
New EQuery EDB(custdb) NewVar(columnListQuery);
Run columnListQuery.RunSqlColumns TableName('MyTable');
. . .
```

Table 132: RunSqlColumns Field Information

Option Name	Option Description
TABLE_CAT	The catalog name of the column, if applicable.
TABLE_SCHEM	The schema name of the column, if applicable.
TABLE_NAME	The table name of the column.
COLUMN_NAME	The Column name.
DATA_TYPE	The integer data type (ODBC defined).
TYPE_NAME	The string data type (ODBC defined, e.g. VARCHAR).
COLUMN_SIZE	The integer column size.
BUFFER_LENGTH	The integer buffer length (ODBC defined).
DECIMAL_DIGITS	The integer buffer length (ODBC defined).
NUM_PREC_RADIX	Significant digits to the right of the decimal point.
NULLABLE	If can contain nulls
REMARKS	User defined comments.
COLUMN_DEF	Default value of the column.
SQL_DATA_TYPE	The integer SQL data type(ODBC defined).
SQL_DATETIME_SUB	The sub type for Date-Time fields.
CHAR_OCTET_LENGTH	n/a
ORDINAL_POSITION	The position of the column in the table.
IS_NULLABLE	YES or NO
ORDINAL	n/a

GetNext

Format:

```
Run queryVar.GetNext;
```

The `GetNext` method moves the current pointer to the next record in the results set and loads the current fields into the .Fields property of the query object.

Chapter 10

EExpatXml Object

The EExpatXml object represents an instance of the ExPat parser. This object allows you to read an Xml file but not validate it.

ExPat is a stream oriented parser. You provide a set of events which the expat parser will run when it recognizes parts of the xml document. It will call these events when you feed the parser a complete or part of an Xml document.

Creating the object

You create an instance of this object using the `New` command.

Format:

```
New EExPatXml NewVar(expatVar) [PropertyName(PropertyValue)];
```

Table 133: New EExPatXml Options

Option Name	Option Description
`NewVar`	The name of the variable to hold the newly created EExpatXml object.
`PropertyName/PropertyValue`	A set of property names and values from the “EExpatXml Properties” on page 120.

Example:

The following script command creates an EExPatXml object.

```
. . .
New EExpatXml NewVar(expatVar);
. . .
```

Example:
The following script creates an EExpatXml object and provides two events OnElementStart and OnElementEnd. When.you run the ParseXmlFile method, ExPat will run the EltStart event whenever it encounters the start of an element and it will run the EndElt event whenever it encounters the end of an element.

```
. . .
New EExpatxml NewVar(expatVar) OnElementStart('EltStart') OnElementEnd('EltEnd');
...
Run expatVar.ParseXmlFile FileName('MyXmlFile.xml');
...
Event EltStart using pvEltName pvAttrList
...
EndEvent
Event EltEnd using pvEltName
...
EndEvent;
. . .
```

Deleting the object

To delete an instance of this object use the `Delete` Object command. Delete the object when you do not need it anymore.

Delete Object

The Delete method deletes the `EExPatXml` object.

Format:

```
Delete Object(expatVar);
```

EExpatXml Properties

Table 134: EExpatXml Properties

Property Name	Data Type	Property Description
ErrorCode	**Integer**	The ElmScript error code for the last method.
ErrorMsg	**String**	The Text of the last error.
OnElementStart	**String**	The name of the event to run when Expat encounters the start of a new element. This event will have two parameters, the name of the element (`pvEltName`) and a list of attribute/values pairs (`pvAttrList`). The odd numbered members are the attribute names and the even numbered members are the attribute values.
OnElementEnd	**String**	The name of the event to run when Expat encounters the end of an element. This event will have one parameter, the name of the element (`pvEltName`).
OnCharData	**String**	The name of the event to run when Expat encounters the character data. This event will have one parameter, a string containing the characters (pvCharData).

Table 134: EExpatXml Properties

Property Name	Data Type	Property Description
OnPI	**String**	The name of the event to run when Expat encounters a processing instruction. This event will have two parameters, a string containing the first word in the processing instruction (pvTarget) and a string containing the rest of the characters stripping the leading whitespace (pvData).
OnComment	**String**	The name of the event to run when Expat encounters a comment. This event will have one parameter, a string containing the text of the comment (pvData).
OnStartCDataSection	**String**	The name of the event to run when Expat encounters the start of a CData section. There are no parameters.
OnEndCDataSection	**String**	The name of the event to run when Expat encounters end of a CData section. There are no parameters
OnDefault	**String**	The name of the event to run when Expat encounters something that wouldn't otherwise be handled. This includes both data for which no other event can be set (like some kind of DTD declarations) and data which could be reported but which currently has no event declared. This event will have one parameter, a string containing the text of the event (pvData).
OnDefaultExpand	**String**	This is the same as OnDefault but it doesn't affect the expansion of internal references.
OnExternalEntityRef	**String**	The name of the event to run when Expat encounters an external DTD subset, if parameter entity parsing is in effect. This event will have four parameters, a string containing the context (pvContext), a string containing the base to use for relative system identfiers (pvBase), a string containing the system Id (pvSystemId) and a string containing the publicId (pvPublicId)

EExpatXml Methods

Table 135: EExpatXml Methods

Method Name	Method Description
New	Creates the EExpatXml object.
Delete	Deletes the EExpatXml object.
ParseXmlFile	Tells ExPath to process a file containing an Xml document.
ParseXmlBuffer	Tells ExPath to process a string containing a part or all of an Xml document.

EExpatXml Method Descriptions

ParseXmlFile

The ParseXmlFile method tells ExPat to process a file containing an Xml document. You should have previously set the properties identifying the events for which you want ExPat to run.

Format:

```
Run expatVar.ParseXmlFile FileName(fileName) NewVar(retValue);
```

Table 136: ParseXmlFile Options

Option Name	Option Description
FileName	A string value specifying the name of the file to process.
NewVar	'Success' or an error message.

Example:
The following script creates an `EExpatXml` object then tells it to parse the specified file.

```
. . .
New EExpatXml NewVar(expatVar) OnElementStart('EltStart') OnElementEnd('EltEnd');
Run expatVar.ParseXmlFile FileName('MyXmlFile.xml');
. . .
```

ParseXmlBuffer

The ParseXmlBuffer method tells ExPat to process a string containing all or part of an Xml document. You should have previously set the properties identifying the events for which you want ExPat to run. You can feed the ExPat parser part of the Xml document in succesive calls to this method. Set the Done option to True on the last string.

Format:

```
Run expatVar.ParseXmlFile Buffer(stringValue) [Done(True/False)] NewVar(retVal);
```

Table 137: ParseXmlBuffer Options

Option Name	Option Description
Buffer	A string value containing the Xml text to process.
Done	True if this is the last string to parse. False if there will be more strings for this Xml document. The default value is True.
NewVar	'Success' or an error message.

Example:
The following script creates an `EExpatXml` object then sends it an Xml document in three pieces.

```
. . .
New EExpatXml NewVar(expatVar) OnElementStart('EltStart') OnElementEnd('EltEnd');
Set gvText='<?Xml Version="1.0"> <MainElt><AnotherElt> Data </AnotherElt>';
Run expatVar.ParseXmlBuffer Buffer(gvText) Done(False);
...
Set gvText2='<AnotherElt> More Data </AnotherElt>';
Run expatVar.ParseXmlBuffer Buffer(gvText2) Done(False);
...
Set gvText3='<AnotherElt> Yet More Data </AnotherElt></MainElt>';
Run expatVar.ParseXmlBuffer Buffer(gvText2) Done(True);
Delete Object(expatVar);
. . .
Event EltStart using pvEltName pvAttrList
   Write Console 'Found Start of Element-'+pvEltName;
EndEvent
Event EltEnd using pvEltName
   Write Console 'Found End of Element-'+pvEltName;
EndEvent;
```

Chapter 11

ETextParser Object

The ETextParser object represents a text parser. This object allows you to break a text file or text string into tokens.

Creating the object

You create an instance of this object using the `New` command.

Format:

```
New ETextParser NewVar(gvParser) [PropertyName(PropertyValue)];
```

Table 138: New ETextParser Options

Option Name	Option Description
NewVar	The name of the variable to hold the newly created ETextParser object.
PropertyName/PropertyValue	A set of property names and values from the “ETextParser Properties” on page 126.

Example:
The following script command creates an ETextParser object.

```
. . .
New ETextParser NewVar(gvParser);
. . .
```

Example:
The following script creates an ETextParser object and provides an input string to parse and a delimiter of the dash character..

```
. . .
New ETextParser NewVar(gvParser) InputString('Text-To-Parse')
   DelimiterList('-');
...
Run gvParser.GetNextToken;
...
. . .
```

ETextParser Properties

Table 139: ETextParser Properties

Property Name	Data Type	Property Description
ErrorCode	**Integer**	The ElmScript error code for the last method.
ErrorMsg	**String**	The Text of the last error.
InputFile	**String**	The name of the input file to parse. You should provide this option or the next option (`InputString`) before running the GetNextToken method.
InputString	**String**	The input string to parse. You should provide this option or the previous option (`InputFile`) before running the `GetNextToken` method.
AtEos	**Integer**	True if the parser has reached the end of the input stream, False otherwise.
BeginBlockComment	**String**	The text of the string that is used to start a block comment in the input text. This is combined with the `EndBlockComment` option to provide the boundaries for a block comment. For example, in a ElmScript script the BeginBlockComment is `'/*'`
EndBlockComment	**String**	The text of the string that is used to end a block comment in the input text. This is combined with the `BeginBlockComment` option to provide the boundaries for a block comment. For example, in a ElmScript script the EndBlockComment is `'*/'`
LineComment	**String**	The text of the string that is used to start a line comment in the input text. The end of line will act to end the line comment. For example, in a ElmScript script the LineComment is `'//'`
QuoteCharList	**String**	The list of characters that will be treated as a quote. The default value specifies a single quote (') and a double quote (").
DelimiterList	**String**	The list of one character delimiters. The default value is `'<>,;#%&*()+=-|?/:!'`
MaxTokenSize	**Integer**	The maximum size of a token. The default value is 300 characters.
CurrToken	**String**	The last token parsed..
CurrTokenType	**String**	The type of the last token parsed. The possible values are as follows: 'CHR' - character string 'STR' - quoted string 'DLM' - delimiter 'EOL' - end of a line 'EOS' - end of the input stream.
CurrLine	**Integer**	The line number of the last token parsed..
CurrPos	**Integer**	The column number of the last token parsed.
LastUsedQuoteChar	**String**	If the last token type was a string, this indicates which quote character was used.
WhiteSpaceCount	**Integer**	The number of white space characters before the last token.

ETextParser Methods

Table 140: ETextParser Methods

Method Name	Method Description
GetNextToken	Parses the next token.
GetNextTokenSkip	Parses the next token and optionally skips the specified token types.
PutBackToken	Puts back a token to the parsing stream. This will be the next token retrieved.
Rewind	Rewinds the input file or string.

ETextParser Method Descriptions

GetNextToken

The GetNextToken method parses the next token. The results of the method will be in the CurrToken and CurrTokenType properties.

Format:

```
Set gvRet=gvParser.GetNextToken{};
```

Table 141: GetNextToken Options

Option Name	Option Description
gvRet	This returns a 0 if successful or a -1 for an error..

Example:
The following script creates an `ETextParser` object then tells it to parse the specified string, writing the information to the FrameMaker console.

```
. . .
New ETextParser NewVar(gvParser);
Set gvInputString = 'XXX    ,YYY  , ZZZ : PPP(Q,W);';

Set gvParser.InputString=gvInputString;
Set gvDone=False;
Loop While(gvDone=False)
    Set gvRet=gvParser.GetNextToken{};
    If gvRet>=0
        Set gvToken=gvParser.CurrToken;
        Set gvTokenType=gvParser.CurrTokenType;
        Write Console 'Type='+gvTokenType+' Token-'+gvToken;
        If gvTokenType='EOS'
            Set gvDone=True;
        EndIf
    Else
        Write Console 'Return Error-'+gvRet;
        Set gvDone=True;
    EndIf
EndLoop

. . .
```

GetNextTokenSkip

The GetNextToken method parses the next token, allowing you to skip some token types. This is convenient if, for example, you do not care about the end of line tokens. The results of the method will be in the CurrToken and CurrTokenType properties.

Format:

```
Set gvRet=gvParser.GetNextTokenSkip{tokenType[,tokenType2]...};
```

Table 142: GetNextTokenSkip Options

Option Name	Option Description
tokenType	A string value containing a token type to skip. See the `CurrTokenType` property for the possible values.
gvRet	This returns a 0 if successful or a -1 for an error.

Example:
The following script creates an `ETextParser` object then provides a file name and comment type indicators. It then parses the file writing out all the tokens to the FrameMaker console, while skipping any end of line tokens.

```
. . .
New ETextParser NewVar(gvParser);
Set gvInputFile = 'MyScript.fsl';
Set gvParser.LineComment='//';
Set gvParser.BeginBlockComment='/*';
Set gvParser.EndBlockComment='*/';

Set gvParser.InputFile=gvInputFile;
Set gvDone=False;
Loop While(gvDone=False)
    Set gvRet=gvParser.GetNextTokenSkip{'EOL'};
    If gvRet>=0
        Set gvToken=gvParser.CurrToken;
        Set gvTokenType=gvParser.CurrTokenType;
        Write Console 'Type='+gvTokenType+' Token-'+gvToken;
        If gvTokenType='EOS'
            Set gvDone=True;
        EndIf
    Else
        Write Console 'Return Error-'+gvRet;
        Set gvDone=True;
    EndIf
EndLoop
```

PutBackToken

The PutBackToken method sends back a token, so that it may be retrieved later. This is useful if you are parsing some text until a certain token appears. When it appears you can put it back for another part of the script to process.

Format:

```
Set gvRet=gvParser.PutBackToken{token, tokenType};
```

Table 143: PutBackToken Options

Option Name	Option Description
token	A string value containing the token to put back.
tokenType	A string value containing a token type toput back. See the `CurrTokenType` property for the possible values.
gvRet	This returns a 0 if successful or a -1 for an error..

Example:
The following script creates an `ETextParser` object then tells it to parse the specified string, writing the information to the FrameMaker console.

```
. . .
New ETextParser NewVar(gvParser);
Set gvInputString = 'XXX    ,YYY  , ZZZ : PPP(Q,W);';
Set gvParser.InputString=gvInputString;
Run SkipUntilParen;
Set gvDone=False;
Loop While(gvDone=False)
    Set gvRet=gvParser.GetNextToken{};
    If gvRet>=0
        Set gvToken=gvParser.CurrToken;
        Set gvTokenType=gvParser.CurrTokenType;
        Write Console 'Type='+gvTokenType+' Token-'+gvToken;
        If gvTokenType='EOS'
            Set gvDone=True;
        EndIf
    Else
        Write Console 'Return Error-'+gvRet;
        Set gvDone=True;
    EndIf
EndLoop

. . .
Sub SkipUntilParen
    Local lvDone(False);
    Loop While(lvDone=False)
        Set gvRet=gvParser.GetNextToken{};
        If gvRet>=0
            If gvParser.CurrTokenType='DLM' and gvParser.CurrToken='('
                Set lvDone=True;
                    // This token will be processed back in the
                    // calling routine.
                gvParser.PutBackToken{gvParser.CurrToken,'DLM'};
            EndIf
        Else
            Set lvDone=True;
        EndIf
    EndLoop
EndSub
```

Rewind

The Rewind method rewinds the input stream to the beginning.

Format:

```
Set gvRet=gvParser.Rewind{};
```

Table 144: Rewind Options

Option Name	Option Description
gvRet	This returns a 0 if successful or a -1 for an error..

Chapter 12

Forms Overview

Introduction

The ElmScript Forms objects provide the ability to create and to present custom Forms (Windows and Dialog boxes) to the users. In other words, you can develop a GUI (graphical user interface) for your scripts, instead of using the relatively limited user interaction capability of standard ElmScript. There are three basic types of 'Forms Objects', the Form object itself (EForm), the Menu objects and the Control objects. These correspond to the GUI constructs with which every computer user is familiar regardless of platform.

EForm Object

The EForm object represents an entire form (window or dialog box). A form acts as a container for other objects, such as controls and menubars. It has a caption and is optionally resizeable. Forms can vary in size (width and height) and placement (position on the screen). You can set properties to specify these values.

Control Objects

Control objects are the familiar components of a GUI such as Buttons, Checkboxes, Radio buttons, Labels, Edit boxes, Group boxes, List boxes, Drop down boxes and Scrollbars. These control objects are placed on forms via panels (see below). Controls cannot exist outside of a form. Users can interact with controls to enter data, view data and cause events to be triggered. Controls can be of varying size and can be placed anywhere on the form.

> **IMPORTANT:** Due to technical difficulties in displaying dialog boxes from within FrameMaker, some keyboard operations do not always work as is customary. The Tab key in particular will not always work as it usually does inside dialog boxes. Using the mouse is recommended for accessing form controls.

Panel Objects

As mentioned above, controls are not placed directly on forms, but instead are placed on panels. A panel is container for controls. A panel is itself a control, therefore a panel can contain other panels as well as the standard control objects.

Every form contains at least one panel. This is called the main panel and its dimensions completely cover the client area of the form. This panel is always present and it changes sizes only when the form itself is resized. When you place a control on a form using the Form option, as follows:

```
New EForm NewVar(gvForm) Caption('My Test Form');
New EButton Form(gvForm) NewVar(gvBtn) Caption('My Button');
```

The button actually goes on the main panel.

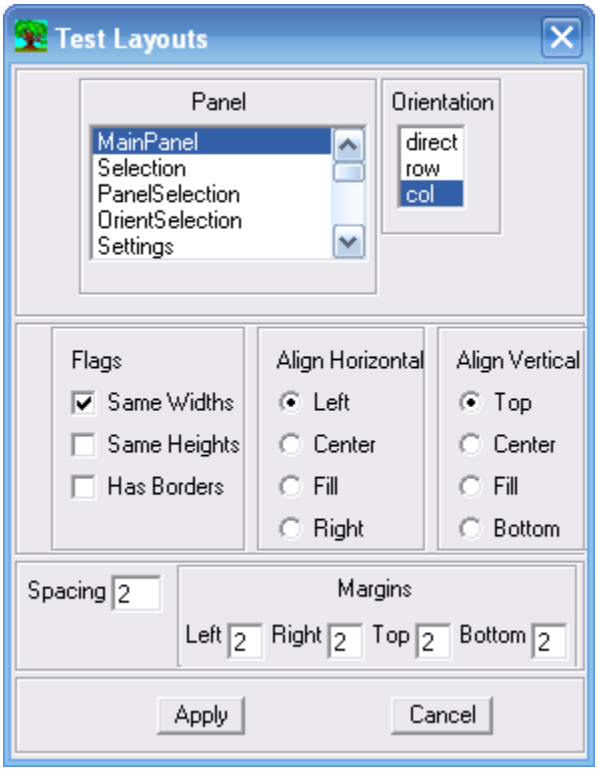
Before ElmScript version 6, there was only one panel, the main panel. Controls were placed on this panel by providing coordinates (left, top, width, height). Starting with ElmScript version 6, panels can be created and placed on the main panel (or contained within other panels). When you specify coordinates for a control, they are relative to the parent panel's upper left corner, not the form itself.

Panel Layout

In addition to being able to use panels to organize controls, panels can be used to place and size the controls on the panel. This is done by specifying the panel's orientation. The default way controls are placed on a panel is to specify the coordinates of each control. This is the 'direct' orientation. You can also use the 'row' or 'col' orientation. In the 'row' orientation, the layout manager places the controls in a horizontal manner in the panel. In the 'col' orientation, controls are placed in a vertical manner. In either case, the left and top coordinates (if specified) are ignored. The panel layout managers also use other properties to determine how the controls are placed and sized. You can specify the margins for each part of the panel (left, top, bottom, right). You can also specify the distance between controls as well as the alignment of controls within the panel.

PanelLayoutDemo.fsl

There is a sample script, PanelLayoutDemo.fsl, located in the Demos\MiscDemos folder under the ElmScript folder. This script illustrates using the layout managers (orientation 'direct' 'row' and 'col'), see below.



At the top is a list box containing the names of all the panels on the form. The main panel contains four other panels (Selection,Settings,MargSpace and ButtonPanel). These panels also have panels within them. When you select an item from the Panel Listbox, the panel properties of that panel will be presented in the other controls. If you change one or

more of the values and press the Apply button (at the bottom), the properties of the selected panel will be modifed and you can see the result of the changes immediately on the form. For example, select the Selection panel in the list box, then use the buttons under the Align Horizontal panel (Left,Center,Fill and Right) to see how it looks with a different horizontal alignment. Don't forget to press the Apply button after each change. Also, just for fun, change the orientation (in the orientation list box) from 'row' to 'col' and press Apply to see how the panel looks with a different layout. Feel free to change anything on the form, then appy the changes to see what happens.

Menu Objects

Menu objects are similar in spirit to controls, in that they can cause events to be triggered. They are not placed or sized, but exist as a tree structure with the Menubar as the ultimate parent object. A menu can also be a popup menu, which can exist outside of a menubar. A form can have one menubar at a time. An EMenubar object represents a menu bar on a form. and consists of one or more EMenu objects, which represent the drop down (or popup) menus. An EMenu object can consist of other menus, menuitems (EMenuItem) and menu separators (EMenuSeparator). Unlike controls, Menus can exist outside of a form and they can be displayed (popped up) independently.

Events

When a control event is called (such as OnRightClick) two parameters are passed to the event. The first is the form object variable (called EFormVar) and the second is the Control Object variable (EControlVar) of the control that caused the event. This allows you to use the same event subroutine to handle events in different forms and controls.

Panels, Placement and Sizing

Controls are usually placed on a form using absolute coordinates, top, left, width and height. The Top and Left coordinates are relative to upper left hand corner of the form. The form itself has the same properties. The Top and Left coordinates are relative to the monitor desktop. Imagine the desktop as a grid with the (0,0) coordinate at the top left of the screen and imagine the form as a grid with the upper left corner at position (0,0).

> **Note:** The form itself has a caption area that is included in the height of the form but is not included in the client area (where the controls are placed). The upper left corner is of the client area, not the actual top-left of the form on the screen. So always make the form a little higher than the lowest control position.

There are alternate ways to specify the placement and size of a control or form. For example, if you want a control to be near the bottom of a form, you can specify that the top coordinate value is relative to the bottom of the form instead of the top. You can also specify that the top coordinate is as a percentage of the height of the containing form. The other values (Left, Width and Height) have similar options.

For fixed size forms this is only useful at the design stage, but for resizeable forms these options are much more interesting. In FrameMaker, when you bring up many of the dialog boxes (the paragraph designer, for example), you can resize them. Move the mouse to the lower right corner and drag the dialog box to make it bigger. In FrameMaker, this have very little use. The dialog box gets bigger but the controls remain where they are. You just get a large grey empty space. With ElmScript you can make resizeable forms a valuable design criteria. By using relative instead of absolute coordinates, you can have the controls move and change sizes based on the size of the form.

For example, if you specify that a control's top coordinate is relative to the bottom of the form instead of the top, when the user resizes the form that control will stay in the same relative position. In other words, the control will move along with the form size.

The next chapter gives the placement and sizing options for controls and forms.

Chapter 13

Form and Control Objects

Common Form Object Properties

All Form objects (forms and controls) have the following properties. The container for a form is the entire display screen itself. The container for a control is the form on which it is placed.

Table 145: Common Form Object Properties

Property Name	Data Type	Property Description
	Location of the vertical position of the top-left corner of the control or form. Choose one of the following options.	
`Top`	`Integer`	Top position (in screen coordinates) from the container top.
`TopLocFromBottom`	`Integer`	Top position (in screen coordinates) from the container bottom.
`TopLocPercent`	`Real`	Top position as a percentage of the container height.
	Location of the horizontal position of the top-left corner of the control or form. Choose one of the following options.	
`Left`	`Integer`	Left position (in screen coordinates) from the container left.
`LeftLocFromRight`	`Integer`	Left position (in screen coordinates) from the container right.
`LeftLocPercent`	`Real`	Left position as a percentage of the container width.
	Width of the control or form. Choose one of the following options. If the width is not specified, then a default width (usually based on the contents of the control) will be used.	
`Width`	`Integer`	Width in screen coordinates.
`WidthMinus`	`Integer`	Width in screen coordinates minus the width of the container.
`WidthPercent`	`Real`	Width as a percentage of the container width.
	Height of the control or form. Choose one of the following options. If the height is not specified, then a default height (usually based on the contents of the control) will be used.	
`Height`	`Integer`	Height in screen coordinates.
`HeightMinus`	`Integer`	Height in screen coordinates minus the height of the container.
`HeightPercent`	`Real`	Height as a percentage of the container height.
	Limits on the dimensions. The following properties limit the boundaries of the control or form. After the height and width are calculated based on the above properties, these boundary properties are applied. A value of 0 (zero) means that there is no limit.	
`MinWidth`	`Integer`	Minimum width in screen coordinates. Default value is 0.
`MaxWidth`	`Integer`	Maximum width in screen coordinates. Default value is 0.
`MinHeight`	`Integer`	Minimum height in screen coordinates. Default value is 0.

Table 145: Common Form Object Properties

Property Name	Data Type	Property Description
MaxHeight	**Integer**	Maximum height in screen coordinates. Default value is 0.
The calculated dimensions of the control. These values are calculated from the above values. You can use these values but not modify them.		
ActualTop ***(Read-Only)***	**Integer**	Calculated top position (in screen coordinates) from the container top. *(Read-Only)*
ActualLeft ***(Read-Only)***	**Integer**	Calculated left position (in screen coordinates) from the container left. *(Read-Only)*
ActualWidth ***(Read-Only)***	**Integer**	Calculated width in screen coordinates. *(Read-Only)*
ActualHeight ***(Read-Only)***	**Integer**	Calculated height in screen coordinates. *(Read-Only)*
	The font information. For forms, this is the default font information for its controls. For controls, this will provide the font information for any text in the control. If not specified, the font information for the control is taken from the form font information.	
FontName	**String**	The name of the font. For controls, the default value is the form font name. For Forms the default value is ’**MS Sans Serif’**
FontCharset	**String**	The name of the Character set for the FontName. Most of the time this should be omitted, because most of the time, the default character set is the correct one. Some fonts, such as Arial, though, have several character sets available. The following values are possible for the FontCharset value. Whether any are valid, depends on the FontName chosen: ’DEFAULT’, ’SYMBOL’, ’BALTIC’, ’MAC’, ’RUSSIAN’, ’EASTEUROPE’, ’THAI’, ’VIETNAMESE’, ’TURKISH’, ’GREEK’, ’ARABIC’, ’HEBREW’, ’JOHAB’, ’OEM’, ’CHINESEBIG5’, ’GB2312’, ’HANGUL’, ’SHIFTJIS’, ’HANGEUL', 'ANSI' ’DEFAULT’ is the default value.
FontWeight	**String**	The font weight ('Normal', 'Bold'). For controls, the default value is the form font weight. For Forms the default value is **Normal**
FontAngle	**String**	The Font angle ('Normal', 'Italic'). For controls, the default value is the form font angle. For Forms the default value is **Normal**
FontSize	**Integer**	The size of the font. For controls, the default value is the form font size. For Forms the default value is **12**
	Scripter specific information. You may use these values to store your own information. The EForm system will store it but will not use it.	
UserString	**String**	An optional value into which the script writer may put a string value.
UserInteger	**Integer**	An optional value into which the script writer may put an integer value.

EForm Object

The EForm object represents a Dialog box form. You must create an EForm object using the NEW EForm command before can use the form. When you use the Delete method to delete a EForm object, *all of its controls* are also deleted at the same time, making them inaccessible.

Creating the object

You create an instance of this object using the NEW command.

Format:

```
New EForm NewVar(formVar) [PropertyName(PropertyValue)] ...;
```

The New method creates a new EForm object. You must create an EForm object before attempting to create controls for it. You may

Table 146: New EForm Options

Option Name	Option Description
`NewVar`	The name of the variable to hold the newly created object.
`PropertyName/PropertyValue`	Set of property names (from the list of properties on "Common Form Object Properties" on page 137 and "EForm Properties" on page 140) with their associated values.

Example:
The following script creates an EForm object of the specified size.

```
. . .
New EForm NewVar(efm) Top(100) Left(100) Width(500) Height(300);
. . .
```

Example:
The following script creates a user resizeable EForm object of the specified size.

```
. . .
New EForm NewVar(efm) Top(100) Left(100) Width(500) Height(300) Resizeable(True);
. . .
```

Index Access for EForms

The EForm object includes a pseudo-index. The bracket operators ([]) allows you to retrieve the control objects on the form by control name. You can then use the returned value to access the properties and methods of the controls without having to save each control object in a variable. For example,

Example:
The following script creates an EForm object, adds a label (`ELabel` object) then presents the form as a modal dialog box.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
. . .
New ECheckBox Form(efrm) Top(100) Left(50) Caption('OK') Name('MyCB1');
. . .
Set efrm['MyCB1'].IsChecked = True;
Run efrm.ShowModal NewVar(gvRetValue);
. . .
```

EForm Properties

Table 147: EForm Properties

Property Name	Data Type	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137, the following properties are available for EForm objects.	
CanResize	**Integer**	True if the form is user resizeable; False otherwise.
Caption	**String**	The text in the title bar of the form.
ClientHeight	**Integer**	The height of the client area of the form. The client area is the main area of the form excluding the borders, menus (if any) and title bar.
ClientWidth	**Integer**	The width of the client area of the form. This is the form width minus the border area.
CloseForm	**Integer**	True if the form is going to be closed; False otherwise. You can set this value to True inside a form or control event to cause the form to close after the event is processed. You can also set the value to False to cancel a form closing.
ControlCount	**Integer**	The number controls on the form. *(Read-Only)*
CurrentFocus	**String**	The name of the control which currently has focus.
EnableToolTips	**Integer**	True if the tool tips are enabled; False otherwise. Default is False.
ErrorCode	**Integer**	The ElmScript error code for the last method.
ErrorMsg	**String**	The Text of the last error.
FirstControlInForm	**EslObject**	The control object of the first control on the form. *(Read-Only)*
FirstFocus	**String**	The name of the control which has focus when the form is first displayed.
IsOnScreen	**Integer**	True if the form is displayed; False otherwise. *(Read-Only)*
LastControl	**String**	The name of the control which was last accessed. *(Read-Only)*
LastControlInForm	**EslObject**	The control object of the last control on the form. *(Read-Only)*
MainPanel	**EslObject**	The Main Panel object for this form. You can use this to set properties of the main panel.
MaximizeBox	**Integer**	True if the form has a maximize box; False otherwise.
Menubar	**EslObject**	The menu object to use for this form.
MinimizeBox	**Integer**	True if the form has a minimize box; False otherwise.
NcHeight	**Integer**	The height of the non-client area of the form. The non-client area includes the borders, menus (if any) and title bar.
NcWidth	**Integer**	The width of the non-client area of the form, which is the border.
OnClose	**String**	The name of the event to run before the form is dismissed. You can cancel the form closing by setting the CloseForm property to False
OnInitForm	**String**	The name of the event to run just before the screen is displayed (following a ShowModal or ShowModeless method call. Default: Nothing happens when just before the form appears.
OnMove	**String**	The name of the event to run after the form has been moved by the user.

Table 147: EForm Properties

Property Name	Data Type	Property Description
OnRightClick	**String**	The name of the event to run when the user right clicks the mouse button. Default: Nothing happens when the user right clicks the mouse button.
OnSize	**String**	The name of the event to run after the form has been resized by the user.
ReturnValue	**Integer**	The value to return when the form is dismissed. Used for Modal forms.
WindowState	**String**	The current state of the window. 'Normal', 'Minimized' or 'Maximized'.

Form Events

When a form event is called (such as OnInitForm, OnRightClick or OnClose) one parameter is passed to the event (called EFormVar). It is the form object variable. This allows you to use the same event subroutine to handle events in different forms.

EForm Methods

Table 148: EForm Methods

Method Name	Method Description
ShowModal	Creates the form and all its controls, then displays the form modally.
ShowModeless	Creates the form and all its controls, then displays the form modelessly.
FindControl	Finds a control by name within the form.
RecalcLayout	Recalculates the layout of the controls on the screen. This should be run if you change the layout information for the controls or the form.
BringToTop	Brings the form to the top of the stack of overlapping windows.

EForm Method Descriptions

ShowModal

Format:

```
Run efrm.ShowModal NewVar(returnValue)
or
Set returnvalue = efrm.ShowModal{};
```

The ShowModal method creates the form and all of its controls, before displaying the form on the screen modally. The form is displayed on the screen and the user cannot proceed with any other FrameMaker functions until the form is dismissed by user action.

Table 149: ShowModal Options

Option Name	Option Description
NewVar	The return value as defined by the control which caused the Modal form to go away or as set in the ReturnValue form property.

Example:
The following script creates an EForm object, adds a label (`ELabel` object) then presents the form as a modal dialog box.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New ELabel Form(efrm) Top(50) Left(50) Caption('My Label');
New EButton Form(efrm) Top(100) Left(50) Caption('OK') CloseFormWithValue(100);
New EButton Form(efrm) Top(100) Left(100) Caption('Cancel') CloseFormWithValue(999);
Run efrm.ShowModal NewVar(gvRetValue);
If gvRetValue = 100
   Display 'The user pressed the OK button';
Else
   Display 'The user pressed Cancel';
EndIf
. . .
```

ShowModeless

Format:

```
Run efrm.ShowModeless NewVar(returnValue)
or
Set returnValue = efrm.ShowModeless{};
```

The ShowModeless method creates the form and all of its controls, before displaying the form on the screen modelessly. The form stays on the screen until the user specifically dismisses it or the script terminates. The user may perform other FrameMaker functions while the form is on the screen.

Table 150: ShowModeless Options

Option Name	Option Description
NewVar	The return value as defined by the control which caused the Modal form to go away or as set in the ReturnValue form property..

Example:
The following script creates an EForm object, adds a label (`ELabel` object) then presents the form as a modeless dialog box.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New ELabel Form(efrm) Top(50) Left(50) Caption('My Label');
Run efrm.ShowModeless;
. . .
```

FindControl

Format:

```
Run efrm.FindControl Name(controlName) NewVar(eCtlVar);
or
Set eCtlVar = efrm.FindControl{controlName};
```

The FindControl method returns the control object for the control with the specified name.

Table 151: FindControl Options

Option Name	Option Description
`eCtlVar`	The name of the variable to hold the control object.

Example:
The following script creates an EForm object, adds a label (`ELabel` object), then presents the form as a modal dialog box. Later it finds the label control using its name.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New ELabel Form(efrm) Top(250) Left(50) Caption('My Label')
   Name('MyLabel');
Run efrm.ShowModal NewVar(gvRetValue);
. . .
Set gvCtl = efrm.FindControl{'MyLabel'};
. . .
```

RecalcLayout

Format:

```
Run efrm.RecalcLayout NewVar(returnValue)
or
Set returnValue = efrm.RecalcLayout{};
```

The RecalcLayout method recalculates the layout of the controls on the form. This should be run if you change layout information in any of the controls or of the form.

Table 152: RecalcLayout Options

Option Name	Option Description
`returnValue`	The return value is 1 if the recalculation took place and 0 if it did not.

Example:
The following script creates an EForm object, adds a label (`ELabel` object), then presents the form as a modal dialog box. Whenever the form is resized by the user, the label control is moved to be roughly halfway in the form.

```
. . .
Set gvFormHeight = 500;
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(gvFormHeight)
       OnSize(FormSizeEvent) CanResize(True);
New ELabel Form(efrm) Top(gvFormHeight/2) Left(50) Caption('My Label')
   Name('MyLabel');
Run efrm.ShowModal NewVar(gvRetValue);
. . .
Event FormSizeEvent
   Set EFormVar['MyLabel'].Top = EFormVar.Height/2;
   Run EFormVar.RecalcLayout;
EndEvent
. . .
```

BringToTop

Format:

```
Run efrm.BringToTop NewVar(returnValue)
or
Set returnValue = efrm.BringToTop{};
```

The BringToTop method brings the specified window to the top of the z-order.

Table 153: BringToTop Options

Option Name	Option Description
`returnValue`	The return value is 1 if the recalculation took place and 0 if it did not.

Control Objects

Controls are objects which are placed on forms. Controls cannot exist outside of a form. The following table lists the common properties for all controls. The individual control types are described afterward.

Common Properties for all Controls

Table 154: Common Properties for Controls

Property Name	Data Type	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137, the following properties are available for all control objects.	
`BackgroundColor`	`Integer`	The RGB value for the background for this control. Use the eUtl.GetRGB function to produce a color value.

Table 154: Common Properties for Controls

Property Name	Data Type	Property Description
Enabled	**Integer**	True if the control is currently enabled(Default); False otherwise.
ErrorCode	**Integer**	The ElmScript error code for the last method.
ErrorMsg	**String**	The Text of the last error.
ForegroundColor	**Integer**	The RGB value for the foreground for this control. Use the eUtl.GetRGB function to produce a color value.
Form	**EslObject**	The EForm object, which is the container for the control.
Name	**String**	The name of the control. Each control on a form should have a unique name to identify it.
NextControlInForm	**EslObject**	The control object of the next control on the form. *(Read-Only)*
OnKillFocus	**String**	The name of the event to run when the control loses focus. Default: Nothing happens when the control loses focus.
OnRightClick	**String**	The name of the event to run when the user right clicks the mouse button over the control. Default: Nothing happens when the user right clicks the mouse button.
OnSetFocus	**String**	The name of the event to run when the control gains focus. Default: Nothing happens when the control gains focus.
Parent	**EslObject**	The parent object of the control.
PrevControlInForm	**EslObject**	The control object of the previous control on the form. *(Read-Only)*
TooltipText	**String**	The text of the optional tool tip. This text will appear when you rest the mouse over the control. The EnableToolTips form property must be set. Some controls (e.g. ELabel) do not have tooltips.
Visible	**Integer**	True if the control is currently visible(Default); False otherwise.

Control Events

When a control event is called (such as OnRightClick) two parameters are passed to the event. The first is the form object variable (called EFormVar) and the second is the Control Object variable (EControlVar) of the control that caused the event. This allows you to use the same event subroutine to handle events in different forms and controls.

EPanel Object

The EPanel object is a container for form control objects.

Creating the object

You create an instance of this object using the New command.

Format:

```
New EPanel {Form(formObject) or Panel(panelObj)} [PropertyName(PropertyValue)] ...
    [NewVar(panelVar)];
```

The New method creates a new EPanel object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 155: New EPanel Options

Option Name	Option Description
Form or Panel	Identifies the EForm (or EPanel) object associated with this control. *(Required)*
PropertyName/PropertyValue	A set of property names and values from "Common Properties for Controls" on page 144 or "EButton Properties" on page 148 or "Common Form Object Properties" on page 137
NewVar	The name of the variable to hold the newly created object.

Example 1:

The following script creates an `EButton` object after creating an EForm object. See the ButtonDemo.fsl script for a complete example.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New EPanel Form(efrm) Top(50) Left(50) Orientation('col') AllMargins(5)
       NewVar(panelVar);
. . .
```

Example 2:

The following script illustrates how to set the properties of the main panel.

```
. . .
New EForm NewVar(gvForm) Caption('Test Layouts')
   CanResize(True);
Set gvMainPanel=gvForm.mainPanel; // Get main panel
Set gvMainPanel.Orientation='col'; // Set to column orientation
Set gvMainPanel.SameWidths=True; // Make all the children the same width
. . .
```

EPanel Properties

Table 156: EPanel Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for EPanel objects.	
AllMargins	**Integer**	This is a write-only property that sets all of the margins for the panel to the specified value.
BottomMargin	**Integer**	This property is the bottom margin for the panel.
FirstControlInPanel	**Object**	This property retrurns the first child control in the panel.
HasBorder	**Integer**	True if this panel has a visible border, False otherwise.
HorzAlignment	**String**	The horizontal alignment of the child controls in the panel: 'left'(default) , 'right', 'fill' or 'center'

Table 156: EPanel Properties

Property Name	Data Typ;e	Property Description
IsMainPanel	**Integer**	True if this panel is the main panel, False otherwise.
LastControlInPanel	**Object**	The horizontal alignment: 'Left', 'Right', or 'Center' (default) of the caption inside the button.
LeftMargin	**Integer**	This property is the left margin for the panel.
Orientation	**String**	The placement orientation. The default value is ’direct’ meaning that the sizing and placement of all the controls are specified directly using the left,top,width, and height coordinates. These values are relative to the upper left corner of the panel (not the form itself). If the orientation is ’row’ or ’col’, the panel itself decides where to place each control. They are placed by the order in which they are added to the panel. If the orientation is ’row’ they are placed in a horizontal manner. If the orientation is ’col’ they are placed in a vertical manner. The AllMargins,BottomMargin,HorzAlignment,LeftMargin,RightMargin, Spacing and TopMargin properties have no effect if the Orientation is ’direct’
RightMargin	**Integer**	This property is the right margin for the panel.
Spacing	**Integer**	The amount of space between controls during placement.
SameHeights	**Integer**	True, if all controls in the panel have the same heights, False otherwise.
SameWidths	**Integer**	True, if all controls in the panel have the same widths, False otherwise.
TopMargin	**Integer**	This property is the top margin for the panel.
VertAlignment	**String**	The Vertical alignment of the child controls in the panel: 'top'(default), 'bottom', ’fill’ or 'center'.

EPanel Methods

Table 157: EPanel Methods

Method Name	Method Description
RecalcLayout	.

EPanel Method Descriptions

RecalcLayout

Format:

```
Run panelVar.RecalcLayout;
or
panelVar.RecalcLayout{};
```

The **RecalcLayout** method recalculates the placement and sizing of the child controls on the panel. This is usually performed after some child control properties have been changed, which will affect the layout.

EButton Object

The EButton object represents a simple push button control.

Creating the object

You create an instance of this object using the New command.

Format:

```
New EButton Form(formObjectVar) [PropertyName(PropertyValue)] ...
    [NewVar(btnVar)];
```

The New method creates a new EButton object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 158: New EButton Options

Option Name	Option Description
`Form`	Identifies the EForm object associated with this control. *(Required)*
`PropertyName/PropertyValue`	A set of property names and values from "Common Properties for Controls" on page 144 or "EButton Properties" on page 148 or "Common Form Object Properties" on page 137
`NewVar`	The name of the variable to hold the newly created object.

Example:

The following script creates an `EButton` object after creating an EForm object. See the ButtonDemo.fsl script for a complete example.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New EButton Form(efrm) Top(50) Left(50) Caption('My Button Caption')
       NewVar(btnVar);
. . .
```

EButton Properties

Table 159: EButton Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for EButton objects.	
`Bitmap`	`String`	The name of the file containing a bitmap, which will appear on the button.
`Caption`	`String`	The caption text to appear on the button.
`CloseFormWithValue`	`Integer`	Indicates that when this button is clicked, the containing form is marked for closing and the value indicated here will be returned.
`DefaultButton`	`Integer`	True if this button is the default button, False otherwise.
`HorzAlign`	`String`	The horizontal alignment: 'Left', 'Right', or 'Center' (default) of the caption inside the button.

Table 159: EButton Properties

Property Name	Data Typ;e	Property Description
Icon	**String**	The name of the file containing an icon, which will appear on the button.
MultiLine	**Integer**	True if this button can have a caption that extends over multiple lines; False otherwise.
OnClick	**String**	The name of the subroutine to run when this control is clicked by the user.
SystemIcon	**String**	The name of the system icon, which will appear on the button. The value must be one of the following: 'Error' 'Information' 'Warning' 'Question'.
VertAlign	**String**	The horizontal alignment: 'Top', 'Bottom', or 'Center' (default) of the caption inside the button.

ECheckBox Object

The ECheckBox object represents a checkbox control.

Creating the object

You create an instance of this object using the New command.

Format:

```
New ECheckBox Form(formObjectVar) [PropertyName(PropertyValue)] ...
    [NewVar(cbxVar)];
```

The New method creates a new ECheckBox object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 160: New ECheckBox Options

Option Name	Option Description
Form	Identifies the EForm object associated with this control. *(Required)*
PropertyName/PropertyValue	A set of property names and values from "Common Properties for Controls" on page 144 or "ECheckBox Properties" on page 150 or "Common Form Object Properties" on page 137
NewVar	The name of the variable to hold the newly created object.

Example:

The following script creates an `ECheckBox` object after creating an EForm object.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New ECheckBox Form(efrm) Top(50) Left(50) Caption('My Button Caption')
       NewVar(btnVar) TextPos('Left');
. . .
```

ECheckBox Properties

Table 161: ECheckBox Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for ECheckBox objects.	
`Caption`	`String`	The caption to appear beside the checkbox.
`CloseFormWithValue`	`Integer`	Indicates that when this control is clicked, the containing form is marked for closing and the value indicated here will be returned.
`IsChecked`	`Integer`	True if the checkbox is checked, False otherwise. The default value is False.
`OnClick`	`String`	The name of the subroutine to run when this control is clicked by the user.
`TextPos`	`String`	'Left' for the caption to be to the left of the checkbox or 'Right' for the caption to be on the right side. Default is for the caption to be on the right side.
`TriState`	`Integer`	True if the checkbox is can have three states; False otherwise. The default value is False.

ERadioButton Object

The ERadioButton object represents a radio button control.

Creating the object

You create an instance of this object using the New command.

Format:

```
New ERadioButton Form(formObjectVar) [PropertyName(PropertyValue)] ...
    [NewVar(ctlVar)];
```

The New method creates a new ERadioButton object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 162: New ERadioButton Options

Option Name	Option Description
`Form`	Identifies the EForm object associated with this control. *(Required)*
`PropertyName/PropertyValue`	A set of property names and values from "Common Properties for Controls" on page 144 or "Common Form Object Properties" on page 137
`NewVar`	The name of the variable to hold the newly created object.

Example:
The following script creates a several `ERadioButton` object in the same group after creating an EForm object.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New ERadioButton Form(efrm) Top(50) Left(50) Caption('My RB Caption 1')
       NewVar(btnVar1) Group(1);
New ERadioButton Form(efrm) Top(150) Left(50) Caption('My RB Caption 2')
       NewVar(btnVar2) Group(1);
New ERadioButton Form(efrm) Top(250) Left(50) Caption('My RB Caption 3')
       NewVar(btnVar3) Group(1);
. . .
```

ERadioButton Properties

Table 163: ERadioButton Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for ERadioButton objects.	
CloseFormWithValue	**Integer**	Indicates that when this control is clicked, the containing form is marked for closing and the value indicated here will be returned.
Caption	**String**	The caption to appear beside the radio button.
IsChecked	**Integer**	True if the radio button is checked, False otherwise. The default value is False.
Group	**Integer**	An integer indicating the group to which this radio button belongs. Radio buttons do not stand alone. They are group together with other radio buttons, only one of which is checked at any one time. When you click on one radio button in a group all the other radio buttons in the same group are unset.
TextPos	**String**	'Left' for the caption to be to the left of the radio button or 'Right' for the caption to be on the right side. Default is for the caption to be on the right side.
OnClick	**String**	The name of the subroutine to run when this control is clicked by the user.

ELabel Object

The ELabel object represents a line of static text.

Creating the object

You create an instance of this object using the New command.

Format:

```
New ELabel Form(formObjectVar) [PropertyName(PropertyValue)] ...
    [NewVar(ctlVar)];
```

The New method creates a new ELabel object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 164: New ELabel Options

Option Name	Option Description
Form	Identifies the EForm object associated with this control. *(Required)*
PropertyName/PropertyValue	A set of property names and values from "Common Properties for Controls" on page 144 or "ELabel Properties" on page 152 or "Common Form Object Properties" on page 137
NewVar	The name of the variable to hold the newly created object.

Example:
The following script creates an `ELabel` object after creating an EForm object. See the LabelDemo.fsl script for a complete example.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New ELabel Form(efrm) Top(50) Left(50) Caption('My Form Title');
. . .
```

ELabel Properties

Table 165: ELabel Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for ELabel objects.	
Align	**String**	The alignment of the caption: 'Left', 'Right', 'Center' or 'LeftNoWordWrap'.
Caption	**String**	The text to appear in the label.
EllipsisStyle	**String**	The method for truncating the caption (if necessary) into the rectangle and adding an ellipsis to indicate the missing text: Possible values are 'End' ellipsis at the end of the caption 'Path' ellipsis in the middle, especially useful for path names. 'Word' truncates text and adds ellipsis 'None' (Default).
Sunken	**Integer**	True if the label is drawn inside a sunken rectangle, False otherwise. The default value is False.

ERectCtrl Object

The ERectCtrl object represents a graphic rectangle with various shadings.

Creating the object

You create an instance of this object using the New command.

Format:

```
New ERectCtrl Form(formObjectVar) [PropertyName(PropertyValue)] ...
    [NewVar(ctlVar)];
```

The New method creates a new ERectCtrl object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 166: New ERectCtrl Options

Option Name	Option Description
Form	Identifies the EForm object associated with this control. *(Required)*
PropertyName/PropertyValue	A set of property names and values from "Common Properties for Controls" on page 144 or "ERectCtrl Properties" on page 153 or "Common Form Object Properties" on page 137
NewVar	The name of the variable to hold the newly created object.

Example:

The following script creates an `ERectCtrl` object after creating an EForm object. The rectangle is a black square and if the user clicks on it a subroutine names RectClicked is run. See the RectCtrlDemo.fsl script for a complete example.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New ERectCtrl Form(efrm) Top(50) Left(50) Width(50) Height(50)
       Style('BlackRect') OnClick('RectClicked');
. . .
```

ERectCtrl Properties

Table 167: ERectCtrl Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for ERectCtrl objects.	
OnClick	**String**	The name of the subroutine to run when this control is clicked by the user.
OnDblClick	**String**	The name of the subroutine to run when this control is double clicked by the user.
Style	**String**	The style for the rectangle. Possible values are 'BlackRect' black rectangle 'GrayRect' gray rectangle. 'WhiteRect' black rectangle 'BlackFrame' black outline 'GrayFrame' gray outline. 'Whiteframe' black outline 'EtchedFrame' sunken frame (Default) 'EtchedHorz' top and bottom lines are etched. 'EtchedVert' left and right lines are etched.

EImageBox Object

The EImageBox object represents a graphic (BMP or ICON) picture box.

Creating the object

You create an instance of this object using the New command.

Format:

```
New EImageBox Form(formObjectVar) [PropertyName(PropertyValue)] ...
    [NewVar(ctlVar)];
```

The New method creates a new EImageBox object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 168: New EImageBox Options

Option Name	Option Description
Form	Identifies the EForm object associated with this control. *(Required)*
PropertyName/PropertyValue	A set of property names and values from "Common Properties for Controls" on page 144 or "EImageBox Properties" on page 154 or "Common Form Object Properties" on page 137
NewVar	The name of the variable to hold the newly created object.

Example:

The following script creates an `EImageBox` object after creating an EForm object. The image box will show the image in the file 'MyBitMap.bmp'. See the ImageBoxDemo.fsl script for a complete example.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New EImageBox Form(efrm) Top(50) Left(50) Width(50) Height(50)
    BitMap('MyBitMap.bmp');
. . .
```

EImageBox Properties

Table 169: EImageBox Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for EImageBox objects.	
Bitmap	**String**	The name of the file containing a bitmap (.bmp), which will appear in the image.
Centered	**Integer**	True if the image is centered inside the rectangle, False otherwise. The default value is False.
Icon	**String**	The name of the file containing a icon (.ico), which will appear in the image.
OnClick	**String**	The name of the subroutine to run when this control is clicked by the user.

Table 169: EImageBox Properties

Property Name	Data Typ;e	Property Description
OnDblClick	**String**	The name of the subroutine to run when this control is double clicked by the user.
SystemIcon	**String**	The name of the system icon, which will appear on the button. The value must be one of the following: 'Error' 'Information' 'Warning' 'Question'.

EAnimateCtrl Object

The EAnimateCtrl object represents a graphic (BMP or ICON) picture box.

Creating the object

You create an instance of this object using the New command.

Format:

```
New EAnimateCtrl Form(formObjectVar) [PropertyName(PropertyValue)] ...
    [NewVar(ctlVar)];
```

The New method creates a new EAnimateCtrl object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 170: New EAnimateCtrl Options

Option Name	Option Description
Form	Identifies the EForm object associated with this control. *(Required)*
PropertyName/PropertyValue	A set of property names and values from "Common Properties for Controls" on page 144 or "EAnimateCtrl Properties" on page 156 or "Common Form Object Properties" on page 137
NewVar	The name of the variable to hold the newly created object.

Example:

The following script creates an `EAnimateCtrl` object after creating an EForm object. The animate control will show the movie centered within the control. See the AnimateCtrlDemo.fsl script for a complete example.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New EAnimateCtrl Form(efrm) Top(50) Left(50) Width(50) Height(50)
    Centered(True);
. . .
```

EAnimateCtrl Properties

Table 171: EAnimateCtrl Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for EAnimateCtrl objects.	
AutoPlay	**Integer**	True if the movie will run continuously, False otherwise. The default value is False.
Centered	**Integer**	True if the movie is centered inside the rectangle, False otherwise. The default value is False.
OnStartPlay	**String**	The name of the subroutine to run when movie starts playing.
OnStopPlay	**String**	The name of the subroutine to run when movie stops playing.
Transparent	**Integer**	True if the background is transparent, False otherwise. The default value is False.

EAnimateCtrl Methods

Table 172: EAnimateCtrl Methods

Method Name	Method Description
Close	Closes the current movie (.avi) file.
Open	Opens a movie (.avi) file.
Play	Plays the current movie.
Seek	Displays a selected single frame of the avi file.
Stop	Stops playing the clip.

EAnimateCtrl Method Descriptions

Close

Format:

```
Run eAnimateVar.Close;
or
eAnimateVar.Close{};
```

The Close method closes the current movie.

Open

Format:

```
Run eAnimateVar.Open Filename(fileNameString);
or
eAnimateVar.Open{filename);
```

The Open method opens a movie file (.avi) and makes it available for the control.

Table 173: EAnimateCtrl Open Options

Option Name	Option Description
Filename	A string expression specifying the file name of the avi movie file.

Example:
The following script creates an `EAnimateCtrl` object after creating an EForm object. The control will open the movie file during the form initialize stage.

```
. . .
New EForm NewVar(gvForm) Top(100) Left(100) Width(300) Height(500)
        OnInitForm('InitMyForm');
New EAnimateCtrl Form(gvForm) Top(50) Left(50) Width(200) Height(200)
       Centered(True) AutoPlay(True) NewVar(gvAnimateVar);

Run gvForm.ShowModal

Event InitMyForm
   Run gvAnimateVar.Open File('MyMovie.avi');
   Run gvAnimateVar.Play;
EndEvent
. . .
```

Play

Format:

```
Run eAnimateVar.Play [From(startFrame)] [To(endFrame)] [RepCount(repCount)];
or
eAnimateVar.Play{[startFrame[,endFrame[,repCount]]]);
```

The Play method starts a movie clip previously opened.

Table 174: EAnimateCtrl Play Options

Option Name	Option Description
startFrame	The integer starting frame number (the first frame is 0). The default is to start at the beginning (frame 0).
endFrame	The integer ending frame number. The default is to stop at the end.
repCount	The integer number of times to repeat the clip. The default value is 1.

For an example, see the Open method.

Seek

Format:

```
Run eAnimateVar.Seek To(seekFrame);
or
eAnimateVar.Seek{seekFrame);
```

The Seek method displays the indicated Frame in the control. Frame numbers start at 0.

Table 175: EAnimateCtrl Seek Options

Option Name	Option Description
seekFrame	The integer value of the frame number to display.

Stop

Format:

```
Run eAnimateVar.Stop;
or
eAnimateVar.Stop{};
```

The Stop method stops the current movie clip.

EGroupBox Object

The EGroupBox object represents a box for visually organizing a group of controls.

Creating the object

You create an instance of this object using the New command.

Format:

```
New EGroupBox Form(formObjectVar) [PropertyName(PropertyValue)] ...
    [NewVar(ctlVar)];
```

The New method creates a new EGroupBox object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 176: NEW EGroupBox Options

Option Name	Option Description
Form	Identifies the EForm object associated with this control. *(Required)*
PropertyName/PropertyValue	A set of property names and values from "Common Properties for Controls" on page 144 or "EGroupBox Properties" on page 159 or "Common Form Object Properties" on page 137
NewVar	The name of the variable to hold the newly created object.

Example:
The following script creates an `ELabel` object after creating an EForm object.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New EGroupBox Form(efrm) Top(50) Left(50) Width(200) Height(200)
       Caption('Box of Radio Buttons');
New ERadioButton Form(efrm) Top(60) Left(60) Caption('My Group RB 1')
       NewVar(btnVar1) Group(1);
New ERadioButton Form(efrm) Top(150) Left(60) Caption('My Group RB 2')
       NewVar(btnVar2) Group(1);
. . .
```

EGroupBox Properties

Table 177: EGroupBox Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for ELabel objects.	
`Caption`	`String`	A caption to appear on the border of the box.

EEdit Object

The EEdit object represents an editable, single line text field. This is commonly used for data entry.

Creating the object

You create an instance of these objects using the New command.

Format:

```
New EEdit Form(formObjectVar) [PropertyName(PropertyValue)] ...
    [NewVar(ctlVar)];
```

The New method creates a new EEdit object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 178: New EEdit Options

Option Name	Option Description
`Form`	Identifies the EForm object associated with this control. *(Required)*
`PropertyName/PropertyValue`	A set of property names and values from "Common Properties for Controls" on page 144 or "EEdit Properties" on page 160 or "Common Form Object Properties" on page 137
`NewVar`	The name of the variable to hold the newly created object.

Example:
The following script creates an `EEdit` object after creating an EForm object.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New EEdit Form(efrm) Top(50) Left(50) Width(200) Height(25) Text('My Initial Text');
. . .
```

EEdit Properties

Table 179: EEdit Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for EEdit objects.	
Text	**String**	The text in the edit field.
SelectedText	**String**	The text in the edit field that is marked (either by the user or via properties) as selected.
IsModified	**Integer**	True if the field has been modified, False otherwise. The initial value is False.
HorzAutoScroll	**Integer**	True if the field will scroll to the right as text is entered, False if the text is limited to the size of the field. The default value is False.
NumChars	**Integer**	This property allows you to specify the width of the edit control using characters instead of pixels. This is an estmated value based on font average character width.
StartSel	**Integer**	The number of the starting character in the selection.
EndSel	**Integer**	The number of the ending character in the selection.
OnChange	**String**	The name of the event to run when a user changes(adds, deletes or replaces) the text in this control.

EEdit Methods

Table 180: EEdit Methods

Method Name	Method Description
Clear	Clears the selection in the field.
Cut	Clears the selection in the field and sends the information to the clipboard.
Copy	Copies the selection in the field and sends the information to the clipboard.
Paste	Pastes the information from the clipboard to the field at current insertion point.
Undo	Undo the last change.
SelectAll	Cause all the text in the edit control to be selected.
InsertText	Inserts the specified text into the control at the current insertion point. If text has been selected, then the text will be replaced by the new text.

EEdit Method Descriptions

Clear

Format:

```
Run editCtl.Clear;
or
editCtl.Clear{};
```

The Clear method clears the selected text from the edit control.

Cut

Format:

```
Run editCtl.Cut;
or
editCtl.Cut{};
```

The Cut method clears the selected text from the edit control and copies it to the clipboard.

Copy

Format:

```
Run editCtl.Copy;
or
editCtl.Copy{};
```

The Copy method copies the selected text to the clipboard.

Paste

Format:

```
Run editCtl.Paste;
or
editCtl.Paste{};
```

The Paste method copies text from the clipboard to the edit control at the current insertion point.

Undo

Format:

```
Run editCtl.Undo;
or
editCtl.Undo{};
```

The Undo method removes the last change made.

SelectAll

Format:

```
Run editCtl.SelectAll;
or
editCtl.SelectAll{};
```

The SelectAll method selects all the text in the edit control.

InsertText

Format:

```
Run editCtl.InsertText Text(TextExpression);
or
editCtl.InsertText{TextExpression};
```

The InsertText method inserts the specified text to the edit control at the current insertion point.

Table 181: InsertText Options

Option Name	Option Description
Text	The text to insert *(Required)*

EMEdit Object

The EMEdit object represents an editable, multi-line, word wrapping text field.

Creating the object

You create an instance of these objects using the New command.

Format:

```
New EMEdit Form(formObjectVar) [PropertyName(PropertyValue)] ...
    [NewVar(ctlVar)];
```

The New method creates a new EMEdit object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 182: New EMEdit Options

Option Name	Option Description
Form	Identifies the EForm object associated with this control. *(Required)*
PropertyName/PropertyValue	A set of property names and values from "Common Properties for Controls" on page 144 or "ERichEdit Properties" on page 166 or "Common Form Object Properties" on page 137
NewVar	The name of the variable to hold the newly created object.

Example:

The following script creates an `EEdit` object after creating an EForm object.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New EMEdit Form(efrm) Top(100) Left(50) Width(200) Height(25);
. . .
```

EMEdit Properties

Table 183: EMEdit Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for EMEdit objects.	
Text	**String**	The text in the edit field.
SelectedText	**String**	The text in the edit field that is marked (either by the user or via properties) as selected.
IsModified	**Integer**	True if the field has been modified, False otherwise. The initial value is False.
HorzAutoScroll	**Integer**	True if the field will scroll to the right as text is entered, False if the text will go to the next line word wrapping as necessary. The default value is False.
StartSel	**Integer**	The number of the starting character in the selection.
EndSel	**Integer**	The number of the ending character in the selection.
LineCount	**Integer**	The number of lines in the edit control.
CurrentLine	**Integer**	The current line (Line containing the cursor) in the edit control. Setting this property also sets the CurrentCol property to 1. 1 is the first line and LineCount is the last line.
CurrentCol	**Integer**	The current column in the edit control.
OnChange	**String**	The name of the event to run when a user changes(adds, deletes or replaces) the text in this control.

EMEdit Methods

Table 184: EMEdit Methods

Method Name	Method Description
Clear	Clears the selection in the field.
Cut	Clears the selection in the field and sends the information to the clipboard.
Copy	Copies the selection in the field and sends the information to the clipboard.
Paste	Pastes the information from the clipboard to the field at current insertion point.
Undo	Undo the last change.
SelectAll	Cause all the text in the EMEdit control to be selected.
GetLine	Gets the text from the specified line.
InsertText	Inserts the specified text into the control at the current insertion point. If text has been selected, then the text will be replaced by the new text.

EMEdit Method Descriptions

Clear

Format:

```
Run editCtl.Clear;
or
editCtl.Clear{};
```

The Clear method clears the selected text.

Cut

Format:

```
Run editCtl.Cut;
or
editCtl.Cut{};
```

The Cut method clears the selected text and copies it to the clipboard.

Copy

Format:

```
Run editCtl.Copy;
or
editCtl.Copy{};
```

The Copy method copies the selected text to the clipboard.

Paste

Format:

```
Run editCtl.Paste;
or
editCtl.Paste{};
```

The Paste method copies text from the clipboard at the current insertion point.

Undo

Format:

```
Run editCtl.Undo;
or
editCtl.Undo{};
```

The Undo method removes the last change made.

SelectAll

Format:

```
Run editCtl.SelectAll;
or
editCtl.SelectAll{};
```

The SelectAll method selects all the text in the control.

GetLine

Format:

```
Run editCtl.GetLine Number(lineNumber) NewVar(lineTextVar);
or
Set lineTextVar = editCtl.GetLine{lineNumber};
```

The GetLine method gets a line of text (specified by the 'Number' option) the control.

Table 185: GetLine Options

Option Name	Option Description
Number	The number of the line to get from the multi-line edit control *(Required)*
NewVar	The name of the variable to hold the text line.

InsertText

Format:

```
Run editCtl.InsertText Text(TextExpression);
or
editCtl.InsertText{TextExpression};
```

The InsertText method inserts the specified text to the edit control at the current insertion point.

Table 186: InsertText Options

Option Name	Option Description
Text	The text to insert *(Required)*

ERichEdit Object

The ERichEdit object represents an editable, multi-line text field. It is used for memos, notes or text editing.

Creating the object

You create an instance of these objects using the New command.

Format:

```
New ERichEdit Form(formObjectVar) [PropertyName(PropertyValue)] ...
    [NewVar(ctlVar)];
```

The New method creates a new ERichEdit object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 187: New ERichEdit Options

Option Name	Option Description
Form	Identifies the EForm object associated with this control. *(Required)*
PropertyName/PropertyValue	A set of property names and values from "Common Properties for Controls" on page 144 or "ERichEdit Properties" on page 166 or "Common Form Object Properties" on page 137
NewVar	The name of the variable to hold the newly created object.

Example:
The following script creates an `EEdit` object after creating an EForm object.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New ERichEdit Form(efrm) Top(100) Left(50) Width(200) Height(25);
. . .
```

ERichEdit Properties

Table 188: ERichEdit Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for EMEdit objects.	
Text	**String**	The text in the edit field.
SelectedText	**String**	The text in the edit field that is marked (either by the user or via properties) as selected.
IsModified	**Integer**	True if the field has been modified, False otherwise. The initial value is False.
StartSel	**Integer**	The number of the starting character in the selection.
EndSel	**Integer**	The number of the ending character in the selection.
LineCount	**Integer**	The number of lines in the edit control.
CurrentLine	**Integer**	The current line (Line containing the cursor) in the edit control. Setting this property also sets the CurrentCol property to 1. 1 is the first line and LineCount is the last line.
CurrentCol	**Integer**	The current column in the edit control.
OnChange	**String**	The name of the event to run when a user changes(adds, deletes or replaces) the text in this control.
OnSelChange	**String**	The name of the event to run when a user changes the selection (or cursor position) in this control.

ERichEdit Methods

Table 189: ERichEdit Methods

Method Name	Method Description
Clear	Clears the selection.
Cut	Clears the selection and sends the information to the clipboard.
Copy	Copies the selection and sends the information to the clipboard.
Paste	Pastes the information from the clipboard at current insertion point.
PasteRTF	Pastes the information from the clipboard at current insertion point with formatting information.
Undo	Undo the last change.
SelectAll	Cause all the text in the ERichEdit control to be selected.
GetLine	Gets the text from the specified line.
InsertText	Inserts the specified text into the control at the current insertion point. If text has been selected, then the text will be replaced by the new text.
FindText	Finds a string of text in the edit control.

ERichEdit Method Descriptions

Clear

Format:

```
Run editCtl.Clear;
or
editCtl.Clear{};
```

The Clear method clears the selected text.

Cut

Format:

```
Run editCtl.Cut;
or
editCtl.Cut{};
```

The Cut method clears the selected text and copies it to the clipboard.

Copy

Format:

```
Run editCtl.Copy;
or
editCtl.Copy{};
```

The Copy method copies the selected text to the clipboard.

Paste

Format:

```
Run editCtl.Paste;
or
editCtl.Paste{};
```

The Paste method copies text from the clipboard at the current insertion point.

PasteRTF

Format:

```
Run editCtl.PasteRTF;
or
editCtl.PasteRTF{};
```

The PasteRTF method copies text from the clipboard at the current insertion point saving formatting information.

Undo

Format:

```
Run editCtl.Undo;
or
editCtl.Undo{};
```

The Undo method removes the last change made.

SelectAll

Format:

```
Run editCtl.SelectAll;
or
editCtl.SelectAll{};
```

The SelectAll method selects all the text in the control.

GetLine

Format:

```
Run editCtl.GetLine Number(lineNumber) NewVar(lineTextVar);
or
Set lineTextVar = editCtl.GetLine{lineNumber};
```

The GetLine method gets a line of text (specified by the 'Number' option) the control.

Table 190: GetLine Options

Option Name	Option Description
`Number`	The number of the line to get from the multi-line edit control *(Required)*
`NewVar`	The name of the variable to hold the text line.

InsertText

Format:

```
Run editCtl.InsertText Text(TextExpression);
or
editCtl.InsertText{TextExpression};
```

The InsertText method inserts the specified text to the edit control at the current insertion point.

Table 191: InsertText Options

Option Name	Option Description
`Text`	The text to insert *(Required)*

FindText

Format:

```
Run editCtl.FindText Text(textString) [StartPos(charPosition)]
    [Case][NoCase][Word][NoWord] NewVar(charPos)
or
Set charPos = editCtl.FindText{textString[,charposition][,'Case'][,'NoCase']
    [,'Word'][,'NoWord']};
```

The FindText method searches for the specified text in the control.

Table 192: FindText Options

Option Name	Option Description
`Text`	The text to find *(Required)*
`StartPos`	The character position to start searching. Default: use the character of the insertion point
`Case`	The search is case sensitive. This is the default.
`NoCase`	The search is case insensitive. Default: Case
`Word`	The search looks for whole words only. Default: NoWord
`NoWord`	The search looks for any occurrence of the string. This is the default.
`charPos`	The character position returned. -1, if not found.

EListBox Object

The EListBox object represents a list of items in a rectangular box.

Creating the object

You create an instance of this object using the New command.

Format:

```
New EListBox Form(formObjectVar) [PropertyName(PropertyValue)] ...
    [NewVar(ctlVar)];
```

The New method creates a new EListBox object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 193: New EListBox Options

Option Name	Option Description
Form	Identifies the EForm object associated with this control. *(Required)*
PropertyName/PropertyValue	A set of property names and values from "Common Properties for Controls" on page 144 or "EListBox Properties" on page 170 or "Common Form Object Properties" on page 137
NewVar	The name of the variable to hold the newly created object.

Example:

The following script creates an `EListBox` object after creating an EForm object. The Box will contain three strings. See the ListBoxDemo.fsl script for a complete example.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New StringList NewVar(sList) Value('Item 1') Value('Item 2') Value('Item 3');
New EListBox Form(efrm) Top(50) Left(50) Width(200) Height(200)
       Strings(sList);
. . .
```

EListBox Properties

Table 194: EListBox Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for EListBox objects.	
ExtendedSelect	**Integer**	True if the user can select multiple items using the Shift key, False otherwise. The default value is False.
LineCount	**Integer**	The number of strings in the box.
MultiSelect	**Integer**	True if multiple selections are allowed, False otherwise. The default value is False. This value has to be set before the form is displayed. Once the form is on the screen (ShowModal or ShowModeless), then setting this has no effect.
NoSelect	**Integer**	True specifies that the user can view items but cannot select them, False otherwise. The default value is False.
NumChars	**Integer**	This property allows you to specify the width of the listbox control using characters instead of pixels. This is an estmated value based on font average character width.
NumLines	**Integer**	This property allows you to specify the height of the listbox control using line count instead of pixels. This is an estmated value based on font average character height.
OnDblClick	**String**	The name of the subroutine to run when a user double clicks on an item.
OnSelChange	**String**	The name of the subroutine to run when a user changes the selected item in this control.

Table 194: EListBox Properties

Property Name	Data Typ;e	Property Description
`SelCount`	`Integer`	The number of selected strings in the box. Usually used when MultiSelect is on.
`SelIndex`	`Integer/ IntList`	The number(s) of the selected item(s). When Multi-Select is off, this value is an integer indicating the number of the selected item. When Multi-Select is on, this value is an IntList, where each integer in the list is a number of a selected item.
`SelStrings`	`StringList`	The selected strings in the box. Usually used when MultiSelect is on.
`Sorted`	`Integer`	True if the list is sorted, False otherwise. The default value is False.
`Strings`	`StringList`	The strings in the box.
`TopIndex`	`Integer`	The number of the item at the visible top of the box.
`VertScroll`	`Integer`	True to include a vertical scrollbar, False otherwise. Default:True

EListBox Methods

Table 195: EListBox Methods

Method Name	Method Description
`Clear`	Removes all the strings from the box.
`AddString`	Add a string to the list of strings.
`DeleteString`	Deletes a string from the list.
`GetString`	Gets a string in the list by member number.
`FindString`	Finds a string in the list.
`SelectString`	Selects the string in the list by the specified member number. Usually used when MultiSelect is on.

EListBox Method Descriptions

Clear

Format:

```
Run eListBoxVar.Clear;
or
eListBoxVar.Clear{};
```

The Clear method clears all the strings from the listbox control.

AddString

Format:

```
Run eListBoxVar.AddString Value(stringExpression) [Before(beforeIndex)];
or
eListBoxVar.AddString{string[,beforeIndex]);
```

The AddString method adds a new string to the listbox. If the Before option is not given (or if the value is 0), the new string is added to the end of the list, unless the list is sorted, in which case it adds it to the proper place.

Table 196: EListBox AddString Options

Option Name	Option Description
Value	A string expression specifying the string value to add.
Before	The member number before which to add the string. Use 0 (or omit this option) to add to the end of the list. For unsorted lists only.

Example:
The following script creates an `EListBox` object after creating an EForm object. The Box will contain four strings, after the new one is added.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New StringList NewVar(sList) Value('Item 1') Value('Item 2') Value('Item 3');
New EListBox Form(efrm) Top(50) Left(50) Width(200) Height(200)
       Strings(sList) NewVar(listBoxVar);
Run listBoxVar.AddString Value('My New Item');
. . .
```

DeleteString

Format:

```
Run eListBoxVar.DeleteString Number(memberNumber)
or
eListBoxVar.DeleteString{memberNumber};
```

The DeleteString method deletes the specified string from the listbox.

Table 197: EListBox DeleteString Options

Option Name	Option Description
Number	The member number to delete.

Example:
The following script creates an `EListBox` object after creating an EForm object. The Box will contain two strings after the second member is deleted.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New StringList NewVar(sList) Value('Item 1') Value('Item 2') Value('Item 3');
New EListBox Form(efrm) Top(50) Left(50) Width(200) Height(200)
       Strings(sList) NewVar(listBoxVar);
Run listBoxVar.DeleteString Number(2);
. . .
```

GetString

Format:

```
Run eListBoxVar.GetString Number(memberNumber) NewVar(stringItem);
or
Set stringItem = eListBoxVar.GetString{memberNumber}
```

The GetString method gets the string of the specified member number from the listbox.

Table 198: EListBox GetString Options

Option Name	Option Description
Number	The member number of the string to get.
NewVar	The string of the specified member number.

Example:
The following script creates an EListBox object after creating an EForm object, then gets a member in the list.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New StringList NewVar(sList) Value('Item 1') Value('Item 2') Value('Item 3');
New EListBox Form(efrm) Top(50) Left(50) Width(200) Height(200)
       Strings(sList) NewVar(listBoxVar);
Run listBoxVar.GetString Number(2) NewVar(memString);
. . .
```

FindString

Format:

```
Run eListBoxVar.FindString Value(stringExpression) NewVar(memberNumber);
or
Set memberNumber = eListBoxVar.FindString{stringExpression};
```

The FindString method finds the member number of the specified string from the listbox.

Table 199: EListBox FindString Options

Option Name	Option Description
Value	The string to find.
NewVar	The member number of the string or 0 if not found.

Example 1:
The following script creates an EListBox object after creating an EForm object, then finds a member in the list. The memnum variable should have the value 2 afterward.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New StringList NewVar(sList) Value('Item 1') Value('Item 2') Value('Item 3');
New EListBox Form(efrm) Top(50) Left(50) Width(200) Height(200)
       Strings(sList) NewVar(listBoxVar);
Run listBoxVar.FindString Value('Item 2') NewVar(memnum);
. . .
```

SelectString

Format:

```
Run eListBoxVar.SelectString Number(memberNumber) [Value(sel)];
or
eListBoxVar.SelectString{memberNumber[,sel]};
```

The SelectString method selects the string of the specified member number in the listbox.

Table 200: EListBox SelectString Options

Option Name	Option Description
Number	The member number of the string to select. Use 0 (zero) to select or unselect all items.
sel	True if the member is to be selected and False to Unselect the member. This is for multi-select list boxes.

Example:
The following script creates a multi-select `EListBox` object after creating an EForm object, then selects two members in the list.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New StringList NewVar(sList) Value('Item 1') Value('Item 2') Value('Item 3');
New EListBox Form(efrm) Top(50) Left(50) Width(200) Height(200) MultiSelect(True)
      Strings(sList) NewVar(listBoxVar);
Run listBoxVar.SelectString Number(2);
Run listBoxVar.SelectString Number(3);
. . .
```

EDropDownBox Object

The EDropDownBox object represents a list of items in a box that drops down when you click on the arrow.

Creating the object

You create an instance of this object using the New command.

Format:

```
New EDropDownBox Form(formObjectVar) [PropertyName(PropertyValue)] ...
   [NewVar(ctlVar)];
```

The New method creates a new EDropDownBox object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 201: NEW EDropDownBox Options

Option Name	Option Description
Form	Identifies the EForm object associated with this control. *(Required)*
PropertyName/PropertyValue	A set of property names and values from "Common Properties for Controls" on page 144 or "EDropDownBox Properties" on page 175 or "Common Form Object Properties" on page 137
NewVar	The name of the variable to hold the newly created object.

Example:
The following script creates an `EDropDownBox` object after creating an EForm object. The Box will contain three strings. See the DropDownDemo.fsl script for a complete example.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New StringList NewVar(sList) Value('Item 1') Value('Item 2') Value('Item 3');
New EDropDownBox Form(efrm) Top(50) Left(50) Width(200) Height(200)
       Strings(sList);
. . .
```

EDropDownBox Properties

Table 202: EDropDownBox Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for EDropDownBox objects.	
EditBox	**Integer**	True if the list has an editable text field, False otherwise. The default value is False.
LineCount	**Integer**	The number of lines in the box.
OnSelChange	**String**	The name of the event to run when a user changes the selected item in this control.
SelIndex	**Integer**	The line number of the selected item.
Sorted	**Integer**	True if the list is sorted, False otherwise. The default value is False.
Strings	**StringList**	The strings in the box.
Text	**String**	The text from the Editbox. Value only if EditBox is True.
TopIndex	**Integer**	The line number of the item at the visible top of the box.

EDropDownBox Methods

Table 203: EDropDownBox Methods

Method Name	Method Description
Clear	Removes all the strings from the box.
AddString	Add a string to the list of strings.
DeleteString	Deletes a string from the list.
GetString	Gets a string in the list by member number.
FindString	Finds a string in the list.
SelectString	Selects the string in the list by the specified member number.

EDropDownBox Method Descriptions

Clear

Format:

```
Run eDropDownBoxVar.Clear;
```

The Clear method clears all the strings from the control.

AddString

Format:

```
Run eDropDownBoxVar.AddString Value(stringExpression)
    [Before(memberNum)];
or
eDropDownBoxVar.AddString{stringExpression[,memberNum]};
```

The AddString method adds a new string to the control. If the Before option is not given the new string is added to the end of the list, unless the list is sorted, in which case it adds it to the proper place.

Table 204: EDropDownBox AddString Options

Option Name	Option Description
Value	A string expression specifying the string value to add.
Before	The member number before which to add the string. For unsorted lists only. Omit or use 0 (zero) to add to the end of the list.

Example:
The following script creates an `EDropDownBox` object after creating an EForm object. The Box will contain four strings, after the new one is added.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New StringList NewVar(sList) Value('Item 1') Value('Item 2') Value('Item 3');
New EDropDownBox Form(efrm) Top(50) Left(50) Width(200) Height(200)
       Strings(sList) NewVar(dropDownBoxVar);
Run dropDownBoxVar.AddString Value('My New Item');
. . .
```

DeleteString

Format:

```
Run eDropDownBoxVar.DeleteString Number(memberNumber)
or
eDropDownBoxVar.DeleteString{memberNumber};
```

The DeleteString method deletes the specified string from the listbox..

Table 205: EDropDownBox DeleteString Options

Option Name	Option Description
Number	The member number to delete.

Example:
The following script creates an `EDropDownBox` object after creating an EForm object. The Box will contain two strings after the second member is deleted.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New StringList NewVar(sList) Value('Item 1') Value('Item 2') Value('Item 3');
New EDropDownBox Form(efrm) Top(50) Left(50) Width(200) Height(200)
      Strings(sList) NewVar(dropDownBoxVar);
Run dropDownBoxVar.DeleteString Number(2);
. . .
```

GetString

Format:

```
Run eDropDownBoxVar.GetString Number(memberNumber) NewVar(stringItem);
or
Set stringItem = eDropDownBoxVar.GetString{memberNumber};
```

The GetString method gets the string of the specified member number from the control.

Table 206: EDropDownBox GetString Options

Option Name	Option Description
Number	The member number of the string to get.
NewVar	The string of the specified member number.

Example:
The following script creates an `EDropDownBox` object after creating an EForm object, then gets a member in the list.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New StringList NewVar(sList) Value('Item 1') Value('Item 2') Value('Item 3');
New EDropDownBox Form(efrm) Top(50) Left(50) Width(200) Height(200)
      Strings(sList) NewVar(dropDownBoxVar);
Run dropDownBoxVar.GetString Number(2) NewVar(memString);
. . .
```

FindString

Format:

```
Run dropDownBoxVar.FindString Value(stringExpression)
      NewVar(memberNumber);
or
Set memberNumber = dropDownBoxVar.FindString{stringExpression};
```

The FindString method finds the member number of the specified string from the control.

Table 207: EDropDownBox FindString Options

Option Name	Option Description
Value	The string to find.
NewVar	The member number of the string or 0 if not found.

Example 1:
The following script creates an `EDropDownBox` object after creating an EForm object, then finds a member in the list. The memnum variable should have the value 2 afterward.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New StringList NewVar(sList) Value('Item 1') Value('Item 2') Value('Item 3');
New EDropDownBox Form(efrm) Top(50) Left(50) Width(200) Height(200)
       Strings(sList) NewVar(dropDownBoxVar);
Set memberNumber = dropDownBoxVar.FindString{'Item 2'};
. . .
```

SelectString

Format:

```
Run dropDownBoxVar.SelectString Number(memberNumber);
or
dropDownBoxVar.SelectString{memberNumber};
```

The SelectString method selects the string of the specified member number in the control.

Table 208: EDropDownBox SelectString Options

Option Name	Option Description
`Number`	The member number of the string to select.

ETabCtrl Object

The ETabCtrl object represents a list of tabs that allow you to have panels in a dialog box.

Creating the object

You create an instance of this object using the New command.

Format:

```
New ETabCtrl Form(formObjectVar) [PropertyName(PropertyValue)] ...
    [NewVar(ctlVar)];
```

The New method creates a new ETabCtrl object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 209: New ETabCtrl Options

Option Name	Option Description
`Form`	Identifies the EForm object associated with this control. *(Required)*
`PropertyName/PropertyValue`	A set of property names and values from "Common Properties for Controls" on page 144 or "ETabCtrl Properties" on page 179 or "Common Form Object Properties" on page 137
`NewVar`	The name of the variable to hold the newly created object.

Example:
The following script creates an `ETabCtrl` object after creating an EForm object. The Tab list will have three tabs. See the TabCtrlDemo.fsl script for a complete example.

```
. . .
New EForm NewVar(gvForm) Top(100) Left(100) Width(300) Height(500);
New StringList NewVar(sList) Value('Item 1') Value('Item 2') Value('Item 3');
New ETabCtrl Form(gvForm) Top(50) Left(50) Width(200) Height(25)
       Strings(sList);
. . .
```

ETabCtrl Properties

Table 210: ETabCtrl Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for EDropDownBox objects.	
Buttons	**Integer**	True if the tabs have the form of buttons. False otherwise.
CancelOp	**Integer**	An indicator that you may set during the OnSelChanging event for this control to cancel the change operation.
FlatButtons	**Integer**	True if the tab buttons should have a flat appearance. False otherwise. This is only valid if the Buttons flag is also set to True.
MultiLine	**Integer**	True if the tabs can have a caption that extends over multiple lines; False otherwise.
OnSelChange	**String**	The name of the event to run when a user changes the selected tab in this control.
OnSelChanging	**String**	The name of the event to run when tab is in the process of changing.
RightJustify	**Integer**	True if the text is right justified. False otherwise.
SelTab	**Integer**	The line number of the selected item.
Strings	**StringList**	The list of strings for the tabs.
TabCount	**Integer**	The number of lines in the box.

ETabCtrl Methods

Table 211: ETabCtrl Methods

Method Name	Method Description
Clear	Removes all the tabs.
AddTab	Add a tab to the control.
DeleteTab	Deletes a tab from the control.
GetTabText	Gets the text of the tab in the control by member number.
SetTabText	Sets the text of the tab in the control by member number.

ETabCtrl Method Descriptions

Clear

Format:

```
Run gvTabCtrlVar.Clear;
```

The Clear method clears all the tabs from the control.

AddTab

Format:

```
Run gvTabCtrlVar.AddString Value(stringExpression)
    [Before(memberNum)];
or
gvTabCtrlVar.AddTab{stringExpression[,memberNum]};
```

The AddTab method adds a new tab to the control. If the Before option is not given the new tab is added to the end of the list of tabs.

Table 212: ETabCtrl AddTab Options

Option Name	Option Description
Value	A string expression specifying the string value of the tab to add.
Before	The member number before which to add the tab.

Example:
The following script creates an `ETabCtrl` object after creating an EForm object. The Tab Control will have six tabs, after the new one is added.

```
. . .
New EForm NewVar(gvForm) Top(100) Left(100) Width(400) Height(500);
New StringList NewVar(gvTabNameList) Value('Basic') Value('Default Font')
   Value('Pagination') Value('Numbering') Value('Advanced');
New ETabCtrl Form(gvForm) Top(50) Left(50) Width(300) Height(30)
      Strings(gvTabNameList) NewVar(gvTabCtrl);
Run gvTabCtrl.AddTab Value('Table Cell');
. . .
```

DeleteTab

Format:

```
Run gvTabCtrlVar.DeleteTab Number(memberNumber)
or
gvTabCtrlVar.DeleteTab{memberNumber};
```

The DeleteTab method deletes the specified string from the listbox..

Table 213: ETabCtrl DeleteTab Options

Option Name	Option Description
Number	The member number to delete.

Example:
The following script creates an `ETabCtrl` object after creating an EForm object. The Tab Control will have five tabs after the second member is deleted.

```
. . .
New EForm NewVar(gvForm) Top(100) Left(100) Width(400) Height(500);
New StringList NewVar(gvTabNameList) Value('Basic') Value('Default Font')
   Value('Pagination') Value('Numbering') Value('Advanced');
New ETabCtrl Form(gvForm) Top(50) Left(50) Width(300) Height(30)
       Strings(gvTabNameList) NewVar(gvTabCtrl);
Run gvTabCtrl.DeleteTab Number(2);
. . .
```

GetTabText

Format:

```
Run gvTabCtrlVar.GetTabText Number(memberNumber) NewVar(gvTabText);
or
Set gvTabText = gvTabCtrlVar.GetTabText{memberNumber};
```

The GetTabText method gets the string of the specified member number from the control.

Table 214: ETabCtrl GetTabText Options

Option Name	Option Description
Number	The member number of the tab.
NewVar	The string of the specified member number.

Example:
The following script creates an `ETabCtrl` object after creating an EForm object, then gets the text of the second tab in the list.

```
. . .
New EForm NewVar(gvForm) Top(100) Left(100) Width(400) Height(500);
New StringList NewVar(gvTabNameList) Value('Basic') Value('Default Font')
   Value('Pagination') Value('Numbering') Value('Advanced');
New ETabCtrl Form(gvForm) Top(50) Left(50) Width(300) Height(30)
       Strings(gvTabNameList) NewVar(gvTabCtrl);
Run gvTabCtrl.GetTab Number(2) NewVar(gvTabText);
. . .
```

SetTabText

Format:

```
Run gvTabCtrlVar.SetTabText Number(memNumber) Value(stringExpression)
or
Set memberNumber = gvTabCtrlVar.SetTabText{memNumber,stringExpression};
```

The SetTabText method sets the text of the specified tab member number.

Table 215: ETabCtrl SetTabText Options

Option Name	Option Description
Value	The text of the tab.
Number	The member number of the tab.

Example 1:
The following script creates an ETabCtrl object after creating an EForm object, then sets the text of the second member in the tab list.

```
. . .
New EForm NewVar(gvForm) Top(100) Left(100) Width(300) Height(500);
New StringList NewVar(gvTabNameList) Value('Basic') Value('Default Font')
   Value('Pagination') Value('Numbering') Value('Advanced');
New ETabCtrl Form(gvForm) Top(50) Left(50) Width(200) Height(200)
       Strings(gvTabNameList) NewVar(gvTabCtrl);
gvTabCtrl.SetTabText{'My Tab Value'};
. . .
```

ETreeCtrl Object

The ETreeCtrl object represents a list of hierarchically arranged items called nodes. Each node has a caption associated with and an optional checkbox. There are methods to add and delete these nodes. Each node is identified by an id number called a handle. The node handle is used to get and set information for a particular node. Nodes can optionally have child nodes.

A real world example of a tree control is the folder panel of the Windows Explorer program that comes with MS Windows. The TreeCtrlDemo.fsl demo script demonstrates many of the properties and methods of the ETreeCtrl control.

Creating the object

You create an instance of this object using the New command.

Format:

```
New ETreeCtrl Form(formObjectVar) [PropertyName(PropertyValue)] ...
   [NewVar(ctlVar)];
```

The New method creates a new ETreeCtrl object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 216: New ETreeCtrl Options

Option Name	Option Description
Form	Identifies the EForm object associated with this control. *(Required)*
PropertyName/PropertyValue	A set of property names and values from "Common Properties for Controls" on page 144 or "ETreeCtrl Properties" on page 183 or "Common Form Object Properties" on page 137
NewVar	The name of the variable to hold the newly created object.

Example:
The following script creates an `ETreeCtrl` object after creating an EForm object. See the TreeCtrlDemo.fsl script for a complete example.

```
. . .
New EForm NewVar(gvForm) Top(100) Left(100) Width(300) Height(500);
New ETreeCtrl Form(gvForm) Top(50) Left(50) Width(200) Height(200) NewVar(gvTreeVar);
. . .
```

ETreeCtrl Properties

Table 217: ETreeCtrl Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for ETreeCtrl objects.	
CancelOp	**Integer**	An indicator that you may set during some of the events for this control to cancel an ongoing function, such as node expanding or node changing.
Checkboxes	**Integer**	True to have checkboxes appear on the nodes, False otherwise. The default value is False. Note, once this is set to True it cannot be removed. You must destroy and re-create the control.
EditLabels	**Integer**	True to allow the user to edit the labels of the nodes, False otherwise. The default value is False.
FirstVisibleNode	**Handle**	The id of the first visible node in the control.
HasButtons	**Integer**	True to have a button to the left of each parent item, False otherwise. The default value is False.
HasLines	**Integer**	True to have graphical lines linking parent and child items, False otherwise. The default value is False.
Indent	**Integer**	The number of pixels to indent the nodes from the edge of the tree control.
LinesAtRoot	**Integer**	True to have graphical lines the root item. False otherwise. The default value is False.
NodeCount	**Integer**	The number of nodes in the control.
NodeHeight	**Integer**	The height of each node in pixels.
OnBeginEditLabel	**String**	The name of the subroutine to run when the node has expanded.
OnEndEditLabel	**String**	The name of the subroutine to run when the node is currently expanding. You can cancel the expansion, by setting the CancelOp property to True.
OnNodeExpanded	**String**	The name of the subroutine to run when the node has expanded.
OnNodeExpanding	**String**	The name of the subroutine to run when the node is currently expanding. You can cancel the expansion, by setting the CancelOp property to True.
OnSelChange	**String**	The name of the subroutine to run when a user changes the selected item in this control.
OnSelChanged	**String**	The name of the subroutine to run when the selection has changed.
OnSelChanging	**String**	The name of the subroutine to run when the selection is currently changing. You can cancel the change, by setting the CancelOp property to True.
RootNode	**Handle**	The id of the root node.
SelectedNode	**Handle**	The id of the currently selected node.
ShowSelAlways	**Integer**	True to keep the selected item selected when the control loses focus. False otherwise. The default value is False.

Table 217: ETreeCtrl Properties

Property Name	Data Typ;e	Property Description
SingleExpand	**Integer**	True to have the selected node automatically expand and have the previous node automatically collapse. False otherwise. The default value is False.
VisibleCount	**Integer**	The number of nodes in the control that are visible.

ETreeCtrl Methods

Table 218: ETreeCtrl Methods

Method Name	Method Description
DeleteAllNodes	Deletes all the nodes in the tree control.
InsertNode	Add a new node to the tree.
DeleteNode	Deletes a specified node from the tree.
EnsureNodeVisible	Run this method to ensure that the specified node is visible.
ExpandNode	Expands the specified node.
CollapseNode	Collapses the specified node.
CollapseAndResetNode	Collapses the specified node and removes the child nodes.
ToggleNode	Collapses the specified node, if it is expanded, or expand the node if it is collapsed.
SelectNode	Changes the selection to the specified node.
GetFirstChild	Gets the handle of the first child of the specified node.
GetNextSibling	Gets the handle of the next sibling node of the specified node.
GetPrevSibling	Gets the handle of the previous sibling node of the specified node.
GetParent	Gets the handle of the parent node of the specified node. This is 0 if the specified node has no parent.
GetNodeText	Gets the caption of the specified node.
SetNodeText	Sets the caption of the specified node.
GetNodeData	Gets the integer data associated with the specified node. Each node can have an optional integer data associated with it.
SetNodeData	Sets the integer data associated with the specified node. Each node can have an optional integer data associated with it.
IsChecked	Returns True if the specified node is checked, False otherwise. Applicable if CheckBoxes property is True.
SetCheck	Sets the check box. Applicable if CheckBoxes property is True.
HasChildren	Returns True if the specified node has children, False otherwise.

ETreeCtrl Method Descriptions

DeleteAllNodes

Format:

```
Run gvTreeVar.DeleteAllNodes;
or
gvTreeVar.DeleteAllNodes{};
```

The DeleteAllNodes method deletes all nodes from the control.

DeleteNode

Format:

```
Run gvTreeVar.DeleteNode Node(nodeHandle);
or
gvTreeVar.DeleteNode{nodeHandle);
```

The DeleteNode method deletes the specified node from the tree. All child nodes are also deleted.

Table 219: ETreeCtrl DeleteNode Options

Option Name	Option Description
`nodeHandle`	A node handle retrieved from a previous property or method.

Example:
The following script fragment deletes the current node from the tree control identified by the gvTreeVar variable.

```
. . .
Set lvCurrentNode = gvTreeVar.SelectedNode;
If lvCurrentNode
   gvTreeVar.DeleteNode{lvCurrentNode};
Else
   MsgBox 'There is no current node selected!';
EndIf
. . .
```

InsertNode

Format:

```
Run gvTreeVar.InsertNode Text(strExpr) [Parent(parentNodeHdl) [After(afterNodeHdl)]];
or
gvTreeVar.InsertNode{strExpr[,parentNodeHdl[,afterNodeHdl]]};
```

The InsertNode method creates and inserts the new node under the node identified by parentNodeHdl and after the child node specified by afterNodeHdl.

> **IMPORTANT:** The form on which this control is placed must be shown on the screen before you can insert nodes. The controls is not actually created until the form is displayed with the ShowModal or ShowModeless methods.

Table 220: ETreeCtrl InsertNode Options

Option Value	Option Description
strExpr	The text of the caption for the new node.
parentNodeHdl	The handle of the parent node for the new node. If this is omitted, then the new node will appear at the root level. You can also specified the string 'Root' to indicate the root level.
afterNodeHdl	The handle of the node after which the new node will be added. If this is omitted, then the new node will appear at the end of the child nodes for the parent. This can also be a string value indicating the position. The possible values are: 'First' - Make this new node the first child of the parent. 'Last' - Make this new node the last child of the parent (default). 'Sort' - Place the new node in sorted (ascending caption order) among the parent's children.

Example:

The following script creates an `ETreeCtrl` object after creating an EForm object. The tree will contain several names on several levels. The nodes are inserted in the Form Init event

```
. . .
New EForm NewVar(gvForm) Top(100) Left(100) Width(300) Height(500)
   OnInitForm('MyFormInit');;
New ETreeCtrl Form(gvForm) Top(50) Left(50) Width(200) Height(200)
   HasLines(gvHasLinesFlag) HasButtons(gvHasButtonsFlag)
   LinesAtRoot(gvLinesAtRootFlag) NewVar(gvTreeVar);
Run gvForm.ShowModal;
. . .
Event MyFormInit
   Local lvRootNode lvNode lvChild;
   Set lvRootNode = gvTreeVar.InsertNode{'The Shire'};
   Set lvNode = gvTreeVar.InsertNode{'Baggins', lvRootNode};
   Set lvChild = gvTreeVar.InsertNode{'Frodo', lvNode};
   Set lvChild = gvTreeVar.InsertNode{'Bilbo', lvNode};
   Set lvChild = gvTreeVar.InsertNode{'Drogo', lvNode,'First'};
EndEvent
```

IsChecked

Format:

```
Run gvTreeVar.IsChecked Node(nodeHandle) NewVar(checked);
or
Set checked = gvTreeVar.IsChecked{nodeHandle}
```

The IsChecked method returns the checked state, if the CheckBoxes property is set.

Table 221: ETreeCtrl IsChecked Options

Option Value	Option Description
nodeHandle	A node handle retrieved from a previous property or method.
checked	The return value, True if checked, False otherwise.

Example:
The following script fragment examines the current node to see if it is checked.

```
. . .
Set lvCurrentNode = gvTreeVar.SelectedNode;
If lvCurrentNode
    Set lvIsChecked = gvTreeVar.IsChecked{lvCurrentNode};
    MsgBox 'Checked state is '+lvIsChecked;
Else
    MsgBox 'There is no current node selected!';
EndIf
. . .
```

SetCheck

Format:

```
Run gvTreeVar.SetCheck Node(nodeHandle) Check(checkValue) NewVar(wasChecked);
or
Set wasChecked = gvTreeVar.SetChecked{nodeHandle[,checkValue]}
```

The SetChecked method sets the check mark for the specified node, if the CheckBoxes property is set.

Table 222: ETreeCtrl SetChecked Options

Option Value	Option Description
`nodeHandle`	A node handle retrieved from a previous property or method.
`checkValue`	True to set the check mark, False to clear the check mark.
`wasChecked`	The return value, True if checked was set, False if there was an error.

Example:
The following script fragment toggles the check mark for the selected node.

```
. . .
Set lvCurrentNode = gvTreeVar.SelectedNode;
If lvCurrentNode
    If gvTreeVar.IsChecked{lvCurrentNode};
        gvTreeVar.SetCheck{lvCurrentNode, False};
    Else
        gvTreeVar.SetCheck{lvCurrentNode, True};
    EndIf
Else
    MsgBox 'There is no current node selected!';
EndIf
. . .
```

HasChildren

Format:

```
Run gvTreeVar.HasChildren Node(nodeHandle) NewVar(gvHasChildrenFlag);
or
Set gvHasChildrenFlag = gvTreeVar.HasChildren{nodeHandle};
```

The HasChildren method returns True if the specified node has child nodes, False otherwise.

Table 223: ETreeCtrl HasChildren Options

Option Name	Option Description
nodeHandle	A node handle retrieved from a previous property or method.
gvHasChildrenFlag	The return value, True if the node has child nodes, False otherwise.

Example:
The following script fragment examines the current node to see if it has child nodes.

```
. . .
Set lvCurrentNode = gvTreeVar.SelectedNode;
If lvCurrentNode
   Set lvHasChildren = gvTreeVar.HasChildren{lvCurrentNode};
   If lvHasChildren
      MsgBox 'The current node has children';
   Else
      MsgBox 'The current does not have children';
   EndIf
Else
   MsgBox 'There is no current node selected!';
EndIf
. . .
```

GetNodeText

Format:

```
Run gvTreeVar.GetNodeText Node(nodeHandle) NewVar(gvNodeCaption);
or
Set gvNodeCaption = gvTreeVar.GetNodeText{nodeHandle};
```

The GetNodeText method returns the caption of the specified node.

Table 224: ETreeCtrl GetNodeText Options

Option Name	Option Description
nodeHandle	A node handle retrieved from a previous property or method.
gvNodeCaption	The return value: the text of the caption.

Example:
The following script fragment examines the displays the caption of the currently selected node.

```
. . .
Set lvCurrentNode = gvTreeVar.SelectedNode;
If lvCurrentNode
   Set lvCaption = gvTreeVar.GetNodeText{lvCurrentNode};
   MsgBox 'The current node caption is '+lvCaption;
Else
   MsgBox 'There is no current node selected!';
EndIf
. . .
```

SetNodeText

Format:

```
Run gvTreeVar.SetNodeText Node(nodeHandle) Text(gvNodeCaption);
or
gvTreeVar.SetNodeText{nodeHandle,gvNodeCaption};
```

The SetNodeText method sets the caption of the specified node.

Table 225: ETreeCtrl SetNodeText Options

Option Name	Option Description
`nodeHandle`	A node handle retrieved from a previous property or method.
`gvNodeCaption`	The new caption.

Example:
The following script fragment examines the sets the caption of the currently selected node.

```
. . .
Set lvCurrentNode = gvTreeVar.SelectedNode;
If lvCurrentNode
    Set lvRetCode = gvTreeVar.GetNodeText{lvCurrentNode,'My New Caption'};
Else
    MsgBox 'There is no current node selected!';
EndIf
. . .
```

GetNodeData

Format:

```
Run gvTreeVar.GetNodeData Node(nodeHandle) NewVar(gvNodeInt);
or
Set gvNodeInt = gvTreeVar.GetNodeData{nodeHandle};
```

The GetNodeData method returns the integer data associated with the specified node.

Table 226: ETreeCtrl GetNodeData Options

Option Name	Option Description
`nodeHandle`	A node handle retrieved from a previous property or method.
`gvNodeInt`	The return value: the integer data associated with the specified node.

Example:
The following script fragment examines the displays the data of the currently selected node.

```
. . .
Set lvCurrentNode = gvTreeVar.SelectedNode;
If lvCurrentNode
    Set lvInt = gvTreeVar.GetNodeData{lvCurrentNode};
    MsgBox 'The current node data is '+lvInt;
Else
    MsgBox 'There is no current node selected!';
EndIf
. . .
```

SetNodeData

Format:

```
Run gvTreeVar.SetNodeData Node(nodeHandle) Data(gvNodeInt);
or
Set gvRetCode = gvTreeVar.SetNodeData{nodeHandle,gvNodeInt};
```

The SetNodeData method sets the integer data associated with the specified node.

Table 227: ETreeCtrl SetNodeData Options

Option Name	Option Description
nodeHandle	A node handle retrieved from a previous property or method.
gvNodeInt	The new integer data to associate with the specified node.

Example:

The following script fragment examines the sets the data of the currently selected node to 999.

```
. . .
Set lvCurrentNode = gvTreeVar.SelectedNode;
If lvCurrentNode
   Set lvRetCode = gvTreeVar.SetNodeData{lvCurrentNode, 999};
Else
   MsgBox 'There is no current node selected!';
EndIf
. . .
```

GetFirstChild

Format:

```
Run gvTreeVar.GetFirstChild Node(nodeHandle) NewVar(gvChildNodeHdl);
or
Set gvChildNodeHdl = gvTreeVar.GetFirstChild{nodeHandle};
```

The GetFirstChild method returns the handle to the first child node of the specified node.

Table 228: ETreeCtrl GetFirstChild Options

Option Name	Option Description
nodeHandle	A node handle retrieved from a previous property or method.
gvChildNodeHdl	The handle to the first child node of the specified node.

Example:
The following script fragment examines the displays the caption of the first child of the currently selected node.

```
. . .
Set lvCurrentNode = gvTreeVar.SelectedNode;
If lvCurrentNode
   Set lvChild = gvTreeVar.GetFirstChild{lvCurrentNode};
   If lvChild
      MsgBox 'The caption of the first child is '+gvTreeVar.GetNodeText{lvChild};
   Else
      MsgBox 'The current node has no children';
   EndIf
Else
   MsgBox 'There is no current node selected!';
EndIf
. . .
```

GetNextSibling

Format:

```
Run gvTreeVar.GetNextSibling Node(nodeHandle) NewVar(gvNodeHdl);
or
Set gvNodeHdl = gvTreeVar.GetNextSibling{nodeHandle};
```

The GetNextSibling method returns the handle to the next sibling node of the specified node.

Table 229: ETreeCtrl GetNextSibling Options

Option Name	Option Description
`nodeHandle`	A node handle retrieved from a previous property or method.
`gvNodeHdl`	The handle to the next sibling node of the specified node.

GetPrevSibling

Format:

```
Run gvTreeVar.GetPrevSibling Node(nodeHandle) NewVar(gvNodeHdl);
or
Set gvNodeHdl = gvTreeVar.GetPrevSibling{nodeHandle};
```

The GetPrevSibling method returns the handle to the previous sibling node of the specified node.

Table 230: ETreeCtrl GetPrevSibling Options

Option Name	Option Description
`nodeHandle`	A node handle retrieved from a previous property or method.
`gvNodeHdl`	The handle to the previous sibling node of the specified node.

GetParent

Format:

```
Run gvTreeVar.GetParent Node(nodeHandle) NewVar(gvNodeHdl);
or
Set gvNodeHdl = gvTreeVar.GetParent{nodeHandle};
```

The GetParent method returns the handle to the parent node of the specified node.

Table 231: ETreeCtrl GetParent Options

Option Name	Option Description
nodeHandle	A node handle retrieved from a previous property or method.
gvNodeHdl	The handle to the parent node of the specified node.

ExpandNode

Format:

```
Run gvTreeVar.ExpandNode Node(nodeHandle);
or
Set gvRetCode = gvTreeVar.ExpandNode{nodeHandle};
```

The ExpandNode method expands the specified node.

Table 232: ETreeCtrl ExpandNode Options

Option Name	Option Description
nodeHandle	A node handle retrieved from a previous property or method.

CollapseNode

Format:

```
Run gvTreeVar.CollapseNode Node(nodeHandle);
or
Set gvRetCode = gvTreeVar.CollapseNode{nodeHandle};
```

The CollapseNode method collapses the specified node.

Table 233: ETreeCtrl CollapseNode Options

Option Name	Option Description
nodeHandle	A node handle retrieved from a previous property or method.

CollapseAndResetNode

Format:

```
Run gvTreeVar.CollapseAndResetNode Node(nodeHandle);
or
Set gvRetCode = gvTreeVar.CollapseAndResetNode{nodeHandle};
```

The CollapseAndResetNode method collapses the specified node and removes all of its children.

Table 234: ETreeCtrl CollapseAndResetNode Options

Option Name	Option Description
`nodeHandle`	A node handle retrieved from a previous property or method.

ToggleNode

Format:

```
Run gvTreeVar.ToggleNode Node(nodeHandle);
or
Set gvRetCode = gvTreeVar.ToggleNode{nodeHandle};
```

The ToggleNode method expands the specified node if it is collapsed or collapses it if it is expanded.

Table 235: ETreeCtrl ToggleNode Options

Option Name	Option Description
`nodeHandle`	A node handle retrieved from a previous property or method.

SelectNode

Format:

```
Run gvTreeVar.SelectNode Node(nodeHandle);
or
Set gvRetCode = gvTreeVar.SelectNode{nodeHandle};
```

The SelectNode method makes the specified node the currently selected node.

Table 236: ETreeCtrl SelectNode Options

Option Name	Option Description
`nodeHandle`	A node handle retrieved from a previous property or method.

EnsureNodeVisible

Format:

```
Run gvTreeVar.EnsureNodeVisible Node(nodeHandle);
or
Set gvRetCode = gvTreeVar.EnsureNodeVisible{nodeHandle};
```

The EnsureNodeVisible method scrolls the tree control (if necessaryt) to make the specified node visible in the control window.

Table 237: ETreeCtrl EnsureNodeVisible Options

Option Name	Option Description
`nodeHandle`	A node handle retrieved from a previous property or method.

EGridCtrl Object

The EGridCtrl object represents a rectangular set of rows and columns called cells. Each cell can have a variable length of text. There can be fixed rows and columns that do not scroll. Each row and column can be sized by the script or by the user. You can add and delete rows and columns. Rows and columns can be selected..

The GridCtrlDemo.fsl demo script demonstrates many of the properties and methods of the EGridCtrl control.

Creating the object

You create an instance of this object using the New command.

Format:

```
New EGridCtrl Form(formObjectVar) [PropertyName(PropertyValue)] ...
    [NewVar(ctlVar)];
```

The New method creates a new EGridCtrl object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 238: New EGridCtrl Options

Option Name	Option Description
`Form`	Identifies the EForm object associated with this control. *(Required)*
`PropertyName/PropertyValue`	A set of property names and values from "Common Properties for Controls" on page 144 or "EGridCtrl Properties" on page 195 or "Common Form Object Properties" on page 137
`NewVar`	The name of the variable to hold the newly created object.

Example:
The following script creates an `EGridCtrl` object after creating an EForm object. See the GridCtrlDemo.fsl script for a complete example.

```
. . .
New EForm NewVar(gvForm) Top(100) Left(100) Width(300) Height(500);
New EGridCtrl Form(gvForm) Top(50) Left(50) Width(200) Height(200) NewVar(gvGridVar);
. . .
```

EGridCtrl Properties

Table 239: EGridCtrl Properties

Property Name	Data Type	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for EGridCtrl objects.	
Sizing		
RowCount	**Integer**	The total number of rows in the grid.
ColCount	**Integer**	The total number of columns in the grid.
FixedRowCount	**Integer**	The number of fixed rows.
FixedColCount	**Integer**	The number of fixed columns.
DefCellHeight	**Integer**	The default cell height. This can only be set when the control is on the screen.
DefCellWidth	**Integer**	The default cell width. This can only be set when the control is on the screen.
Modes		
CellSelMode	**Integer**	True if the user can select the cells in the grid, False otherwise. The default value is True.
EditableMode	**Integer**	True if the user can edit the text in the cells, False otherwise. The default value is True.
ListMode	**Integer**	True to set the grid into List mode, False otherwise. The default value is False. When the grid is in list mode, full row selection is enabled and clicking on the column header will sort the grid by rows.
HeaderSortMode	**Integer**	If set to True, the rows are sorted on the column header click when ListMode is True.
FixedRowSelMode	**Integer**	If set to True, clicking on a fixed row selects the cell next to it. The default value is True.
FixedColSelMode	**Integer**	If set to True, clicking on a fixed column selects the cells underneath to it. The default value is True.
SingleRowSelMode	**Integer**	If set to True, only a single column can be selected at a time. This is only applicable if the ListMode is set to True. The default value is False.
SingleColSelMode	**Integer**	If set to True, only a single row can be selected at a time. The default value is False.
RowResizeMode	**Integer**	If set to True, rows can be resized. The default value is True.
ColResizeMode	**Integer**	If set to True, columns can be resized. The default value is True.
TitleTipsMode	**Integer**	If set to True, cell text is expanded when the mouse is passed over it. The default value is True.
TrackFocusCellMode	**Integer**	If set to True, the fixed cells on the same row/column as the current focus are highlighted with a sunken border. The default value is True.
FrameFocusCellMode	**Integer**	If set to True, the cell with the focus is highlighted with a framed border. The default value is True.
Styles		

Table 239: EGridCtrl Properties

Property Name	Data Type	Property Description
AutoSizeStyle	**String**	Sets how the autosizing should be performed. This should be one of the following values: 'Header' - Fixed cells only 'Data' - Non-fixed cells only 'Both' - Both kind of cells The default value is 'Both'.
GridLinesStyle	**String**	Sets the style of the grid lines. This should be one of the following values: 'None' - No Grid Lines 'Horz' - Only Horizontal grid lines 'Vert' - Only Vertical grid lines 'Both' - Both Horizontal and Vertical grid lines The default value is 'Both'.
Colors		
GridBackColor	**Integer**	The RGB background color of the grid. The color outside of the cells. The default value is RGB(128,128,128).
GridLineColor	**Integer**	The RGB color of the grid lines. Default value is RGB(192,192,192).
TitleTipBackColor	**Integer**	The RGB background color of the title tip text. The default is to use the background color.
TitleTipTextColor	**Integer**	The RGB foreground color of the title tip text. The default is to use the background color.
Position		
CurrRow	**Integer**	The Row number of the cell with the focus.
CurrCol	**Integer**	The column number of the cell with the focus.
Events		
CancelOp	**Integer**	An indicator that you may set during some of the events for this control to cancel an ongoing function, such as node expanding or node changing.
OnBeginEdit	**String**	The name of the event to run when the cell editing has started.
OnEndEdit	**String**	The name of the event to run when the cell editing has ended.
OnSelChange	**String**	The name of the event to run after a cell has been selected.
OnSelChanging	**String**	The name of the event to run just before a cell has been selected. You can cancel the selection, by setting the CancelOp property to True.
Miscellaneous		
SelectedCount	**Integer**	The number of cells selected.

EGridCtrl Methods

Table 240: EGridCtrl Methods

Method Name	Method Description
Cell Selection	
GetSelStartRow	Returns the starting row of the cell selection. `Set gvRow=gvGridCtrlVar.GetSelStartRow{};`
GetSelEndRow	Returns the ending row of the cell selection. `Set gvRow=gvGridCtrlVar.GetSelEndRow{};`
GetSelStartCol	Returns the starting column of the cell selection. `Set gvColumn=gvGridCtrlVar.GetSelStartCol{};`
GetSelEndCol	Returns the ending column of the cell selection. `Set gvColumn=gvGridCtrlVar.GetSelEndCol{};`
SetSel	Sets the cell selection range. `Set gvRet=gvGridCtrlVar.SetSel{startRow, startCol [,endRow, endCol]};` If the endRow and endCol or omitted, then it only selects the cell specified by startRow and startCol.
SetCurrCell	Sets the cell with the focus. `gvGridCtrlVar.SetCurrCell{row,col};`
Grid Operations	
Clear	Deletes all the cells and their content. `Set gvRet=gvGridCtrlVar.Clear{};`
ClearNonFixedRows	Deletes all the non-fixed rows and their content. `Set gvRet=gvGridCtrlVar.ClearNonFixedRows{};`
Refresh	Redraws the entire grid. `Set gvRet=gvGridCtrlVar.Refresh{};`
AutoSizeAll	Autosizes all the rows and columns. `Set gvRet=gvGridCtrlVar.AutoSizeAll{};`
AutoSizeRows	Autosizes all the rows. `Set gvRet=gvGridCtrlVar.AutoSizeRows{};`
AutoSizeColumns	Autosizes all the columns. `Set gvRet=gvGridCtrlVar.AutoSizeColumns{};`
ExpandAllToFit	Expands all the rows and columns to fit inside the grid. `Set gvRet=gvGridCtrlVar.ExpandAllToFit{};`
ExpandRowsToFit	Expands all the rows to fit inside the grid. `Set gvRet=gvGridCtrlVar.ExpandRowsToFit{};`
ExpandColumnsToFit	Expands all the columns to fit inside the grid. `Set gvRet=gvGridCtrlVar.ExpandColumnsToFit{};`
ExpandLastColToFit	Expands all the last columns to fit inside the grid. `Set gvRet=gvGridCtrlVar.ExpandLastColToFit{};`
Row Operations	
DeleteRow	Deletes a specified row. `Set gvRet=gvGridCtrlVar.DeleteRow{rowNumber};`

Table 240: EGridCtrl Methods

Method Name	Method Description
InsertRow	Inserts a new row. `Set gvNewRowNumber=gvGridCtrlVar.InsertRow{rowNumber[,text]};` This method will insert a new row before the `rowNumber` specified. If `rowNumber` is 0, it is inserted at the end. The optional text is the row heading text.
SelectRow	Selects the specified row. `Set gvRet=gvGridCtrlVar.SelectRow{rowNumber};`
GetRowHeight	Returns the height of the specified row. `Set gvRowHeight=gvGridCtrlVar.GetRowHeight{rowNumber};`
SetRowHeight	Sets the height of the specified row. `Set gvRet=gvGridCtrlVar.SetRowHeight{rowNumber,newHeight};`
AutoSizeRow	Autosizes the specified row. `Set gvRet=gvGridCtrlVar.AutoSizeRow{rowNumber[,scrollReset]};` If scrollReset is True then the scrollbars will be reset. The default value is True.
RedrawRow	Redraws the specified row. `Set gvRet=gvGridCtrlVar.RedrawRow{rowNumber};`
Column Operations	
DeleteCol	Deletes a specified column. `Set gvRet=gvGridCtrlVar.DeleteCol{ColumnNumber};`
InsertCol	Inserts a new column. `Set gvNewColNumber=gvGridCtrlVar.InsertCol{colNumber[,text]};` This method will insert a new column before the `colNumber` specified. If the `colNumber` is 0, it is inserted at the end. The optional text is the column heading text.
SelectCol	Selects the specified row. `Set gvRet=gvGridCtrlVar.SelectCol{colNumber};`
GetColWidth	Returns the width of the specified column. `Set gvColWidth=gvGridCtrlVar.GetColWidth{colNumber};`
SetColWidth	Sets the width of the specified column. `Set gvColWidth=gvGridCtrlVar.SetColWidth{colNumber,newWidth};`
RedrawCol	Redraws the specified column. `Set gvRet=gvGridCtrlVar.RedrawCol{colNumber};`
SortColumn	Sorts by the specified column. `Set gvRet=gvGridCtrlVar.SortColumn{colNumber[,asc]};` This method will sort the grid by the specified column. If asc is True, then the sort will be ascending, otherwise it will be descending. The default is ascending.

AutoSizeColumn	Autosizes the specified column.
	`Set gvRet =`
	`gvGridCtrlVar.AutoSizeColumn{colNumber[,style[,scrollReset]]};`
	The style (if specified) should be one of the following strings 'Header' 'Data' 'Both' 'Default'.
	If `scrollReset` is True then the scrollbars will be reset. The default value is True.

Cell Operations	
RedrawCell	Redraws the specified cell. `Set gvRet=gvGridCtrlVar.RedrawCell{rowNumber,colNumber};`

Table 240: EGridCtrl Methods

Method Name	Method Description
IsCellFixed	Returns True if the specified cell is a fixed cell. `Set gvRet=gvGridCtrlVar.IsCellFixed{rowNumber,colNumber};`
GetModifiedFlag	Returns True if the specified cell is has been modified. `Set gvRet=gvGridCtrlVar.GetModifiedFlag{rowNumber,colNumber};`
EnsureVisible	This method ensures that the specified cell is visible. `Set gvRet=gvGridCtrlVar.EnsureVisible{rowNumber,colNumber};`
IsCellValid	Returns True if the specified cell is valid. `Set gvRet=gvGridCtrlVar.IsCellValid{rowNumber,colNumber};`
IsCellVisible	Returns True if the specified cell is visible. `Set gvRet=gvGridCtrlVar.IsCellVisible{rowNumber,colNumber};`
IsCellEditable	Returns True if the specified cell can be edited. `Set gvRet=gvGridCtrlVar.IsCellEditable{rowNumber,colNumber};`
IsCellSelected	Returns True if the specified cell is selected. `Set gvRet=gvGridCtrlVar.IsCellSelected{rowNumber,colNumber};`
GetCellText	Returns the text in the specified cell. `Set gvText=gvGridCtrlVar.GetCellText{rowNumber,colNumber};`
SetCellText	Sets the text in the specified cell. `Set gvRet=gvGridCtrlVar.SetCellText{row,col,newText};`
GetCellData	Returns the integer data value associated with the cell. This value can be used as a pointer (or index) to a data structure. `Set gvData=gvGridCtrlVar.GetCellData{rowNumber,colNumber};`
SetCellData	Sets the integer data value associated with the cell. .This value can be used as a pointer (or index) to a data structure `Set gvRet=gvGridCtrlVar.SetCellData{row,col,intValue};`
GetCellFormat	Returns a StringList containing a set of cell format types. `Set gvList=gvGridCtrlVar.GetCellFormat{rowNumber,colNumber};` For the possible values, See "Draw Format Strings" on page 201.

SetCellFormat	Sets for Draw Format strings for the cell.
	`Set gvList=gvGridCtrlVar.SetCellFormat{rowNumber,`
	`colNumber[,fmtString][,fmtStringList],...};`
	The format strings are the same strings as in the `GetCellFormat` method (above). You can provide the strings as separate parameters or you can use a StringList.
	For the possible values, See "Draw Format Strings" on page 201.

Cell Colors	
GetCellBackColor	Returns the background color of the specified cell. `Set gvColor=gvGridCtrlVar.GetCellBackColor{row,col};`
SetCellBackColor	Sets the background color of the specified cell. `Set gvRet=gvGridCtrlVar.SetCellBackColor{row,col,color};`
GetCellTextColor	Returns the foreground color of the specified cell. `Set gvColor=gvGridCtrlVar.GetCellTextColor{row,col};`
SetCellTextColor	Sets the foreground color of the specified cell. `Set gvRet=gvGridCtrlVar.SetCellTextColor{row,col,color};`
Cell Font Info	

Table 240: EGridCtrl Methods

Method Name	Method Description
GetCellFontName	Returns the font family name of the specified cell. `Set gvFontName=gvGridCtrlVar.GetCellFontName{row,col};`
GetCellFontSize	Returns the font size of the specified cell. `Set gvSize=gvGridCtrlVar.GetCellFontSize{row,col};`
GetCellFontWeight	Returns the font weight of the specified cell. `Set gvWeight=gvGridCtrlVar.GetCellFontWeight{row,col};` The returned value should be 'Normal' or 'Bold'
GetCellFontAngle	Returns the font angle of the specified cell. `Set gvAngle=gvGridCtrlVar.GetCellFontAngle{row,col};` The returned value should be 'Normal' or 'Italic'
GetCellFontCharset	Returns the font charset of the specified cell. `Set gvCharset=gvGridCtrlVar.GetCellFontCharset{row,col};`
SetCellFontInfo	Sets the font information of the specified cell. See the description below.

EGridCtrl Method Descriptions

SetCellFontInfo

Format:

```
Run gvGridVar.SetCellFontInfo Row(rowNum) Col(colNum) [EndRow(endRowNum)]
    [EndCol(endColNum)] [FontName(name)] [FontSize(size)] [FontWeight(weight)]
    [FontAngle(angle)] [FontCharset(charset)] [DefaultCellFont(true)];
```

The SetCellFontInfo method sets the font information for a cell or a range of cells. You can select one or more of the five font properties. The others will remain as they were.

Table 241: EGridCtrl SetCellFontInfo Options

Option Name	Option Description
Row	The integer row number. Rows are numbered starting at 1.
Col	The integer column number. Columns are numbered starting at 1.
EndRow	The last row number. If not specified, it is the same as the Row option.
EndCol	The last column number. If not specified, it is the same as the Col option.
FontName	The name of the font family.
FontSize	The size of the font.
FontWeight	The weight of the font. Values are 'Normal' or 'Bold'.
FontAngle	The weight of the font. Values are 'Normal' or 'Italic'.

Table 241: EGridCtrl SetCellFontInfo Options

Option Name	Option Description
FontCharset	The name of the Character set for the FontName. Most of the time this should be omitted, because most of the time, the default character set is the correct one. Some fonts, such as Arial, though, have several character sets available. The following values are possible for the FontCharset value. Whether any are valid, depends on the FontName chosen: 'DEFAULT', 'SYMBOL', 'BALTIC', 'MAC', 'RUSSIAN', 'EASTEUROPE', 'THAI', 'VIETNAMESE', 'TURKISH', 'GREEK', 'ARABIC', 'HEBREW', 'JOHAB', 'OEM', 'CHINESEBIG5', 'GB2312', 'HANGUL', 'SHIFTJIS', 'HANGEUL', 'ANSI' 'DEFAULT' is the default value.
DefaultCellFont	True to use the values of the default cell font.

Table 242: Draw Format Strings

Option Name	Option Description
'Top'	Justifies the text to the top of the cell.
'Left'	Aligns text to the left.
'Center'	Centers text horizontally in the cell.
'Bottom'	Justifies the text to the bottom of the cell. This value is used only with the `SingleLine` format.
'SingleLine'	Displays the text as a single line. Carriage returns and line feeds do not break the line.
'Right'	Aligns text to the right.
'VCenter'	Centers text vertically in the cell. This value is used only with the `SingleLine` format
'ExternalLeading'	Includes the font external leading in the line height. Normally, external leading is not included in the height of the text.

'HidePrefix'	Ignores the ampersand (&) prefix character. The letter that follows will not be underlined, but other mnemonic prefix characters are still processed. for example:
	`Input string: "A&bc&&d"`
	`Normal: "Abc&d"`
	`HidePrefix: "Abc&d"`

'NoPrefix'	Turns off processing of prefix characters. Normally, the ampersand (&) character causes the following character to be underscored and the double ampersand (&&) will be a single ampersand. The NoPrefix option turns this off. For example:
	`Input string: "A&bc&&d"`
	`Normal: "Abc&d"`
	`NoPrefix: "A&bc&&d"`

'WordBreak'	Break words. Lines are automatically broken between words if a word would extend past the edge of the cell. A carriage return-line feed sequence also breaks the line.
'PathEllipsis'	Replaces characters in the middle of the string with ellipses so that the result fits in the cell. If the string contains backslash (\) characters, it preserves as much as possible of the text after the last backslash.
'WordEllipsis'	Truncates any word that does not fit in the cell and adds ellipsis.
'EndEllipsis'	If the end of the string does not fit in the cell, it is truncated and ellipses are added. If a word that is not at the end of the string goes beyond the limits of the cell, it is truncated without ellipses.

EScrollBar Object

The EScrollBar object represents a control acts as a scrollable rectangle. This is usually used for progress measures.

Creating the object

You create an instance of this object using the New command.

Format:

```
New EScrollBar Form(formObjectVar) [PropertyName(PropertyValue)] ...
    [NewVar(ctlVar)];
```

The New method creates a new EScrollBar object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 243: New EScrollBar Options

Option Name	Option Description
Form	Identifies the EForm object associated with this control. *(Required)*
PropertyName/PropertyValue	A set of property names and values from "Common Properties for Controls" on page 144 or "ELabel Properties" on page 152 or "Common Form Object Properties" on page 137
NewVar	The name of the variable to hold the newly created object.

Example:

The following script creates an `EScrollBar` object after creating an EForm object.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New EScrollBar Form(efrm) Top(50) Left(50) Width(200) Height(200)
       MinValue(0) MaxValue(100);
. . .
```

EScrollBar Properties

Table 244: EScrollBar Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for EScrollBar objects.	
CurrentValue	**Integer**	The current value (within the minvalue and maxvalue).
MinValue	**Integer**	The minimum value for the scrollbar.
MaxValue	**Integer**	The maximum value for the scrollbar.
PageIncr	**Integer**	The amount to scroll when a page up or page is done.

Table 244: EScrollBar Properties

Property Name	Data Typ;e	Property Description
`LineIncr`	`Integer`	The amount to scroll when a line up or page is done.
`Position`	`String`	Whether the scroll is vertical or horizontal. 'V' for Vertical and 'H' for horizontal. Default is 'H'.
`OnChange`	`String`	The name of the subroutine to run when a user changes the value (page up/down, line up/down).

EProgressBar Object

The EProgressBar object represents a bar on the screen that shows the progress of some operation.

Creating the object

You create an instance of this object using the New command.

Format:

```
New EProgressBar Form(formObjectVar) [PropertyName(PropertyValue)] ...
    [NewVar(ctlVar)];
```

The New method creates a new EProgressBar object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 245: New EProgressBar Options

Option Name	Option Description
`Form`	Identifies the EForm object associated with this control. *(Required)*
`PropertyName/PropertyValue`	A set of property names and values from "Common Properties for Controls" on page 144 or "EProgressBar Properties" on page 204 or "Common Form Object Properties" on page 137
`NewVar`	The name of the variable to hold the newly created object.

Example:

The following script creates an `EProgressBar` object after creating an EForm object.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New EProgressBar Form(efrm) Top(50) Left(50) Width(200) Height(200)
       MinValue(0) MaxValue(100);
. . .
```

EProgressBar Properties

Table 246: EProgressBar Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for EProgressBar objects.	
CurrentValue	**Integer**	The current value (within the minvalue and maxvalue).
MinValue	**Integer**	The minimum value for the scrollbar.
MaxValue	**Integer**	The maximum value for the scrollbar.

EDateTimeCtrl Object

The EDateTimeCtrl object allows the use a visual way of choosing a date and time.

Creating the object

You create an instance of this object using the New command.

Format:

```
New EDateTimeCtrl Form(formObjectVar) [PropertyName(PropertyValue)] ...
    [NewVar(ctlVar)];
```

The New method creates a new EDateTimeCtrl object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 247: New EDateTimeCtrl Options

Option Name	Option Description
Form	Identifies the EForm object associated with this control. *(Required)*
PropertyName/PropertyValue	A set of property names and values from "Common Properties for Controls" on page 144 or "EDateTimeCtrl Properties" on page 205 or "Common Form Object Properties" on page 137
NewVar	The name of the variable to hold the newly created object.

Example:

The following script creates an `EDateTimeCtrl` object after creating an EForm object. The date will have a long date format. See the DateTimeCtrlDemo.fsl script for a complete example.

```
. . .
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500);
New EDateCtrlCtrl Form(efrm) Top(50) Left(50) Width(150) Height(25)
       DisplayFormat('LongDateFormat');
. . .
```

EDateTimeCtrl Properties

Table 248: EDateTimeCtrl Properties

Property Name	Data Typ;e	Property Description
	In addition to the properties described in "Common Form Object Properties" on page 137 and those described in "Common Properties for Controls" on page 144, the following properties are available for EDateTimeCtrl objects.	
`UpDown`	`Integer`	True if the control has a pair of buttons allowing the user to increment or decrement the date, False means that there is a single arrow button the produces a calendar box when clicked. The default value is False.
`DisplayFormat`	`String`	The format to display the date-time when the calendar is not dropped down. The possible values are 'LongDateFormat' 'ShortDateFormat' or 'TimeFormat'.
`DateTime`	`EslObject`	The EDateTime object representing the date and time value in the control.
`OnDropDown`	`String`	The name of the event to run when a user causes the calendar to drop down. Default: None
`OnCloseUp`	`String`	The name of the event to run when a user causes the calendar to close. Default: None
`OnDateTimeChange`	`String`	The name of the event to run when the date-time changes in the control. Default: None

Menus Objects

Menu objects consists of EMenubar, EMenu, EMenuItem and EMenuSeparator objects.

Menu Events

When a menu event is called (such as OnClick) two parameters are passed to the event. The first is the form object variable (called EFormVar) and the second is the MenuItem Object variable (EMenuItemVar) of the control that caused the event. This allows you to use the same event subroutine to handle events in different forms and controls.

EMenubar Object

The EMenubar object represents a tree of menu objects, which may be assigned to a form.

Creating the object

You create an instance of this object using the New command.

Format:

```
New EMenubar [PropertyName(PropertyValue)] ...
    [NewVar(menubarVar)];
```

The New method creates a new EMenubar object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 249: New EMenubar Options

Option Name	Option Description
PropertyName/PropertyValue	A set of property names and values.
NewVar	The name of the variable to hold the newly created object.

Example:
The following script creates an `EMenubar` object and an EForm object, then assigns the menubar to the form..

```
. . .
New EMenubar NewVar(vMyMenuBarVar) Name('MyMenuBar');
New EForm NewVar(efrm) Top(100) Left(100) Width(300) Height(500)
       Menubar(vMyMenuBarVar);
. . .
```

EMenuBar Properties

Table 250: EMenubar Properties

Property Name	Data Type	Property Description
Name	**String**	The name of the menu.
ErrorCode	**Integer**	The ElmScript error code for the last method.
ErrorMsg	**String**	The Text of the last error.
FirstChildItem	**EslObject**	The first child menu object for this menu object. *(Read-Only)*
LastChildItem	**EslObject**	The last child menu object for this menu object. *(Read-Only)*
MenuCount	**Integer**	The number of child menus for this menu object.

EMenuBar Methods

Table 251: EMenubar Methods

Method Name	Method Description
Insert	Inserts an EMenu object into the EMenubar object.
Append	Appends an EMenu object onto the EMenubar object.
FindItem	Finds child menu object by name.

EMenuBar Method Descriptions

Insert

Format:

```
Run vMenuVar.Insert Before(menuObject)
or
vMenuVar.Insert{menuObject};
```

The Insert method inserts the menu object into an EMenubar object.

Table 252: EMenubar Insert Options

Option Name	Option Description
Before	Inserts this item before the specified item in the menu list
NewVar	The name of the variable to hold the newly created object.

. . .

Append

Format:

```
Run vMenuVar.Append AppendTo(menuObject)
or
vMenuVar.Append{menuObject};
```

The Append method inserts the menu object into an EMenubar object.

Table 253: EMenubar Append Options

Option Name	Option Description
AppendTo	Appends this item into the EMenubar at the end.
NewVar	The name of the variable to hold the newly created object.

. . .

FindItem

Format:

```
Run vMenuVar.FindItem Name(menuName) NewVar(menuObject)
or
Set menuObject = vMenuVar.FindItem{menuName};
```

The FindItem method search for a EMenu object in an EMenubar object.

Table 254: EMenubar FindItem Options

Option Name	Option Description
Name	The name of the menu to find.
NewVar	The name of the variable to hold the menu object. This will be NULL if not found.

```
. . .
```

EMenu Object

The EMenu object represents a drop down (popup) menu object.

Creating the object

You create an instance of these objects using the New command.

Format:

```
New EMenu [PropertyName(PropertyValue)] ...
    [NewVar(menuVar)];
```

The New method creates a new EMenu object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 255: New EMenu Options

Option Name	Option Description
PropertyName/PropertyValue	A set of property names and values.
NewVar	The name of the variable to hold the newly created object.

Example:

The following script creates an `EMenu` object and adds it to a menu bar.

```
. . .
New EMenubar NewVar(vMyMenuBarVar) Name('MyMenuBar');
New EMenu NewVar(vMyFileMenu) Name('MyFile') Parent(vMyMenuBarVar);
. . .
```

EMenu Properties

Table 256: EMenu Properties

Property Name	Data Type	Property Description
Name	**String**	The name of the menu.
Caption	**String**	The Label text that appears in the menu.

Table 256: EMenu Properties

Property Name	Data Type	Property Description
Enabled	**Integer**	True if the menu object is currently enabled(Default); False otherwise.
Checked	**Integer**	True if the menu object is currently checked; False (Default) otherwise..
Parent	**EslObject**	The parent object (EMenu or EMenubar).
ErrorCode	**Integer**	The ElmScript error code for the last method.
ErrorMsg	**String**	The Text of the last error.
NextItemInMenu	**EslObject**	The next menu object in the parent menu object. *(Read-Only)*
PrevItemInMenu	**EslObject**	The previous menu object in the parent menu object.. *(Read-Only)*
FirstChildItem	**EslObject**	The first child menu object for this menu object. *(Read-Only)*
LastChildItem	**EslObject**	The last child menu object for this menu object. *(Read-Only)*
MenuCount	**Integer**	The number of child menus for this menu object.

EMenu Methods

Table 257: EMenu Methods

Method Name	Method Description
Insert	Inserts an EMenu object into an EMenu object.
Append	Appends an EMenu object onto the EMenu object.
Remove	Removes a menu object from an EMenu object.
FindItem	Finds child menu object by name.
Popup	Popups up a menu on a form.

EMenu Method Descriptions

Insert

Format:

```
Run vMenuVar.Insert [Before(menuObject)] or {AppendTo(menuObject)]
```

The Insert method inserts the menu object into another EMenu object.

Table 258: EMenu Insert Options

Option Name	Option Description
Before	Inserts this item before the specified item in the menu list
NewVar	The name of the variable to hold the newly created object.

Example:
The following script creates an `EMenubar` object and two EMenu objects. One EMenu object is added to the menubar on creation; the other is added to the first EMenu object at the end.

```
. . .
New EMenubar NewVar(vMyMenuBarVar) Name('MyMenuBar');
New EMenu NewVar(vMyFileMenu) Name('MyFileMenu') Parent(vMyMenuBarVar)
       Label('File');
...
New EMenu NewVar(vMySubMenu) Name('MySubMenu') Label('Sub menu Off file menu');
...
RUN vMySubMenu.Insert Before(vMyFileMenu);
. . .
```

Append

Format:

```
Run vMenuVar.Append AppendTo(menuObject)
or
vMenuVar.Append{menuObject};
```

The Append method appends the menu object onto an EMenu object.

Table 259: EMenu Append Options

Option Name	Option Description
AppendTo	Appends this item into the EMenu at the end.
NewVar	The name of the variable to hold the newly created object.

```
. . .
```

Find

Format:

```
Run vMenuVar.Find Name(menuName) NewVar(menuObject)
or
Set menuObject = vMenuVar.Find{menuName};
```

The FindItem method search for a EMenuItem or EMenu object in an EMenu object.

Table 260: EMenu FindItem Options

Option Name	Option Description
Name	The name of the menu to find.
NewVar	The name of the variable to hold the menu object. This will be NULL if not found.

. . .

Remove

Format:

```
Run vMenuVar.Remove
or
vMenuVar.Remove{};
```

The Remove method removes the menu from its parent.

PopUp

Format:

```
Run vMenuVar.PopUp Form(parentFormName)
or
vMenuVar.PopUp{parentFormName};
```

The PopUp method creates and displays a popup menu independent of the menubar. This is typically used when the user right clicks the mouse on a form.

Table 261: EMenu PopUp Options

Option Name	Option Description
Form	The Form Object the acts as the parent of the Popup menu.

. . .

EMenuItem Object

The EMenuItem object represents a command menu item.

Creating the object

You create an instance of these objects using the New command.

Format:

```
New EMenuItem [PropertyName(PropertyValue)] ...
    [NewVar(menubarVar)];
```

The New method creates a new EMenuItem object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 262: New EMenuItem Options

Option Name	Option Description
PropertyName/PropertyValue	A set of property names and values.
NewVar	The name of the variable to hold the newly created object.

Example:
The following script creates an `EMenubar` object, an EMenu object and an EMenuItem object and illustrates the use of an event.

```
. . .
New EMenubar NewVar(vMyMenuBarVar) Name('MyMenuBar');
New EMenu NewVar(vMyFileMenu) Name('MyFile') Parent(vMyMenuBarVar)
       Caption('File');
New EMenuItem NewVar(vMyOpenMenuItem) Name('MyOpenMenuItem')
       Caption('Open My Kind of File') OnClick('MyFileOpen');
...
...
SUB MyFileOpen

       // Open My Kind of file

ENDSUB
. . .
```

EMenuItem Properties

Table 263: EMenuItem Properties

Property Name	Data Type	Property Description
Name	**String**	The name of the menuitem.
Caption	**String**	The Label text that appears in the menuitem.
Shortcut	**String**	The keyboard shortcut for this menu item command. Only the control keys with letters are available for shortcuts. If used, this string should be in the form 'Ctl-L', where L is the letter you wish to use. By default, there is no shortcut.
Enabled	**Integer**	True if the menuitem object is currently enabled(Default); False otherwise.
Checked	**Integer**	True if the menuitem object is currently checked; False (Default) otherwise..
Parent	**EslObject**	The parent object (EMenu or EMenubar).
ErrorCode	**Integer**	The ElmScript error code for the last method.
ErrorMsg	**String**	The Text of the last error.
NextItemInMenu	**EslObject**	The next menu object in the parent menu object. *(Read-Only)*

Table 263: EMenuItem Properties

Property Name	Data Type	Property Description
PrevItemInMenu	**EslObject**	The previous menu object in the parent menu object.. *(Read-Only)*
OnClick	**String**	The name of the event subroutine to run when someone selects this menu item.

EMenuItem Methods

Table 264: EMenuItem Methods

Method Name	Method Description
Insert	Inserts an EMenuItem object into an EMenu object.
Append	Adds an EMenuItem object into an EMenu object. at the end.
Remove	Removes a EMenuItem object from an EMenu object.

EMenuItem Method Descriptions

Insert

Format:

```
Run vMenuItemVar.Insert Before(menuObject)
```

The Insert method inserts the menu object into another EMenu or EMenubar object.

Table 265: EMenuItem Insert Options

Option Name	Option Description
Before	Inserts this item before the specified item in the menu list
NewVar	The name of the variable to hold the newly created object.

Example:

The following script creates an `EMenubar` object and an EForm object, then assigns the menubar to the form..

```
. . .
New EMenubar NewVar(vMyMenuBarVar) Name('MyMenuBar');
New EMenu NewVar(vMyFileMenu) Name('MyFileMenu') Parent(vMyMenuBarVar)
       Label('File');
...
New EMenuItem NewVar(vMySubMenuItem1) Name('MySubMenuItem1')
          Label('My First file Menu Item') Parent(vMyFileMenu);
...
New EMenuItem NewVar(vMySubMenuItem2) Name('MySubMenuItem2')
          Label('My Prev file Menu Item');
...
Run vMySubMenuItem.Insert Before(vMySubMenuItem1);
. . .
```

Append

Format:

```
Run vMenuItemVar.Append AppendTo(menuObject)
```

The Append method appends the menu object onto an EMenu object.

Table 266: EMenuItem Append Options

Option Name	Option Description
AppendTo	Append this item into the specified item in the menu list at the end.
NewVar	The name of the variable to hold the newly created object.

Example:

The following script creates an `EMenubar` object and an EForm object, then assigns the menubar to the form..

```
. . .
New EMenubar NewVar(vMyMenuBarVar) Name('MyMenuBar');
New EMenu NewVar(vMyFileMenu) Name('MyFileMenu') Parent(vMyMenuBarVar)
      Label('File');
...
New EMenuItem NewVar(vMySubMenuItem) Name('MySubMenuItem')
          Label('My file Menu Item');
...
Run vMySubMenuItem.Append AppendTo(vMyFileMenu);
. . .
```

Remove

Format:

```
RUN vMenuItemVar.Remove
or
vMenuItemVar.Remove{};
```

The Remove method removes the menuitem from its parent.

EMenuSeparator Object

The EMenuSeparator object represents a separator line on a menu

Creating the object

You create an instance of these objects using the New command.

Format:

```
New EMenuSeparator [PropertyName(PropertyValue)] ...
    [NewVar(menubarVar)];
```

The New method creates a new EMenuSeparatorItem object. Most of the parameters are optional. You may set them via properties if you don't specify them on the New command.

Table 267: New EMenuSeparator Options

Option Name	Option Description
PropertyName/PropertyValue	A set of property names and values.
NewVar	The name of the variable to hold the newly created object.

Example:
The following script creates an `EMenubar` object, an EMenu object, an EMenuItem object and an EMenuSeparator object and illustrates the use of an event.

```
. . .
New EMenubar NewVar(vMyMenuBarVar) Name('MyMenuBar');
New EMenu NewVar(vMyFileMenu) Name('MyFile') Parent(vMyMenuBarVar)
       Caption('File');
New EMenuItem NewVar(vMyOpenMenuItem)
       Name('MyOpenMenuItem') Parent(vMyFileMenu)
       Caption('Open My Kind of File') OnClick('MyFileOpen');
New EMenuSeparator Parent(vMyFileMenu);
...
...
SUB MyFileOpen

       // Open My Kind of file

ENDSUB
. . .
```

EMenuSeparator Properties

Table 268: EMenuSeparator Properties

Property Name	Data Type	Property Description
Name	**String**	The name of the menuitem.
Enabled	**Integer**	True if the menuitem object is currently enabled(Default); False otherwise.
Parent	**EslObject**	The parent object (EMenu or EMenubar).
ErrorCode	**Integer**	The ElmScript error code for the last method.
ErrorMsg	**String**	The Text of the last error.
NextItemInMenu	**EslObject**	The next menu object in the parent menu object. *(Read-Only)*
PrevItemInMenu	**EslObject**	The previous menu object in the parent menu object.. *(Read-Only)*

EMenuSeparator Methods

Table 269: EMenuSeparator Methods

Method Name	Method Description
Insert	Inserts an EMenuItem object into an EMenu object.
Append	Adds an EMenuItem object into an EMenu object. at the end.
Remove	Removes a EMenuItem object from an EMenu object.

EMenuSeparator Method Descriptions

Insert

Format:

```
Run vMenuSepItemVar.Insert Before(menuObject)
```

The Insert method inserts the menu object into another EMenu or EMenubar object.

Table 270: EMenuItem Insert Options

Option Name	Option Description
Before	Inserts this item before the specified item in the menu list
NewVar	The name of the variable to hold the newly created object.

Example:

The following script creates an `EMenubar` object and an EForm object, then assigns the menubar to the form..

```
. . .
New EMenubar NewVar(vMyMenuBarVar) Name('MyMenuBar');
New EMenu NewVar(vMyFileMenu) Name('MyFileMenu') Parent(vMyMenuBarVar)
       Label('File');
...
New EMenuItem NewVar(vMySubMenuItem) Name('MySubMenuItem')
          Label('My First file Menu Item') Parent(vMyFileMenu);
...
New EMenuSeparator NewVar(vMySubMenuSep) Name('MySep');
...
Run vMySubMenuSep.Insert Before(vMySubMenuItem);
. . .
```

Append

Format:

```
Run vMenuSepItemVar.Append AppendTo(menuObject)
```

The Append method appends the menu object onto an EMenu object.

Table 271: EMenuSeparator Append Options

Option Name	Option Description
AppendTo	Append this item into the specified item in the menu list at the end.
NewVar	The name of the variable to hold the newly created object.

Example:

The following script creates an `EMenubar` object and an EForm object, then assigns the menubar to the form..

```
. . .
New EMenubar NewVar(vMyMenuBarVar) Name('MyMenuBar');
New EMenu NewVar(vMyFileMenu) Name('MyFileMenu') Parent(vMyMenuBarVar)
       Label('File');
...
New EMenuItem NewVar(vMySubMenuItem) Name('MySubMenuItem')
          Label('My file Menu Item');
...
New EMenuSeparator NewVar(vMySubMenuSep) Name('MySep');
...
Run vMySubMenuSep.Append AppendTo(vMyFileMenu);
. . .
```

Remove

Format:

```
Run vMenuSepItemVar.Remove
or
vMenuSepItemVar.Remove{};
```

The Remove method removes the menuSeparator from its parent.

Chapter 14

Forms Tutorial

Introduction

The following is quick tutorial to introduce Scripters to the functionality of the ElmScript Forms system (EForms). These examples were chosen more to illustrate the functionality than to provide useful scripts. The imagination of the community of Scripters will be able to provide many more creatively useful (usefully creative?) examples than we could devise.

> **IMPORTANT:** For this tutorial, you will be asked to type in script commands and run the scripts. You can use the ElmScript text editor or use your favorite editor for this. If you use the ElmScript editor, you can run the scripts directly from the editor (using the Run button or Run menu selection). If you use your own editor, you will have to save the script to a disk file, then use the ElmScript->Run command and select that file to run the script or use the utility functions in TextPad or UltraEdit.

Hello World Example

Traditionally the first program one writes in a new language is the simplest program possible just to get something working. Usually this program/script prints or display the phrase 'Hello World' to the user. Of course, in ElmScript, you don't need forms to do this. You can use the MsgBox command. It is a one line script as follows:

```
MsgBox 'Hello World';
```

This is perhaps the shortest 'Hello World' program/script of all time. Most scripts/programs require that you have, at least, some structural text around the code, not ElmScript. In any case, we are also going to do a 'Hello World' type script using EForms, just to get a simple form working.

Exercise 1. **Simple 'Hello World' Example**

Step: 1. Enter the following two lines into a text editor file.

```
New EForm NewVar(ef) Caption('Hello World');
Run ef.ShowModal;
```

Using EForms, the 'Hello World' script needs two commands instead of just one. The first line creates the form and puts the object handle into the variable called ef. The second line shows the form in a Modal fashion (See introduction for more information on Modal and Modeless display types).

1 Run the script.

A form should appear with the words 'Hello World' in the title bar. The rest of it should be blank.

2 Click on the CloseBox of the form to dismiss it.

There you have it, a simple EForm script that works. Of course, it doesn't do very much. In fact, with all that empty space on the form, it doesn't even look that good. Let's try to fix it up a little bit and so show some of the power of EForms.

Exercise 2. Improved 'Hello World' Example

Step: 1. You can use the last example as a starting point and add the line indicated.

```
New EForm NewVar(ef) Caption('Hello World');
New ELabel Form(ef) Caption('Hello World');
Run ef.ShowModal;
```

This will create an ELabel control which displays static text. The static text is specified by the Caption option, which is, what else, 'Hello World'.

3 Run the script.

A form should appear with the words 'Hello World' in the title bar and some static text on the main area of the form, positioned perfectly in the upper left hand corner.

4 Click on the CloseBox of the form to dismiss it.

That was certainly an improvement, especially if you have exceptionally low standards. As with the rest of ElmScript, EForms uses default values when you do not specify them. Since we did not specify where to place this ELabel control, the default placement put it in the upper left hand corner. This may be what you want on rare occasions, but most of the time, you need to specify a location for a control. See "Panels, Placement and Sizing" on page 135. for more information on control placement.

Exercise 3. New and Improved 'Hello World' Example

For this new and improved version, we will try to do some placement examples, placing 'Hello World' at various places on the form. To place a control, you need to specify the coordinates for the top left corner of the control.

Step: 1. You can use the last example as a starting point and add the lines indicated.

```
New EForm NewVar(ef) Caption('Hello World');
New ELabel Form(ef) Caption('Hello World');
New ELabel Form(ef) Caption('Hello World-2') Top(200);
New ELabel Form(ef) Caption('Hello World-3') Left(200);
New ELabel Form(ef) Caption('Hello World-4') Top(200) Left(200);
New ELabel Form(ef) Caption('Hello World-5') Top(200) LeftLocFromRight(100);
New ELabel Form(ef) Caption('Hello World-6') Left(200) TopLocFromBottom(100);
New ELabel Form(ef) Caption('Hello World-7') TopLocPercent(0.6) LeftLocPercent(.4);
Run ef.ShowModal;
```

This illustrates several ways of placing controls on a form. The **Top** and **Left** options indicate positions relative to the upper left corner of the form. The default values (as shown by the last exercise) are 0 and 0. You can also use the **LeftLocFromRight** option to specify a location from the left side of the form and the **TopLocFromBottom** option to specify a location from the bottom of the form. Also, the **TopLocPercent** and **LeftLocPercent** can be used to specify a location as a percentage of the form size. The value must be between 0 and 1.

5 Run the script.

A form should appear with the 'Hello World' in various places on the form. You can look at the caption suffix to see where each string was placed. For fixed size forms, you will most likely use the **Top** and **Left** options to specify the location.

6 Click on the CloseBox of the form to dismiss it.

Exercise 4. New and Slightly Improved 'Hello World' Example

This is just a small change from the previous example. Add the Resizeable option to the New Form command to make a form that the user can adjust the size. Keep the rest the same as the last example.

Step: 1. Make the indicated change.

```
New EForm NewVar(ef) Caption('Hello World') CanResize(True);
New ELabel Form(ef) Caption('Hello World');
New ELabel Form(ef) Caption('Hello World-2') Top(200);
New ELabel Form(ef) Caption('Hello World-3') Left(200);
New ELabel Form(ef) Caption('Hello World-4') Top(200) Left(200);
New ELabel Form(ef) Caption('Hello World-5') Top(200) LeftLocFromRight(100);
New ELabel Form(ef) Caption('Hello World-6') Left(200) TopLocFromBottom(100);
New ELabel Form(ef) Caption('Hello World-7') TopLocPercent(0.6) LeftLocPercent(.4);
Run ef.ShowModal;
```

7 Run the script.

The same form should appear as in the last example, except that you can now change the size of the form by using the mouse.

8 Change the size of the form using the mouse and see what happens to the placement of the ELabel controls.

The controls that were placed with fixed sizes (Top and Left) should stay in the same place, while the ones placed with the other options should move as the form changes size.

9 Click on the CloseBox of the form to dismiss it.

For forms that can change size, using the LeftLocFromRight, TopLocFromBottom, etc. can help insure that your form will keep its look and feel.

Exercise 5. Fixed size example

You don't have to worry about the size of an ELabel control. The size of the control is determined by the size of the text. If not specified, ElmScript attempts to determine the size of any control by its contents. This works well for ELabel, EButton, ECheckbox, and ERadiobutton controls because they have text associated with them. Other controls, such as EEdit and EListbox, do not necessarily have a fixed text associated with them. They can optionally start with some text but the user may add text to them and they should not change size when that happens.

For this next exercise, we'll go back to our original script and add an EEdit control.

Step: 1. Start with our original EForm script and add an EEdit control as follows:

```
New EForm NewVar(ef) Caption('Hello World') CanResize;
New ELabel Form(ef) Caption('Hello World') Top(30) Left(50);
New EEdit Form(ef) Text('Hello World') Top(50) Left(50) Width(200);
Run ef.ShowModal;
```

10 Run the script.

The form should appear with a label control and an edit control. These are fixed position and fixed size controls.

11 Change the size of the form using the mouse.

You will notice that the controls remain in the same place and keep the same size no matter how you change the size of the form.

12 Click on the CloseBox of the form to dismiss it.

Exercise 6. Variable size example

There are times when you might want a control's size and position to change when the form changes size.

Step: 1. Start with the last exercise and make the indicated changes:

```
New EForm NewVar(ef) Caption('Hello World') CanResize;
New ELabel Form(ef) Caption('Hello World') TopLocFromBottom(50) Left(50);
New EEdit Form(ef) Text('Hello World') TopLocFromBottom(30) Left(50) WidthMinus(70);
Run ef.ShowModal;
```

13 Run the script.

The form should appear with a label control and an edit control near the bottom of the form.

14 Change the size of the form using the mouse.

You will notice that the controls move as the form gets shorter and longer. They stay at the bottom of the form. Also note that the EEdit control changes width as the form gets narrower and wider.

15 Click on the CloseBox of the form to dismiss it.

Exercise 7. Changing Font and Font size

Our original 'Hello World' form with a label (See "Improved 'Hello World' Example" on page 220.) example worked but it looked rather unimaginative. The text was very small and in the top left corner. For our last exercise in this category, we will fix this up to make it a nicer looking 'Hello World' form.

Step: 1. Start with the first exercise that had a ELabel control and add the lines indicated.

```
New EForm NewVar(ef) Caption('Hello World') CanResize;
New ELabel Form(ef) Caption('Hello World')
   TopLocPercent(.4) LeftLocPercent(.3) FontName('Arial')
   FontSize(48) FontWeight('Bold') FontAngle('Italic');
Run ef.ShowModal;
```

16 Run the script.

This will create our ’Hello World’ ELabel control with extra font information to make the text larger, etc..

17 .Change the size of the form using the mouse.

The ELabel control moves as the form changes size. We finally have a "Hello World’ form script that is worthy to be called an EForm ’Hello World’ script.

18 Click on the CloseBox of the form to dismiss it.

Input Dialog Box Example

One of the most common uses of forms is to get and present data to the script user. This is usually accomplished with fixed size forms and fixed size controls. As in the previous example, these will be Modal forms (or Modal dialog boxes). This means that the form is presented to the user and the user must respond to the form before continuing other operations. FrameMaker has many examples of Modal dialogs, including the Variable Dialog, the New Anchored Frame dialog and the Import File dialog.

Exercise 8. A Simple Input Form

Step: 1. Enter the following lines into a script file. If you do not want to enter the text manually, look at the enclosed script named ’InputDialog1.fsl’.

```
New EForm NewVar(ef) Caption('Employment Application')
       Width(400) Height(140);
New ELabel Form(ef) Caption('Name:') Top(10) Left(10);
New EEdit  Form(ef) Top(10) Left(50) Width(330) NewVar(vNameVar);
New EButton Form(ef) TopLocFromBottom(40) LeftLocPercent(.25) Caption('OK')
       CloseFormWithValue(OKButton);
New EButton Form(ef) TopLocFromBottom(40) LeftLocPercent(.66) Caption('Cancel')
       CloseFormWithValue(CancelButton);
Run ef.ShowModal NewVar(result);
If result = CANCELBUTTON
   Display 'Cancel Button Pressed';
Else
   If result = OKBUTTON
      Display 'OK Button Pressed';
      Write Console 'Name Entered-'+vNameVar.Text;
   Else
      Display 'Unknown Form Term.';
   EndIf
EndIf
```

1 Run the script.

At this point, since this is a tutorial, I’ll stop and make a few points. Most modal forms (dialog boxes) have at least two button controls, usually, at least in English, labeled OK and Cancel. The OK button tells the script to process the information on the form, while the Cancel button tells the script to ignore the information on the form and skip any processing. This is so commonplace that to change it would cause confusion in any user trying to run your script. Of course, the caption does not have to be ’OK’ or ’Cancel’, as long as it is obvious to the user what will happen when the buttons are clicked. Also, when a user clicks on one of these buttons, it is common practice for the form to go away

to do the processing (or to skip the processing). If you want the script to process the form information, yet stay on the screen, the common practice is to include a third button, usually labeled 'Apply'.

Back to the exercise:

A form should appear with a label, an edit control and two buttons, OK and Cancel. The edit control allows you to type in some text, in this case, an person's name. Note that although this is a fixed size form, I placed the two buttons, relative to the bottom of the form and as a percentage of the forms width. I could have used coordinates relative to the top just as well, but I know that I am going to add more controls later. I know that I want these two buttons to be at the bottom of the form no matter what size I eventually make it. Also note another option that I used when I created the EButton controls, is the CloseFormWithValue option. For buttons controls using this option causes two things to happen when the button is clicked. The form will close (go away) and the value in this option will be returned as the result of the form. In this case, the OK button uses the standard constant value OKButton in the CloseFormWithValue option, so that is the value that returns when the RUN ef.ShowModal command in run.

2 Type some text into the edit field and click on the OK button..

The form should go away and a message should appear telling you that you pressed the OK button and the text that you entered should appear on the FrameMaker console.

Exercise 9. A Slightly More Complex Input Form

Let's add some more controls to make it look like a more typical dialog box. With apologies to the Dilbert comic strip, do the following steps.

Step: 1. Enter the following lines into a script file. Remember to change the height and width of the form. If you do not want to enter the text manually, look at the enclosed script named 'InputDialog2.fsl'.

```
New EForm NewVar(ef) Caption('Employment Application')
       Width(440) Height(280);
New ELabel Form(ef) Caption('Name:') Top(10) Left(10);
New EEdit  Form(ef) Top(10) Left(50) Width(370) NewVar(vNameVar);
New EButton Form(ef) TopLocFromBottom(40) LeftLocPercent(.25) Caption('OK')
       CloseFormWithValue(OKButton);
New EButton Form(ef) TopLocFromBottom(40) LeftLocPercent(.66) Caption('Cancel')
       CloseFormWithValue(CancelButton);
New EGroupBox Form(ef) Caption('Gender')
   TOP(35) LEFT(10) Width(120) Height(70);
New ERadioButton Form(ef) Top(50) Left(20) Caption('Male') Group(1)
          IsChecked(True) NewVar(vMaleBtn);
New ERadioButton Form(ef) Top(70) Left(20) Caption('Female') Group(1);
New ELabel Form(ef) Caption('Type of Job Desired') Top(40) Left(160);
New EDropDownBox Form(ef) Top(55) Left(160) Width(240)
          LineCount(5) Height(100) NewVar(vDDBox);
New ELabel Form(ef) Caption('References') Top(95) Left(160);
New EEdit Form(ef) Top(110) Left(160) Width(260) NewVar(vRefVar);
New StringList NewVar(vJobTypeList) Value('Dilbert Type Trainee')
                                    Value('Dilbert Type Engineer')
                                    Value('Pointy Hair Boss');
Set vDDBox.Strings = vJobTypeList;
Set vDDBox.SelIndex = 1;
New ECheckBox Form(ef) Top(120) Left(10)
       Caption('Do you want a job?') NewVar(vCbxBox);
Run ef.ShowModal NewVar(result);
If result = CANCELBUTTON
   Display 'Cancel Button Pressed';
Else
   If result = OKBUTTON
      Display 'OK Button Pressed';
      Write Console 'Name Entered-'+vNameVar.Text;
      If vMaleBtn.IsChecked
         Write console 'Gender is Male';
      Else
         Write Console 'Gender is Female';
      EndIf
      IF vCbxBox.IsChecked
         Write Console 'Wants a Job';
      ELSE
         Write Console 'Does Not Want a Job';
      EndIf
      Write Console 'Sel Index-'+vDDBox.SelIndex;
      Write Console 'Job Type-'+vJobTypeList[vDDBox.SelIndex];
      Write Console 'References-'+vRefVar.Text;
   Else
      Display 'Unknown Form Term.';
   EndIf
EndIf
```

We've added a CheckBox control, used for simple On/Off (True/False, Yes/No) types of information, two Radiobuttons, used to allow the user to select from among a small number of fixed choices, a drop down box, used to let the user select from among a larger number of choices and another edit control.

3 Run the script.

This shows a variety of controls. The user can select a gender by clicking on one of the radio buttons and can select a job type from the drop down list.

4 Type some data into the various fields and select some items, then click on the OK button..

The form should go away and a message should appear telling you that you pressed the OK button and the information you entered should appear on the FrameMaker console.

Exercise 10. An Interactive Input Form

EForms can interact with the user as well as accept data. Most of the controls can optionally trigger events. For example, you can specify the name of an event to RUN when a button is clicked, when a listbox or dropdown box changes its selection or when you enter information into an Edit control..

Step: 1. Start with the previous script. If you do not want to enter the text manually, look at the enclosed script named 'InputDialog3.fsl'.

5 Change the command to create the Drop Down List to add an event option.

```
New EDropDownBox Form(ef) Top(55) Left(160) Width(240) OnSelChange('UpdateForm')
          LineCount(5) Height(100) NewVar(vDDBox);
```

6 Add the following lines before the ef.ShowModal command.

```
New ECheckBox Form(ef) Top(140) Left(10) Visible(False)
       Caption('Are you Evil?') NewVar(vBossCbxBox);
New ECheckBox Form(ef) Top(140) Left(10) Visible(False)
       Caption('Are you Gullible?') NewVar(vEngineerCbxBox);
New ECheckBox Form(ef) Top(140) Left(10) Visible(False)
       Caption('Are you a Fool?') NewVar(vTraineeCbxBox);
Run UpdateForm;
```

7 Add the following lines at the very end of the script file.

```
Sub UpdateForm

   Set vBossCbxBox.Visible = False;
   Set vEngineerCbxBox.Visible = False;
   Set vTraineeCbxBox.Visible = False;

   If vDDBox.SelIndex = 1
      Set vTraineeCbxBox.Visible = True;
      Set vRefVar.Enabled = True;
   ElseIf vDDBox.SelIndex = 2
      Set vEngineerCbxBox.Visible = True;
      Set vRefVar.Enabled = True;
   Else
      Set vBossCbxBox.Visible = True;
      Set vRefVar.Enabled = False;
   EndIf

EndSub
```

8 Run the script.

A similar form should appear as last time, except that there should be one more check box control. When the user selects a job type from the drop down list, a different check box appears depending on which item is selected. Also, the references field is disabled when the Dilber Boss type is chosen. Dilbert Bosses do not need references. The 'Are you Evil' checkbox is enough.

9 Click on the OK button to dismiss the form.

A Practical Modal Form

For a practical example of a modal form, look at the tutorial script called 'InsertTableModal.fsl'. This script mimics the dialog box that FrameMaker displays when you select the Table->Insert Table menu command on the FrameMaker user interface. There are two differences though. This sample script does not handle structured tables, so the Element Tag drop down is just for show. However, this form is resizeable.

Exercise 11. Testing the Insert Table Modal Form.

Do the following steps to test the Insert Table Modal form.

Step: 1. In FrameMaker, make a new document.

1 Select Table->Insert Table from the FrameMaker user interface.

2 Move the mouse to the lower right corner and make the dialog box larger.

The dialog box will get larger, but the fields stay in the same place and remain the same size.

3 Click on the Cancel button to dismiss the dialog.

4 Run the InsertTableModal.fsl script.

A form should appear that looks very much like FrameMaker's dialog box.

5 Move the mouse to the lower right corner and make the dialog box larger.

The dialog box will get larger, but, this time, the fields will move accordingly and the list of formats box will expand, letting you see more of the table format names (good for long table format names) and also showing more formats (good for documents with many table formats).

6 Create a sample table.

Modeless Form Example

Modeless forms allow you to do other things while they are on the screen. You do not need to respond to them right away. you can keep them handy for when you want to use it. The paragraph catalog dialog box is a good example of a commonly used modeless dialog box. You keep it on the screen somewhere and use it to apply paragraph formats when desired. In effect, a modeless form is like an Event script. It waits around until some event happens which it must handle. Usually this is a mouse click or keyboard entry. For this reason, modeless forms must be run from Event Scripts, not Standard scripts.

For a practical example of a modeless form, look at the tutorial script called 'VarCatalog.fsl'. This script acts like the FrameMaker paragraph format dialog box, except that it is designed to provide the same functionality for Variable Formats. The FrameMaker variable dialog is a Modal dialog. You have to bring it up every time you want to use it because it goes away as soon as you click the Insert button. This can be annoying if you use a lot of variables in your document.

Exercise 12. Test the VarCatalog.fsl script.

Step: 1. Install the VarCatalog.fsl tutorial script.

Use the ElmScript Install command to install the script. This should put a new menu command, under the FrameMaker Special menu, called **`'Variable Catalog...'`**.

1 Select this command

A resizeable variable catalog form should appear.

2 Enter or edit some text, then add FrameMaker variables whenever you wish by double clicking on a variable name in the window.

3 Click on the CloseBox of the form if ever you don't want to see it on the screen. You can always use the Menu command to bring it back up.

Chapter 15

EslSession Object

The EslSession object represents the Options information provided in the fscript.ini file and set by the user via the Options dialog box. These consist of a set of Read-Only properties.

EslSession Object Properties

Table 2: EslSession Properties

Property Name	Data Type	Property Description
General Information		
ProductName	String	The name of the product. This should be ElmScript.
Version	String	A string with the current version information.
SignonScreen	Integer	True if the signon screen option is on and False otherwise.
UseGlobalDataSpaceForInitialScript	Integer	True if the global variables are kept as Session variables after the initial script terminates, and False otherwise. The initial script is like any other script in that it has a set of global variables. When the script finishes, you have the option to keep these around for other scripts to use (as read-only variables) or you may choose to have these deleted when the script terminates.
UserName	String	The User name from the ElmScript installation.
CompanyName	String	The Company name from the ElmScript installation..
EslRegNum	String	The ElmScript registration number from the ElmScript installation.
FrameRegNum	String	The ElmScript registration number from the ElmScript installation.
UserSettingsDir	String	The directory name where ElmScript keeps the user settings file (fscript.ini) for the current session ElmScript.
File/Directory Information		
InitialScriptFileName	String	The name of the file to use as the initial script. An empty string if there is no initial script.
LogFileName	String	The name of the file to use as a log file. An empty string if there is no log file.
ScriptExtensions	String	A list of file extensions (separated by a semi-colon) to search when the extension of not given as part of a file name.

Table 2: EslSession Properties

Property Name	Data Type	Property Description
ScriptSearchPath	**StringList**	A list of folders that ElmScript uses to search when looking for a file that does not have a full path.
User Interface Menu Information		
IncludeMainMenuItem	**Integer**	True if the ElmScript menu is included., False if not.
MainMenuItemName	**String**	The name of the ElmScript menu.
IncludeRunMenuItem	**Integer**	True if the ElmScript Run menu item is included, False if not.
RunMenuItemName	**String**	The name of the ElmScript Run menu item.
IncludeInstallMenuItem	**Integer**	True if the ElmScript Install menu item is included, False if not.
InstallMenuItemName	**String**	The name of the ElmScript Install menu item.
IncludeUninstallMenuItem	**Integer**	True if the ElmScript Uninstall menu item is included, False if not.
UninstallMenuItemName	**String**	The name of the ElmScript Uninstall menu item.
IncludeOptionsMenuItem	**Integer**	True if the ElmScript Options menu item is included, False if not.
OptionsMenuItemName	**String**	The name of the ElmScript Options menu item.
IncludeScriptWindowMenuItem	**Integer**	True if the ElmScript Script Window menu item is included, False if not.
ScriptWindowMenuItemName	**String**	The name of the ElmScript Script Window menu item.
IncludeCompileMenuItem	**Integer**	True if the ElmScript Compile menu item is included, False if not.
CompileMenuItemName	**String**	The name of the ElmScript Compile menu item.
IncludeScriptsMenuItem	**Integer**	True if the ElmScript Scripts menu item is included, False if not.
ScriptsMenuItemName	**String**	The name of the ElmScript Scripts menu item.
Error Processing Information		
RunErrorDisplay	**Integer**	True if any run error is displayed in an alert box, False if not.
RunErrorConsole	**Integer**	True if any run error is written to the console, False if not.
RunErrorLogFile	**Integer**	True if any run error is written to the log file, False if not.
CompileErrorDisplay	**Integer**	True if any compile error is displayed in an alert box, False if not.
CompileErrorConsole	**Integer**	True if any compile error is written to the console, False if not.
CompileErrorLogFile	**Integer**	True if any compile error is written to the log file, False if not.

Chapter 16

eDebug Object

The eDebug object represents a set of properties and methods that help you to debug a script using a tracing technique. There is one eDebug object per running script. It is created when you first access it by setting or getting properties. Do not create or delete this object. It is always available by the name eDebug.

There are two types of debug output systems, both going to the same output. One is the message output. The Message and MessageLine methods write messages to the trace output, if the MessageStatus flag is True. You could perform the same operation with the Write Console command, but the MessageStatus lets you turn the messages on and off, without having to remove all the debug commands.

The other debug system involves tracing commands, subroutines, functions and data.

Note: The trace commands can be very time consuming. It is best to turn on a trace only for part of a script that is giving you problems (e.g. set the TraceStatus to True or False to turn it on and off respectively.

eDebug Object Properties

Table 3: eDebug Properties

Property Name	Data Type	Property Description
Status Information		
TraceStatus	Integer	True if the general trace function is on and False otherwise.
MessageStatus	Integer	True if the message output function is on and False otherwise.
What To Trace Information		
TraceCommands	Integer	True if the trace commands option is on and False otherwise. If this option is on and the TraceStatus option is also on, then each command will be written to the trace output.
TraceSubs	Integer	True if the trace subroutines option is on and False otherwise. If this option is on and the TraceStatus option is also on, then each subroutine entered and exited will be written to the trace output
TraceData	Integer	True if the trace Data option is on and False otherwise. If this option is on and the TraceStatus option is also on, then each time a variable's value is change the result will be written to the trace output
TraceUserFunctions	Integer	True if the trace user functions option is on and False otherwise. If this option is on and the TraceStatus option is also on, then each user function entered and exited will be written to the trace output

Table 3: eDebug Properties

Property Name	Data Type	Property Description
`TraceSystemFunctions`	`Integer`	True if the trace system functions option is on and False otherwise. If this option is on and the TraceStatus option is also on, then each system function entered will be written to the trace output
`Trace Output Information`		
`FileName`	`String`	The name of the file to use for trace output. Use NULL to write the trace information to the FrameMaker console. This is the default.
`Append`	`Integer`	True to use the append option on the next trace file that is opened, False to rewrite the trace file..

eDebug Object Methods

Table 4: eDebug Methods

Method Name	Method Description
`Message`	Writes a message to the trace output if the message status is on.
`MessageLine`	Writes a message to the trace output if the message status is on, including a linefeed at the end of the message.

Message

The Message command writes a string to the trace output, if the MessageStatus property is set to True, otherwise nothing happens.

Format:

```
eDebug.Message{strval};
```

Table 5: Message Options

Option Name	Option Description
`Message`	The message to write to the trace output

MessageLine

The Message command writes a string to the trace output, if the MessageStatus property is set to True, otherwise nothing happens. This is the same as the message method except that this method adds a line feed character at the end of the message.

Format:

```
eDebug.MessageLine{strval};
```

Table 6: MessageLine Options

Option Name	Option Description
`Message`	The message to write to the trace output

Example:
The following script gives an example of a debug run. It shows the output of a trace and some examples of turning the trace on and off. This simple script runs the three script commands with various trace and message settings

```
. . .
Set eDebug.TraceStatus=True;
Set eDebug.MessageStatus=True;
Set eDebug.TraceSubs=True;
Set eDebug.TraceUserFunctions=True;
Set eDebug.TraceSystemFunctions=True;
Set eDebug.TraceCommands=True;
Set eDebug.TraceData = True;
write console '***************All Traces are ON*******************';

GlobalVar gvIntVar(100) gvIntVar2(200);

Set gvIntVar3 = fnFunc1{gvIntVar1,gvIntVar2};
write console 'After function call-'+gvIntVar3;
Run sbSub1 pvValue1(gvIntVar) pvValue2(gvIntVar2);

Set eDebug.TraceCommands=False;
Set eDebug.TraceData = False;
write console '***************Turn off command and data trace*******************';

Set gvIntVar3 = fnFunc1{gvIntVar1,gvIntVar2};
write console 'After function call-'+gvIntVar3;
Run sbSub1 pvValue1(gvIntVar) pvValue2(gvIntVar2);

Set eDebug.MessageStatus=False;
write console '***************Turn off message output***************';

Set gvIntVar3 = fnFunc1{gvIntVar1,gvIntVar2};
write console 'After function call-'+gvIntVar3;
Run sbSub1 pvValue1(gvIntVar) pvValue2(gvIntVar2);

Set eDebug.TraceStatus=False;
write console '***************Turn off trace***************';

Set gvIntVar3 = fnFunc1{gvIntVar1,gvIntVar2};
write console 'After function call-'+gvIntVar3;
Run sbSub1 pvValue1(gvIntVar) pvValue2(gvIntVar2);
write console '***************END OF SCRIPT***************';

/****************** sbSUB1 ******************/
Sub sbSub1 using pvValue1 pvValue2
   write console 'In Sub 1';
   eDebug.MessageLine{'In Sub1 Debug Message'};
EndSub

/****************** fnFunc1 ******************/
Function fnFunc1  using pvInt1 pvInt2
   Set Result = pvInt1 + pvInt2;
EndFunc
```

Output:
Here is the output from running the above script. This output went to the FrameMaker console.

```
ETrace--SET Line(10)---SetVar(eDebug.TraceData) To(True)
ETrace--write Line(12)
***************All Traces are ON********************
ETrace--GlobalVar Line(14)
ETrace---SetGlobalVar(gvIntVar) Init(100)
ETrace--SetGlobalVar(gvIntVar) To(100)
ETrace---SetGlobalVar(gvIntVar2) Init(200)
ETrace--SetGlobalVar(gvIntVar2) To(200)
ETrace--SET Line(16)---SetVar(gvIntVar3) To(fnFunc1{gvIntVar1,gvIntVar2})
ETrace--UserFunction (fnFunc1)
ETrace--SET Line(54)---SetVar(Result) To(pvInt1+pvInt2)
ETrace--SetVar(Result) To(200)
ETrace--EndUserFunction (fnFunc1) Result(200)
ETrace--SetVar(gvIntVar3) To(200)
ETrace--write Line(17)
After function call-200
ETrace--Run Line(18)
ETrace--Sub (sbSub1)
ETrace--write Line(47)
In Sub 1
ETrace--CALLFUNC Line(48)
ETrace--Function (MessageLine)
In Sub1 Debug Message
ETrace--EndFunction (MessageLine) Result(NULL)
ETrace--EndSub (sbSub1)
ETrace--SET Line(20)---SetVar(eDebug.TraceCommands) To(False)
ETrace--SetVar(eDebug.TraceCommands) To(0)
ETrace--SetVar(eDebug.TraceData) To(0)
***************Turn off command and data trace********************
ETrace--UserFunction (fnFunc1)
ETrace--EndUserFunction (fnFunc1) Result(200)
After function call-200
ETrace--Sub (sbSub1)
In Sub 1
ETrace--Function (MessageLine)
In Sub1 Debug Message
ETrace--EndFunction (MessageLine) Result(NULL)
ETrace--EndSub (sbSub1)
***************Turn off message output***************
ETrace--UserFunction (fnFunc1)
ETrace--EndUserFunction (fnFunc1) Result(200)
After function call-200
ETrace--Sub (sbSub1)
In Sub 1
ETrace--Function (MessageLine)
ETrace--EndFunction (MessageLine) Result(NULL)
ETrace--EndSub (sbSub1)
***************Turn off trace***************
After function call-200
In Sub 1
***************END OF SCRIPT***************
```

Chapter 17

Utility Functions Object(eUtl)

The eStr object represents a set of utility functions. The eUtl is a permanent object. It is not necessary to create nor to delete this object. It is create when ElmScript starts and it lasts as long as ElmScript is around. You just need to use the name eUtl to access its members.

eUtl Object Methods

Table 7: eUtl Methods

Method Name	Method Description
Make Data Item Methods	
TextLoc	Returns a Text Location from the parameters.
TextRange	Returns a Text Range from the parameters.
Make List Methods	
StringList	Returns a StringList. Each parameter will be a string in the list.
IntList	Returns an IntList. Each parameter will be an integer in the list.
Make Array Methods	
EArray	Returns an EArray object. The parameters become the members of the array.
EVector	Returns an EVector object. The parameters become the initial members of the vector.
EStackList	Returns an EStackList object. The parameters become the initial members of the vector.
ECollection	Returns an ECollection object. The parameters become the initial members of the vector.
EStructure	Returns an EStructure object. The parameters become the initial members of the vector. These parameters come as pairs. The first parameter in the pair is a string value representing the name of the member and the second parameter in a pair is the value.
Table Methods	
GetTableRow	This method retrieves a Table Row object from a row number.
GetTableCell	This method retrieves a Table Cell object from a row object or row number and a column number.
GetTableColumn	This method retrieves a Table Column object from a column number.**FM 2015 or greater**
Color Methods	
GetRGB	This method returns an integer color value from the red, green and blue color values.

Table 7: eUtl Methods

Method Name	Method Description
GetRGBRed	This method returns the Red portion from an integer color value.
GetRGBGreen	This method returns the Green portion from an integer color value.
GetRGBBlue	This method returns the Blue portion from an integer color value.
Facet Methods	
GetFacet	This method returns facet information for the specified Inset, Facet name and data type.
HasFacet	This method returns True if the specified facet exists or False if not.
GetFacetNameList	This method retrieves the names of the facets in the specified inset.
Data Conversion Methods	
Convert	This function converts one data item to a different specified type.
CPToString	Returns a string from a set of integers (code points) representing character codes. This is similar, but easier to use, to the New String IntValue command. The was formerly **IntToString.**
StringToCP	Returns an integer (or IntList) value representing the code point(s) for the specified string. The was formerly **StringToInt**.
ItoS	Converts an integer value into a string using the optional base parameter**.**
StoI	Converts a string value into an integer (using the optional base parameter).
Installed Script Property Methods	
GetCmdSrcType	This method gets the type of the specified script name.
GetScriptCommandObject	This method gets the Command Object of the specified script name.
GetScriptFileName	This method gets the file name of the specified script name.
IsScriptAlwaysInstalled	This method gets an integer value indication whether the specified script name is automatically installed at start up.
Build Expression Methods(FM10 or greater)	
AddNewBuildExpr	This method builds a new expressinon.(FM10 or greater)
DeleteBuildExpr	This method deletes an expression.(FM10 or greater)
SetActiveBuildExpr	This method sets the active expression.(FM10 or greater)
GetActiveBuildExpr	This method gets the active expression.(FM10 or greater)
GetBuildExpr	This method gets the expression with the specified name.(FM10 or greater)
GetBuildExprCatalog	This method gets the expression catalog (StringList). (FM10 or greater)
Project Methods(FM 2017 or greater)	
NewProject	This method creates a new project.(**FM 2017 or greater**)
OpenProject	This method opens a project.(**FM 2017 or greater**)
SaveProject	This method saves the current project.(**FM 2017 or greater**)
AddLocationToProject	This method adds a location to the current project.(**FM 2017 or greater**)
DeleteComponentFromProject	This method deletes the component with the specified name.(**FM 2017 or greater**)

Table 7: eUtl Methods

Method Name	Method Description
SaveProject	This method saves the current project.(**FM 2017 or greater**)
EditComponentOfProject	This method edits the specified component in the current project.(**FM 2017 or greater**)
ExploreComponentofProject	This method explores the specified component in the current project.(**FM 2017 or greater**)
RenameComponentofProject	This method renames a component. (**FM 2017 or greater**)
Miscellaneous Methods	
Evaluate	This function evaluates the string argument as an expression and returns the result of the expression.
FCode	This function evaluates the string parameter and returns the Integer F-code value.
Round	This function returns a real number rounded to the specified number of decimal places.
Truncate	This function returns a real number truncated to the specified number of decimal places.
GetFileNameFromSearchPath	This method makes a full path name from a relative file name (without a path), using the ElmScript Search path.
GetPropertyValue	This method retrieves the value of a property of the specified object.
SetPropertyValue	This method sets the value of a property of the specified object.
FormatString	This method returns a string value based on the format provided and a list of data items.
MoveComponent	This method moves a book component within a book.
UpdateXRef	This method updates a specified cross-reference in the document.
ApplyAttributeExpr	This method applies an attribute expression to the document to perform the attribute based filtering.
TrackChangesAcceptAll	This method accepts track changes. FM10 or greater
TrackChangesRejectAll	This method rejects track changes. FM10 or greater
ConvertTiToText	This method converts a text inset to text. FM10 or greater
ForceUpdateXrefAltText	This method forces update a cross-ref. FM10 or greater
UpdateMenus	This method updates menus. FM10 or greater
NewBook	This method creates a new FM book. FM10 or greater
DeleteUnusedFmts	This method deletes unused formats. FM10 or greater
GetConditionalExpression	This method returns the expression for the given conditional tag. **FM 2015 or greater**
GetWorkspaceName	This method returns the current workspace. **FM 2017 or greater**
SetWorkspaceName	This method sets the current workspace. **FM 2017 or greater**
ApplyFitToFrame	This method fitss the image to the frame or the reverse. **FM 2019 or greater**

Make Data Item Functions

TextLoc{}

The TextLoc method is a function that builds and returns a TextLoc value.

Format:

```
Set textLocVal = eUtl.TextLoc{docObject,pgfObject,offset};
```

Table 8: TextLoc{} Options

Option Name	Option Description
docObject	A Frame document object. If this is omitted then the currently active document is used.
pgfObject	A Frame paragraph object. If omitted then the paragraph at current insertion point in the currently active document is used.
offset	An integer offset into the paragraph. If not specified, then zero is used.
textLocVal	The returned TextLoc value.

Example:

The following script creates a TextLoc value from the paragraph object in the gvPgfVar variable at offset 2 in the currently active document.

```
. . .
Set gvTextLocVal = eUtl.TextLoc{,gvPgfVar,2};
New Text TextLoc(gvTextLocVal) 'Insert This Text';
. . .
```

TextRange{}

The TextRange method is a function that builds and returns a TextRange value.

Format:

```
Set textRangeVal = eUtl.TextRange{docObject,beginPgf,offset1,endPgf,offset2};
```

Table 9: TextRange{} Options

Option Name	Option Description
docObject	A Frame document object. If this is omitted then the currently active document is used.
beginPgf	A Frame paragraph object of the beginning of the range. If omitted then the paragraph at current insertion point in the currently active document is used.
offset1	An integer offset into the begin paragraph. If not specified, then zero is used.
endPgf	A Frame paragraph object of the end of the range. If omitted then the beginning paragraph is used.
offset2	An integer offset into the end paragraph. If not specified, then zero is used.
textRangeVal	The returned TextRange value.

Example:

The following script creates a TextRange value from the paragraph object in the gvPgfVar variable at offset 2 in the currently active document through offset 4 in the same paragraph. Then it displays the text in that range.

```
. . .
Set gvTextRangeVal = eUtl.TextRange{,gvPgfVar,2,,4};
Display gvTextRangeVal.Text;
. . .
```

Make List Functions

StringList{}

The StringList method is a function that builds and returns a StringList value.

Format:

```
Set strListVar = eUtl.StringList{str1[,str2]...[,strN]};
```

Table 10: StringList{} Options

Option Name	Option Description
strI	A string value or something that can be converted into a string.
strListVar	The variable to hold the returned StringList value.

Example:

The following script creates a StringList value from a list of strings and displays them in a dialog box.

```
. . .
Set gvStrDevList = eUtl.StringList{'VCR','DVD','CD Player'};
DialogBox Type(ScrollBox) List(gvStrDevList) Title('Select a device')
       NewVar(gvDevString) Button(gvButton);
. . .
```

IntList{}

The IntList method is a function that builds and returns an IntList value.

Format:

```
Set intListVar = eUtl.IntList{int1[,int2]...[,intN]};
```

Table 11: IntList{} Options

Option Name	Option Description
intI	An integer value or something that can be converted into an integer.
intListVar	The variable to hold the returned IntList value.

Example:

The following script creates a IntList value from a list of integers.

```
. . .
Set gvNumberList = eUtl.IntList{100,200,300,400,500,600};
. . .
```

Make Array Functions

EArray{}

The EArray method is a function that builds and returns an EArray object value.

Format:

```
Set fixedArrayVar = eUtl.EArray{val1[,val2]...[,valN]};
```

Table 12: EArray{} Options

Option Name	Option Description
valI	Any data value.
fixedArrayVar	The variable to hold the returned EArray object.

Example:

The following script creates an EArray object (fixed size array) from a list of disparate values.

```
. . .
Set gvArray = eUtl.EArray{100, 'MyString',ActiveDoc};
. . .
```

EVector{}

The EVector method is a function that builds and returns an EVector object value.

Format:

```
Set vectorArrayVar = eUtl.EVector{val1[,val2]...[,valN]};
```

Table 13: EVector{} Options

Option Name	Option Description
valI	Any data value.
vectorArrayVar	The variable to hold the returned EVector object.

Example:

The following script creates an EVector object (variable size array) from a list of disparate values, then later adds a new member to the end of the list.

```
. . .
Set gvVector = eUtl.EVector{100, 'MyString',ActiveDoc};
. . .
Set gvVector.PushBack = 'My New String';
. . .
```

EStackList{}

The EStackList method is a function that builds and returns an EStackList object value.

Format:

```
Set stackArrayVar = eUtl.EStackList{val1[,val2]...[,valN]};
```

Table 14: EStackList{} Options

Option Name	Option Description
valI	Any data value.
stackArrayVar	The variable to hold the returned EStackList object.

Example:
The following script creates an EStackList object from a list of disparate values, then later pops them off.

```
. . .
Set gvStackList = eUtl.EStackList{100, 'MyString',ActiveDoc};
. . .
Set gvDocVar = gvStackList.Pop{}; // This is a method be sure to include the {}
Set gvStrVar = gvStackList.Pop{}; // This is a method be sure to include the {}
. . .
```

ECollection{}

The ECollection method is a function that builds and returns an ECollection object value.

Format:

```
Set collArrayVar = eUtl.ECollection{val1[,val2]...[,valN]};
```

Table 15: ECollection{} Options

Option Name	Option Description
valI	Any data value.
collArrayVar	The variable to hold the returned ECollection object.

Example:
The following script creates an ECollection object from a list of disparate values, then writes the values to the console.

```
. . .
Set gvCollVar = eUtl.ECollection{100, 'MyString',ActiveDoc};
. . .
Set gvMember = gvCollVar.FirstMember;
Loop While(gvMember)
   Write console 'Member Data-'+gvMember;
   Set gvMember = gvMember.NextMember;
EndLoop
. . .
```

EStructure{}

The EStructure method is a function that builds and returns an EStructure object value. The parameters come in pairs. The first of each pair is the data name and the second one is the associate data value.

Format:

```
Set structVar = eUtl.EStructure{'name1',val1[,'name2',val2]...[,'nameN',valN]};
```

Table 16: EStructure{} Options

Option Name	Option Description
nameI	Any data value.
valI	Any data value.
structVar	The variable to hold the returned EStructure object.

Example:
The following script creates an EStructure object from a list of disparate pairs of values, then writes the values to the console.

```
. . .
Set gvStruct = eUtl.EStructure{'svNum',100,'svString','MyString','svDoc',ActiveDoc};
. . .
Write console 'svNum Member-'+gvStruct.svNum;
Write console 'svString Member-'+gvStruct.svString;
Write console 'svDoc Member-'+gvStruct.svDoc;
. . .
```

Table Functions

GetTableRow{}

The GetTableRow method retrieves a table row object from the row number.

Format:

```
Set rowObjectVar = eUtl.GetTableRow{tblObject,rowNumber};
```

Table 17: GetTableRow{} Options

Option Name	Option Description
tblObject	The Table Object containing the row.
rowNumber	An integer row number. **Row numbers start at 0**.
rowObjectVar	The Row Object returned. This will be NULL if the row number is out of range.

Example:
The following script returns the 5th row from the current table in the active document.

```
. . .
Set vTblVar = ActiveDoc.SelectedTbl;
If vTblVar
    Set vRowVar = eUtl.GetTableRow{vTblVar, 5};
    Write Console 'Row number 5 is '+vRowVar;
Else
    Write console 'No selected table';
EndIf
. . .
```

GetTableCell{}

The GetTableCell method retrieves a table cell object from a row object (or row number) and a column number.

Format:

```
Set cellObjectVar = eUtl.GetTableCell{tblObject, rowObject, columnNumber};
or
Set cellObjectVar = eUtl.GetTableCell{tblObject, rowNumber, columnNumber};
```

Table 18: GetTableCell{} Options

Option Name	Option Description
tblObject	The Table Object containing the cell.
rowObject	The Row Object of the cell.
rowNumber	An integer row number. **Row numbers start at 0**.
columnNumber	An integer column number. **Column numbers start at 0**.
cellObjectVar	The Cell Object returned. This will be NULL if either the row or column number is out of range. Note: In the FrameMaker architecture, straddled cells are treated as separate cells. Use the **CellIsStraddled** cell property to see if a cell is straddled.

Example 1:
The following script retrieves each table cell by row and column number and writes the text to the FrameMaker console.

```
Set colCount=tblVar.TblNumCols;
Set rowCount=tblVar.TblNumRows;
Loop InitVal(0) Incr(1) LoopVar(rowIdx) While(rowIdx<rowCount)
   Loop InitVal(0) Incr(1) LoopVar(colIdx) While(colIdx<colCount)
      Set cellVar = eUtl.GetTableCell{tblVar, rowIdx, colIdx};
      If cellVar
         Set pgfVar=cellVar.FirstPgf;
         Write Console 'GetTableCell('+rowIdx+','+colIdx+')-'+
            ' Cell First Pgf Text-'+pgfVar.Text;
      Else
         Write Console 'Cell Missing';
      EndIf
   EndLoop
EndLoop
. . .
```

Example 2:
The following script retrieves each table cell by Row Object and column number and writes the text to the FrameMaker console.

```
Set colCount=tblVar.TblNumCols;
Set rowVar = tblVar.FirstRowInTbl;
Loop While(rowVar)
   Loop InitVal(0) Incr(1) LoopVar(colIdx) While(colIdx<colCount)
      Set cellVar = eUtl.GetTableCell{tblVar, rowIdx, colIdx};
      If cellVar
         Set pgfVar=cellVar.FirstPgf;
         Write Console 'GetTableCell('+colIdx+')-'+
            ' Cell First Pgf Text-'+pgfVar.Text;
      Else
         Write Console 'Cell Missing';
      EndIf
   EndLoop
EndLoop
. . .
```

GetTableColumn{}

The GetTableColumn method retrieves a table column object from a column number. **`FM 2015 or greater`**

Format:

```
Set columnObjectVar = eUtl.GetTableColumn{tblObject, columnNumber};
```

Table 19: GetTableColumn{} Options

Option Name	Option Description
`tblObject`	The Table Object containing the cell.

Table 19: GetTableColumn{} Options

Option Name	Option Description
columnNumber	An integer column number. **Column numbers start at 0**.
columnObjectVar	The Column Object returned. This will be NULL if the column number is out of range. Note: In the FrameMaker architecture, straddled cells are treated as separate cells. Use the **CellIsStraddled** cell property to see if a cell is straddled.

Example 1:
The following script retrieves each table column.

```
Set colCount=tblVar.TblNumCols;
Loop InitVal(0) Incr(1) LoopVar(colIdx) While(colIdx<colCount)
   Set colVar = eUtl.GetTableColumn{tblVar, colIdx};
EndLoop
. . .
```

Color Functions

GetRGB{}

The GetRGB function combines three separate color intensities into an integer color value. A color value can be used for the foreground or background color properties on form controls.

Format:

```
Set gvColorVar = eUtl.GetRGB{redValue,greenValue,blueValue};
```

Table 20: GetRGB{} Options

Option Name	Option Description
redValue	The red component of the color value. Valid values are from 0 to 255.
greenValue	The green component of the color value. Valid values are from 0 to 255.
blueValue	The blue component of the color value. Valid values are from 0 to 255.
gvColorVar	The integer color value. This is the value that can be used for the foreground or background colors on form controls.

Example:
The following script makes a bunch of color values and uses two of them for the label.

```
. . .
Set gvColorWhite = eUtl.GetRGB{255, 255, 255};
Set gvBlackColor = eUtl.GetRGB{0,0,0};
Set gvColorRed = eUtl.GetRGB{255,0,0};
Set gvColorBlue = eUtl.GetRGB{0,0,255};
Set gvColorGreen = eUtl.GetRGB{0,255,0};
Set gvColorStandard = eUtl.GetRGB{236,233,216};
New EForm NewVar(gvForm) Top(100) Left(100) Width(300) Height(500);
New ELabel Form(gvForm) NewVar(gvLBL) Top(50) Left(50)
                  ForegroundColor(gvColorRed)
                  BackgroundColor(gvColorGreen)
                  Caption('My Colored Label');
. . .
```

GetRGBRed{}

The GetRGBRed method retrieves the red component of the integer color value.

Format:

```
Set gvRedValueVar = eUtl.GetRGBRed{gvColor};
```

Table 21: GetRGBRed{} Options

Option Value	Option Description
`gvColor`	A previously made integer color value. See “GetRGB{}” on page 249
`gvRedValueVar`	The red part of the color value.

Example:
The following script makes a color value then later gets the red component.

```
Set gvColorValue=eUtl.GetRGB{236,233,111};
...
Set gvRedValue=eUtl.GetRGBRed{gvColor}; // Should be 236

. . .
```

GetRGBGreen{}

The GetRGBGreen method retrieves the green component of the integer color value.

Format:

```
Set gvGreenValue = eUtl.GetRGBGreen{gvColor};
```

Table 22: GetRGBGreen{} Options

Option Value	Option Description
gvColor	A previously made integer color value. See “GetRGB{}” on page 249
gvGreenValue	The green part of the color value.

Example:
The following script makes a color value then later gets the green component.

```
Set gvColorValue=eUtl.GetRGB{236,233,111};
...
Set gvGreenValue=eUtl.GetRGBGreen{gvColor}; // Should be 233
. . .
```

GetRGBBlue{}

The GetRGBBlue method retrieves the blue component of the integer color value.

Format:

```
Set gvBlueValue = eUtl.GetRGBBlue{gvColor};
```

Table 23: GetRGBBlue{} Options

Option Value	Option Description
gvColor	A previously made integer color value. See “GetRGB{}” on page 249
gvBlueValue	The blue part of the color value.

Example:
The following script makes a color value then later gets the blue component.

```
Set gvColorValue=eUtl.GetRGB{236,233,111};
...
Set gvBlueValue=eUtl.GetRGBBlue{gvColor}; // Should be 111
. . .
```

Facet Functions

In addition to the properties described in the RefManual, Graphic Insets (ObjectName=’Inset’) can have additional properties called facets. The names of these properties vary depending on the type of graphic the inset represents. For example, a .gif image will have a facet named GIF. See the list of Facet Names below.

There are three functions that deal with facets: GetFacet, HasFacet and GetFacetNameList. The GetFacet retrieves the facet information (Integer, Metric or UBytes data types). If you want to determine whether an inset has a facet by a name, use the HasFacet method. This is faster than the GetFacet function. The GetFacetNameList function will return a list of facet names for the specified inset.

A Graphic Inset will have at least one of the following Facet names:

```
'DCS Black','DCS Cyan','DCS Magenta','DCS Yellow','CGM','EPSI',
'FrameImage','FrameVector','GIF','MacPaint','PCX','SVG','TIFF',
'XWD','DIB','EMF','OLE','WMF'
```

GetFacet{}

The GetFacet function retrieves a named facet from a specified graphic inset object. It will return Null if no facet is found.

Format:

```
Set gvFacetVar = eUtl.GetFacet{insetObj,name[,dType]};
```

Table 24: GetFacet{} Options

Option Name	Option Description
`insetObj`	The Graphic Inset object.
`name`	The name (string) of the facet. This name is case sensitive.
`dType`	The data type desired (`'Integer'` for an integer facet, `'Metric'` for a metric facet or `'UBytes'` for a binary facet. If not specified, this function will look first for an integer facet. If not found, it will look for a Metric facet, then a UBytes facet.
`gvFacetVar`	The facet data item. This could be an integer, a real (Metric) or UBytes (binary data).

Example:

The following script gets the GIF facet from the selected graphic.

```
. . .
Set gvGraphic=FirstSelectedGraphicInDoc;
If gvGraphic
   Set gvFacet=eUtl.GetFacet{gvGraphic,'GIF','UBytes'};
   If gvFacet
      Display 'Graphic is a GIF';
   Else
      Display 'Graphic does not have a GIF facet';
   EndIf
EndIf
. . .
```

HasFacet{}

The HasFacet method returns True if the specified Graphic Inset has a facet by the specified name and False otherwise.

Format:

```
Set gvBoolVar = eUtl.HasFacet{insetObj,name[,dType]};
```

Table 25: HasFacet{} Options

Option Value	Option Description
insetObj	The Graphic Inset object.
name	The name (string) of the facet.
dType	The data type desired (`'Integer'` for an integer facet, `'Metric'` for a metric facet or `'UBytes'` for a binary facet. If this option is specified, the function will only look for the facet type specified. If not specified, it will look for each type in the following order: Integer, Metric, UBytes. If any of these are found, the function will return True.
gvBoolVar	The return value: True if the inset has the named facet and False otherwise.

Example:
The following script checks to determine if the selected graphic has a TIFF facet.

```
. . .
Set gvGraphic=FirstSelectedGraphicInDoc;
If gvGraphic
   Set gvCmp=eUtl.HasFacet{gvGraphic,'TIFF'};
   If gvCmp
      Display 'Graphic has a TIFF facet';
   Else
      Display 'Graphic does not have a TIFF facet';
   EndIf
EndIf
. . .
```

GetFacetNameList{}

The GetFacetNameList function retrieves the names of the facets in the specified inset. It will return Null if no facet names are found.

Format:

```
Set gvNameListVar = eUtl.GetFacetNameList{insetObj};
```

Table 26: GetFacetNameList{} Options

Option Name	Option Description
insetObj	The Graphic Inset object.
gvNameListVar	The list of facet names (StringList) or Null if no facet names exist.

Example:
The following script gets all the facet names from the selected graphic.

```
. . .
Set gvGraphic=FirstSelectedGraphicInDoc;
If gvGraphic
   Set gvNameList=eUtl.GetFacetNameList{gvGraphic};
   If gvNameList
      Display 'Facet Names'+gvNameList;
   Else
      Display 'Graphic does not have any facets';
   EndIf
EndIf
. . .
```

Data Conversion Methods

Convert{}

The Convert method converts the specified data item to the specified type. Not all conversions are possible. It will only work if it makes sense and there is enough information. For example, you can convert a TextLoc into an Object (Id) because a TextLoc contains an Object, but you could not convert a Real into an Object.

Format:

```
Set newDataItemVar = eUtl.Convert{dataItem, dataType};
```

Table 27: Convert{} Options

Option Name	Option Description
dataItem	The data item to convert.
dataType	A string value indicating the data type of the new data item (e.g. 'Integer', 'TextLoc', etc. See "List of Data Type Names" on page 255).
newDataItem	A data value returned when the conversion is complete.

Example 1:
The following script converts an integer and a string value to real values.

```
. . .
Set gvInt=999;
Set gvString='888';
Set gvReal=eUtl.Convert{gvInt, 'Real'};
Set gvReal2=eUtl.Convert{gvString, 'Real'};
Write console 'Real from Integer'+gvReal+' Real from String-'+gvReal2;
. . .
```

Example 2:
The following script gets the current element range and converts the begin ElementLoc to a TextLoc and back again..

```
. . .
Set gvEltRange=gvFileVar.ElementSelection;
Set gvEltLoc=gvEltRange.Begin;
Write Console 'EltLoc -'+gvEltLoc;
Set gvTextLoc=eUtl.Convert{gvEltLoc,'TextLoc'};
Write Console 'EltLoc To TextLoc-'+gvTextLoc;
Set gvEltLoc2=eUtl.Convert{gvTextLoc,'ElementLoc'};
Write Console 'TextLoc To EltLoc-'+gvEltLoc2;
. . .
```

Table 28: List of Data Type Names

Value	Description
`'Attribute'`	Attribute
`'AttributeDef'`	AttributeDef
`'AttributeDefs'`	AttributeDefList
`'Attributes'`	AttributeList
`'ElementLoc'`	Element Location
`'ElementRange'`	Element Range
`'Id'`	Frame Object
`'Integer'`	
`'Ints'`	IntList
`'Metric'`	
`'Metrics'`	MetricList
`'Null'`	
`'Point'`	
`'Points'`	Point List
`'Property'`	
`'PropertyList'`	
`'Real'`	
`'String'`	
`'Strings'`	StringList
`'Tab'`	
`'Tabs'`	TabList
`'TextItem'`	
`'TextItemList'`	
`'TextLoc'`	
`'TextRange'`	

Table 28: List of Data Type Names

Value	Description
'UBytes'	UByteList
'UInts'	UIntList

CPToString{}

The CPToString method is a function that returns a string value computed from the set of integer code point parameters. These integer parameters represent the character codes of the string characters. These code points must be value for the current text encoding mode. Usually these codepoints use values between 32 and 255 for the FrameRoman and Ansi character sets. However, if you are using FrameMaker 8 or greater and are in the UTF-8 encoding mode, you can use larger values.

Format:

```
Set strVal = eUtl.CPToString{int1 [,int2],...[,intN]};
```

Table 29: CPToString{} Options

Option Name	Option Description
intI	An integer value representing a code point value.
strVal	The return string value.

Example 1:
The following script creates a two character string using the code points 160 (†) and 224 (‡). These are the character representations in the FrameMaker character set. The string is then inserted into the currently active document and the current insertion point.

```
. . .
Set gvStrVal = eUtl.CPToString{160,224};
New Text gvStrVal;
. . .
```

Example 2:
The following script creates a two character string using the code points 134 (†) and 135 (‡). These are the character representations in the MS Windows ANSI character set. The string is then inserted into the currently active document and the current insertion point, after it converts the string to the FrameMaker character set.

```
. . .
Set gvStrVal = eUtl.CPToString{134,135};
New Text gvStrVal.PlatformToFrame{};
. . .
```

StringToCP{}

The StringToCP method is a function that returns an integer or an IntList value computed from each character in the string. The integer(s) returned represent the character codes of each of the string characters.

Format:

```
Set intVal = eUtl.StringtoCP{String};
or
Set intListVal = eUtl.StringtoCP(String);
```

Table 30: StringToCP{} Options

Option Name	Option Description
String	The source string.
intVal	An integer value (code point) returned when the source string is one character long.
intListVal	An IntList value (list of code points) returned when the source string is greater than one character.

Example:
The following script converts the string 'A' into the integer character code representation of the letter.

```
. . .
Set gvIntVal = eUtl.StringToInt{'A'};
. . .
```

ItoS{}

The ItoS method converts an integer into a string using the selected number base.

Format:

```
Set strVal = eUtl.ItoS{intValue[,base]};
```

Table 31: ItoS{} Options

Option Name	Option Description
intValue	The integer to convert.
base	The base of the resulting string. The default value is 10. If you wish the string to be a hexadecimal representation of the integer use 16 as the base value (8 for octal or even 2 for binary). If specified, this value must be between 2 and 36.
strVal	The return string value.

Example 1:
The following script converts the integer into a string.

```
. . .
Set gvStrVal = eUtl.ItoS{65};
New Text gvStrVal; // The string contains '65'
Set gvStrVal = eUtl.ItoS{65,16};
New Text gvStrVal; // The string contains '41'
Set gvStrVal = eUtl.ItoS{65,8};
New Text gvStrVal; // The string contains '101'
. . .
```

StoI{}

The StoI method converts a string value into an integer. The option base value indicates the base of the number represented by the string. This conversion proceeds left to right and stops at the end of when an invalid character is reached.

Format:

```
Set intVal = eUtl.StoI{String[,base]};
```

Table 32: StoI{} Options

Option Name	Option Description
String	The source string.
base	The base of the source string. The default value is 10. If you wish to identify the string to represent a hexadecimal integer use 16 as the base value (8 for octal or even 2 for binary). If specified, this value must be between 2 and 36.
intVal	An integer value returned.

Example:
The following script converts the string '4F' into the integer value.

```
. . .
Set gvIntVal = eUtl.StoI{'4F',16}; // The integer value is 79
. . .
```

Installed Script Property Methods

GetCmdSrcType{}

The GetCmdSrcType function returns the type of the specified script name.

Format:

```
Set cmdType = eUtl.GetCmdSrcType{scriptName};
```

Table 33: GetCmdSrcType{} Options

Option Name	Option Description
scriptName	The name of the script.
cmdType	The type of the script. `'Event'` for an event script and `'Standard'` for a standard script. This returns `Null` if the script name is not a currently installed script.

Example:
See the example at the end of the installed script property section. See “The following script writes the information for all the installed scripts to the FrameMaker console.” on page 260..

GetScriptCommandObject{}

The GetScriptCommandObject method retrieves the menu command object of the specified script name.

Format:

```
Set cmdObject = eUtl.GetScriptCommandObject{scriptName};
```

Table 34: GetScriptCommandObject{} Options

Option Value	Option Description
scriptName	The name of the script.
cmdObject	The menu command object of the script if the script is a standard script This returns `Null` if the script name is not a standard script or if it is not a currently installed script.

Example:
See the example at the end of the installed script property section. See "The following script writes the information for all the installed scripts to the FrameMaker console." on page 260..

GetScriptFileName{}

The GetScriptFileName method retrieves the file name of the specified script name.

Format:

```
Set scriptFileName = eUtl.GetScriptFileName{scriptName};
```

Table 35: GetScriptFileName{} Options

Option Value	Option Description
scriptName	The name of the script.
scriptFileName	The name of the file associated with the specified scriptname. This returns `Null` if the script name is not a currently installed script.

Example:
See the example at the end of the installed script property section. See "The following script writes the information for all the installed scripts to the FrameMaker console." on page 260..

IsScriptAlwaysInstalled{}

The IsScriptAlwaysInstalled method retrieves the blue component of the integer color value.

Format:

```
Set isAlwaysInstalled = eUtl.IsScriptAlwaysInstalled{scriptName};
```

Table 36: IsScriptAlwaysInstalled{} Options

Option Value	Option Description
scriptName	The name of the script.
isAlwaysInstalled	This is `True` if the specified scriptname is an automatically installed script. False, otherwise.

Example:
The following script writes the information for all the installed scripts to the FrameMaker console.

```
Set gvScriptList = InstalledScriptList;
If gvScriptList.Count < 1
   MsgBox 'There are no installed scripts';
   LeaveSub;
EndIf

Loop ForEach(Member) In(gvScriptList) LoopVar(gvScriptName);
   Set gvScriptFileName=eUtl.GetScriptFileName{gvScriptName};
   Set gvAlways=eUtl.IsScriptAlwaysInstalled{gvScriptName};
   Set gvType=eUtl.GetCmdSrcType{gvScriptName};
   If gvType='Standard'
      Set gvObj=eUtl.GetScriptCommandObject{gvScriptName};
      Write Console 'Standard Script-'+gvScriptName+' Always-'+gvAlways+
         ' File-'+gvScriptFileName;
      If gvObj
         Write Console '   From Menu';
         Write Console '       Name-'+gvObj.ScriptName;
         Write Console '       Always-'+gvObj.AlwaysInstall;
         Write Console '       Type-'+gvObj.CmdSrcType;
         Write Console '       File-'+gvObj.ScriptFileName;
      Else
         Write Console '   No Menu Item';
      EndIf
   Else
      Set gvObj=NULL;
      Write Console 'Event Script-'+gvScriptName+
         ' Always-'+gvAlways+' File-'+gvScriptFileName+' Always-'+gvAlways;
   EndIf

EndLoop
. . .
```

Build Expression Methods

AddNewBuildExpr{}

The AddNewBuildExpr function creates a new build expression. FM10 or greater

Format:

```
Set rc = eUtl.AddNewBuildExpr{docObject, exprName, exprCond};
```

Table 37: AddNewBuildExpr{} Options

Option Name	Option Description
docObject	The Document Object.
exprName	The name of the build expression.

Table 37: AddNewBuildExpr{} Options

Option Name	Option Description
exprCond	The build expression.
rc	The return value

DeleteBuildExpr{}

The DeleteBuildExpr function deletes a build expression. FM10 or greater

Format:

```
Set rc = eUtl.DeleteBuildExpr{docObject, exprName};
```

Table 38: DeleteBuildExpr{} Options

Option Name	Option Description
docObject	The Document Object.
exprName	The name of the build expression.
rc	The return value

SetActiveBuildExpr{}

The SetActiveBuildExpr function sets the active build expression. FM10 or greater

Format:

```
Set rc = eUtl.SetActiveBuildExpr{docObject, exprName};
```

Table 39: SetActiveBuildExpr{} Options

Option Name	Option Description
docObject	The Document Object.
exprName	The name of the build expression.
rc	The return value

GetActiveBuildExpr{}

The GetActiveBuildExpr function gets the active build expression. FM10 or greater

Format:

```
Set expr = eUtl.GetActiveBuildExpr{docObject};
```

Table 40: GetActiveBuildExpr{} Options

Option Name	Option Description
docObject	The Document Object.
expr	The active build expression

GetBuildExpr{}

The GetBuildExpr function gets the build expression by name. FM10 or greater

Format:

```
Set expr = eUtl.GetBuildExpr{docObject, exprName};
```

Table 41: GetBuildExpr{} Options

Option Name	Option Description
docObject	The Document Object.
exprName	The name of the build expression.
expr	The returned build expression

GetBuildExprCatalog{}

The GetBuildExprCatalog function gets a list of build expressions. FM10 or greater

Format:

```
Set exprList = eUtl.GetBuildExprCatalog{docObject};
```

Table 42: GetBuildExprCatalog{} Options

Option Name	Option Description
docObject	The Document Object.
exprList	The returned build expression list.

Project Methods (FM 2017 or greater)

NewProject{}

This method creates a new project.

Format:

```
Set retVal = eUtl.NewProject{strProjectName,strRootFolderName};
```

Table 43: NewProject{} Options

Option Name	Option Description
strProjectName	The name of the project.
strRootFolderName	The folder location to save the project.
retVal	A return code.

Example:
The following script creates a new project.

```
. . .

Set gvVal = eUtl.NewProject{'MyProject','c:\Projects\MyProject'};
Write console 'Value -'+gvVal;
. . .
```

OpenProject{}

The Open Project method opens an existing project.

Format:

```
Set retVal = eUtl.OpenProject{projectFileName};
```

Table 44: OpenProject{} Options

Option Name	Option Description
projectFileName	The name of the project file.
retVal	A return code.

Example:
The following script opens an existing project.

```
. . .

Set retVal=eUtl.OpenProject{'c:\Projects\MyProject\MyProject.fxpr'};
Write console 'retVal -'+retVal;
. . .
```

SaveProject{}

The Save Project method saves the currently active project.

Format:

```
Set retVal=eUtl.SaveProject{};
Write console 'retVal -'+retVal;
. . .
```

Table 45: SaveProject{} Options

Option Name	Option Description
retVal	A return code

Example:
The following script saves the currently active project.

```
. . .

Set retVal=eUtl.SaveProject{};
Write console 'retVal -'+retVal;
. . .
```

AddLocationToProject{}

This method adds a location to the project.

Format:

```
Set retVal = eUtl.AddLocationToProject{strLocationPath,strLocationName};
```

Table 46: AddLocationToProject{} Options

Option Name	Option Description
strLocationPath	The location to add to the project.
strLocationName	The name for the location being added..
retVal	A return code.

Example:
The following script adds a new location to the existing project and also gives it a name.

```
. . .

Set retVal=eUtl.AddLocationToProject{'c:\Projects\MyProject\proj2.png','proj2'};
Write console 'retVal -'+retVal;
. . .
```

DeleteComponentFromProject{}

This method deletes a component from the existing project.

Format:

```
Set retVal = eUtl.DeleteComponentFromProject{strComponentFullPath};
```

Table 47: DeleteComponentFromProject{} Options

Option Name	Option Description
strComponentFullPath	The absolute path of the component to be removed.
retVal	A return code.

Example:
The following script deletes the spcified component from the existing project.

```
. . .

Set retVal=eUtl.DeleteComponentFromProject{'c:\Projects\MyProject\proj2.png'};
Write console 'retVal -'+retVal;
. . .
```

EditComponentOfProject{}

This method opens the selected file for editting.

Format:

```
Set retVal = eUtl.EditComponentOfProject{strComponentFullPath};
```

Table 48: EditComponentOfProject{} Options

Option Name	Option Description
strComponentFullPath	The absolute path of the component to be editted.
retVal	A return code.

Example:
The following script edits a component of the existing project.

```
. . .

Set retVal=eUtl.EditComponentOfProject{'c:\Projects\MyProject\proj2.png'};
Write console 'retVal -'+retVal;
. . .
```

ExploreComponentOfProject{}

This method shows the specified component in windows explorer.

Format:

```
Set retVal = eUtl.ExploreComponentOfProject{strComponentFullPath};
```

Table 49: ExploreComponentOfProject{} Options

Option Name	Option Description
strComponentFullPath	The absolute path of the component.
retVal	A return code.

Example:
The following script shows the component in Windows explorer.

```
. . .

Set retVal=eUtl.ExploreComponentOfProject{'c:\Projects\MyProject\proj2.png'};
Write console 'retVal -'+retVal;
. . .
```

RenameComponentofProject{}

This method renames a component of the current project.

Format:

```
Set retVal = eUtl.RenameComponentofProject{strComponentFullPath,strNewName};
```

Table 50: RenameComponentofProject{} Options

Option Name	Option Description
strComponentFullPath	The location to add to the project.
strNewName	The name for the location being added..
retVal	A return code.

Example:
The following script renames a component of the current project.

```
. . .

Set
retVal=eUtl.RenameComponentofProject{'c:\Projects\MyProject\proj2.png','proj2A.png'};
Write console 'retVal -'+retVal;
. . .
```

Miscellaneous Functions

Evaluate{}

The Evaluate method evaluates the expression enclosed in the string parameter and returns the value of the expression. This allows you to build a computation dynamically and get the answer. This function has access to all the variables in the current data space.

Format:

```
Set dataVal = eUtl.Evaluate{String};
```

Table 51: Evaluate{} Options

Option Name	Option Description
string	The expression string.
dataVal	A data value returned when the expression in the source string is evaluated.

Example:

The following script evaluates the expression and writes the answer to the console. The answer should be 400.

```
. . .
Set gvVar1 = 100; Set gvVar2 = 200;
Set gvVal = eUtl.Evaluate{'2*gvVar1 + gvVar2'};
Write console 'Value -'+gvVal;
. . .
```

FCode{}

The FCode method gets an FCode by name and returns the integer value which can be used later.

Format:

```
Set fcode = eUtl.FCode{string};
```

Table 52: FCode{} Options

Option Name	Option Description
string	A string giving the name of the F-code identifier.
fcode	The Integer fcode value corresponding the the name of the identifier.

Example:

The following script gets the integer F-code value from the name and later on uses it to execute an f-code.

```
Set gvFCode = eUtl.FCode{'HighPgf'};
. . .
Exec FC Expr(gvFCode);
. . .
```

Round{}

The Round function takes a real number and rounds it (up or down) according to the number of decimal places specified (or the default value).

Format:

```
Set roundedNumber = eUtl.Round{realNumber[,decimalPlaces]};
```

Table 53: Round{} Options

Option Name	Option Description
realNumber	The source number.
decimalPlaces	The optional number of decimal places to use in rounding the number (the default value is 5).
roundedNumber	The resulting rounded number.

Example:

The following script demonstrates the round function by rounding the same number with different decimal places specified.

```
Set vTest1 = '7.194379';
Loop InitVal(0) Incr(1) LoopVar(i) While(i<6)
   Set vResult = eUtl.Round{vTest1,i};
   Write Console 'Orig-'+vTest1+'  RoundBy-'+i+' Result-'+vResult;
EndLoop
. . .
The output should be the following:
Orig-7.194379  RoundBy-0 Result-7.000000
Orig-7.194379  RoundBy-1 Result-7.200000
Orig-7.194379  RoundBy-2 Result-7.190000
Orig-7.194379  RoundBy-3 Result-7.194000
Orig-7.194379  RoundBy-4 Result-7.194400
Orig-7.194379  RoundBy-5 Result-7.194380
. . .
```

Truncate{}

The Truncate function takes a real number and truncates it according to the number of decimal places specified (or the default value).

Format:

```
Set truncatedNumber = eUtl.Truncate{realNumber[,decimalPlaces]};
```

Table 54: Truncate{} Options

Option Name	Option Description
realNumber	The source number.
decimalPlaces	The optional number of decimal places to use in rounding the number (the default value is 5).
truncatedNumber	The resulting truncated number.

Example:
The following script demonstrates the truncate function by rounding the same number with different decimal places specified.

```
Set vTest1 = '7.194379';
Loop InitVal(1) Incr(1) LoopVar(i) While(i<6)
   Set vResult = eUtl.Truncate{vTest1,i};
   Write Console 'Orig-'+vTest1+'  TruncateBy-'+i+' Result-'+vResult;
EndLoop
. . .
The output should be the following:
Orig-7.194379  TruncateBy-1 Result-7.100000
Orig-7.194379  TruncateBy-2 Result-7.190000
Orig-7.194379  TruncateBy-3 Result-7.194000
Orig-7.194379  TruncateBy-4 Result-7.194300
Orig-7.194379  TruncateBy-5 Result-7.194370
. . .
```

GetFileNameFromSearchPath{}

The GetFileNameFromSearchPath function makes a full path name from a relative file name (without a path), scanning the ElmScript Search path (defined in the ElmScript->Options dialog, search path panel).

Format:

```
Set fullPath = eUtl.GetFileNameFromSearchPath{relFileName};
```

Table 55: GetFileNameFromSearchPath{} Options

Option Name	Option Description
relFileName	The relative file name (no path).
fullPath	The full path name returned.

Example:
The following script starts with the relative file name ('FRONT.BMP') and searches the Search path for a matching file.

```
Set gvGraphicFileName = eUtl.GetFileNameFromSearchPath{'FRONT.BMP'};
```

GetPropertyValue{}

The GetPropertyValue function retrieves the value of the specified property. This can be used when the property name is known only when the script is running.

Format:

```
Set propValue = eUtl.GetPropertyValue{dataValue, PropertyName};
```

This is equivalent to the following:

```
Set propVal=dataValue.PropertyName;
```

Table 56: GetPropertyValue{} Options

Option Name	Option Description
dataValue	The source object/value.
PropertyName	A String value containing the name of the property.
propVal	The value of the property.

Example 1:
The following script gets the name of the active document.

```
// This is equivalent to Set docName=ActiveDoc.Name;
Set docName = eUtl.GetPropertyValue{ActiveDoc,'Name'};
```

Example 2:
The following subroutine gets the specified properties of the active document and writes them to the FrameMaker console.

```
//
Sub WriteOutSelectedProps using pvPropNames
   Local lvPropName lvPropVal;
   Loop ForEach(Member) In(pvPropNames) LoopVal(lvPropName)
      Set lvPropVal = eUtl.GetPropertyValue{ActiveDoc,lvPropName};
      Write Console lvPropName+'='+lvPropVal;
   EndLoop
EndSub
```

SetPropertyValue{}

The SetPropertyValue function sets the value of the specified property. This can be used when the property name is known only when the script is running.

Format:

```
eUtl.SetPropertyValue{dataValue, PropertyName, PropVal};
```

This is equivalent to the following:

```
Set dataValue.PropertyName=PropVal;
```

Table 57: SetPropertyValue{} Options

Option Name	Option Description
dataValue	The source object/value.
PropertyName	A String value containing the name of the property.
propVal	The value of the property.

Example:
The following script sets the AutoChangeBars property of the active document to True.

```
// This is equivalent to Set ActiveDoc.AutoChangeBars=True;
eUtl.SetPropertyValue{ActiveDoc,'AutoChangeBars',True};
```

FormatString{}

The FormatString method is a function that builds and returns a string value. You supply a format string, which defined how the output string should appear. You also supply zero or more data values, one for each format specification in the format string.

Format:

```
Set strVar = eUtl.FormatString{fmt,val1[,val2]...[,valN]};
```

Table 58: FormatString{} Options

Option Name	Option Description
fmt	The format of the data. This is a string value that describes the output string. This method formats and stores a series of characters and values in output string. Each argument (if any) is converted and output according to the corresponding format specification in the string. This format string consists of ordinary characters and format specifications. These format specifications are patterned after the standard used in the C/C++ printf function. Each format specification starts with a % character and terminates with one of the `type` characters. In between you may optionally specify some flags and some width and precision information. If you wish to put a '%' into the output stream, then you need to put two '%' characters in a row ('%%'). A format specification has the following form: **%[flags][width[.precision]Type** The **flags** may be one or more of the values in the flags table. See "Format Flag Characters" on page 271. The **width** argument is a nonnegative decimal integer controlling the minimum number of characters output. If the number of characters in the output value is less than the specified width, blanks are added to the left or the right of the values — depending on whether the – flag (for left alignment) is specified — until the minimum width is reached. The width specification never causes a value to be truncated. If the number of characters in the output value is greater than the specified width, or if width is not given, all characters of the value are printed (subject to the precision specification). The **precision** is an optional number that specifies the maximum number of characters printed for all or part of the output field, or the minimum number of digits printed for integer values The Type is a one character value that specifies the data type. See "Format Types" on page 272.
valI	Any data value.
strVar	The variable to hold the returned output string.

Table 59: Format Flag Characters

Flag	Description
–	Left align the result within the given field. The default value is right align.
+	Prefix the output value with a sign (+ or –). The default is that the sign appears only for negative signed values (–).
0	If width is prefixed with 0, zeros are added until the minimum width is reached. If 0 and – appear, the 0 is ignored. The default is no padding.

Table 59: Format Flag Characters

Flag	Description
blank(' ')	Prefix the output value with a blank if the output value is signed and positive; the blank is ignored if both the blank and + flags appear. The default is no blank appears.
#	When used with the o, x, or X type, the # flag prefixes any nonzero output value with 0, 0x, or 0X, respectively. When used with the e, E, or f type, the # flag forces the output value to contain a decimal point in all cases. The default is that the decimal point appears only if digits follow it. When used with the g or G format, the # flag forces the output value to contain a decimal point in all cases and prevents the truncation of trailing zeros. The default is that the decimal point appears only if digits follow it. Trailing zeros are truncated. The is ignored when used with c, d, i, u, or s.

Table 60: Format Types

Type	Description
c	A single character is output.
d	A decimal number is output.
o	An octal number is output.
x or X	A hexadecimal number is output. If the X is used, the output will have capital letters.
f	A floating point number in the form [-]dddd.dddd
e or E	A floating point number in the form [-]d.dddd**e**[sign]ddd, where d is a single decimal digit, dddd is one or more decimal digits, ddd is exactly three decimal digits and he sign is + or -. If E is specified, then the E will be in the output.
g or G	A floating point number in f or e format, whichever is more compact for the given value and precision. The e format is used only when the exponent of the value is less than –4 or greater than or equal to the precision argument. Trailing zeros are truncated, and the decimal point appears only if one or more digits follow it.
s	This specifies a single-byte–character string. Characters are output up to the end of the string or until the precision value is reached.

Example:
The following script writes various data items to the FrameMaker console.

```
. . .
Write Console eUtl.FormatString{'VersMajor=%d VersMinor=%d Msg=|%10s|',
      FslVersionMajor, fslVersionMinor, 'Hello World'};
Write Console eUtl.FormatString{'VersMajor=%02d VersMinor=%02d Msg=|%-10s|',
      FslVersionMajor, fslVersionMinor, 'Hello World'};
Write Console eUtl.FormatString{'VersMajor=%d VersMinor=%d Msg=|%20s|',
      FslVersionMajor, fslVersionMinor, 'Hello World'};
Write Console eUtl.FormatString{'VersMajor=%02d VersMinor=%02d Msg=|%-20s|',
      FslVersionMajor, fslVersionMinor, 'Hello World'};
Write Console eUtl.FormatString{'VersMajor=%d VersMinor=%d Msg=|%20.10s|',
      FslVersionMajor, fslVersionMinor, 'Hello World'};
Write Console eUtl.FormatString{'VersMajor=%02d VersMinor=%02d Msg=|%-20.10s|',
      FslVersionMajor, fslVersionMinor, 'Hello World'};

Set gvReal=6667.984;
Write Console eUtl.FormatString{
       'RealE=%8.2E Reale=%8.2e RealG=%8.2G Realg=%8.2g Realf=%8.2f',
          gvReal, gvReal, gvReal, gvReal, gvReal};

Set gvInt=45;
Write Console eUtl.FormatString{'Intd=%d Into=%o Intx=%x IntX=%X',
      gvInt, gvInt, gvInt, gvInt};

Write Console eUtl.FormatString{'Intd=%07d Into=%07o Intx=%07x IntX=%07X',
      gvInt, gvInt, gvInt, gvInt};

Write Console eUtl.FormatString{'StrC=%c StrX=%2X Strd=%0d ',
      ProductName, ProductName, ProductName};

. . .
```

The output on the FrameMaker console is the following:

```
VersMajor=5 VersMinor=0 Msg=|Hello World|
VersMajor=05 VersMinor=00 Msg=|Hello World|
VersMajor=5 VersMinor=0 Msg=|         Hello World|
VersMajor=05 VersMinor=00 Msg=|Hello World         |
VersMajor=5 VersMinor=0 Msg=|          Hello Worl|
VersMajor=05 VersMinor=00 Msg=|Hello Worl          |
RealE=6.67E+003 Reale=6.67e+003 RealG=6.7E+003 Realg=6.7e+003 Realf= 6667.98
Intd=45 Into=55 Intx=2d IntX=2D
Intd=0000045 Into=0000055 Intx=000002d IntX=000002D
StrC=F StrX=46 Strd=70
```

MoveComponent{}

The MoveComponent method moves a book component within a book. **FM 9.0 or greater**

Format:

```
Set rc = eUtl.MoveComponent{bookObject,cmpObject,action};
```

Table 61: MoveComponent{} Options

Option Name	Option Description
bookObject	The book object.
cmpObject	The book component object in the specified book.
action	An integer action value. This should be one of the following values: ComponentMoveup ComponentMovedown ComponentPromote ComponentDemote
rc	The return code of the function. This should be 0 for success.

Example:
The following script moves up a book component.

```
Set gvBook=ActiveBook;
Set gvComp=gvBook.FirstComponentInBook;
Set gvComp2=gvComp.NextComponentInBook;
Set rc=eUtl.MoveComponent{gvBook,gvComp2,ComponentMoveUp};
. . .
```

UpdateXRef{}

The UpdateXRef method updates a specified cross-reference in the document. **FM 9.0 or greater**

Format:

```
Set rc = eUtl.UpdateXRef{srcDocObject,destDocObject,xrefObject};
```

Table 62: UpdateXRef{} Options

Option Name	Option Description
srcDocObject	The Object of the document that the cross-reference references
destDocObject	The Object of the document that contains the cross-reference.
xrefObject	The Object of the cross-reference to be updated.
rc	The return code of the function. This should be 0 for success.

Example:
The following script updates the cross-reference referring to the same/another document.

```
Set gvDoc=ActiveDoc;
Set gvXRef=gvDoc.FirstXRrefInDoc;
Set gvFilename=gvXRef.XRefFile;
If gvFilename.count<1
   Set gvSrcDoc=gvDoc;
Else
   Open Document File(gvFilename) NewVar(gvSrcDoc);
EndIf
Set rc=eUtl.UpdateXRef{gvDoc, gvSrcDoc, gvXRef};

. . .
```

ApplyAttributeExpr{}

The ApplyAttributeExpr method applies an attribute expression to the document to perform the attribute based filtering. **FM 9.0 or greater**

Format:

```
Set rc = eUtl.ApplyAttributeExpr{attrExprObject};
```

Table 63: ApplyAttributeExpr{} Options

Option Name	Option Description
attrExprObject	The Object of the attribute expression.
rc	The return code of the function. This should be 0 for success.

Example:
The following script applies the expression name 'WebOutput' to the document.

```
Get Object Type(AttrCondExpr) Name('WebOutput') NewVar(gvAttrExprObject);
Set rc=eUtl.ApplyAttributeExpr{gvAttrExprObject};

. . .
```

ForceUpdateXRefAltText{}

The ForceUpdateXRefAltText method forces the cross-ref update alt Text. FM10 or greater

Format:

```
Set rc = eUtl.ForceUpdateXrefAltText(xrefObject};
```

Table 64: ForceUpdateXrefAltText{} Options

Option Name	Option Description
xrefObject	The cross reference Object.
rc	The return code of the function. This should be 0 for success.

ConvertTiToText{}

The ConvertTiToText method converts the text inset to text. FM10 or greater

Format:

```
Set rc = eUtl.ConvertTiToText(tiObject};
```

Table 65: ConvertTiToText{} Options

Option Name	Option Description
`tiObject`	The Inset Object.
`rc`	The return code of the function. This should be 0 for success.

. . .

TrackChangesAcceptAll{}

The TrackChangesAcceptAll method accepts all the tracking changes. FM10 or greater

Format:

```
Set rc = eUtl.TrackChangesAcceptAll(docObject};
```

Table 66: TrackChangesAcceptAll{} Options

Option Name	Option Description
`docObject`	The Document Object.
`rc`	The return code of the function. This should be 0 for success.

. . .

TrackChangesRejectAll{}

The TrackChangesRejectAll method rejects all the tracking changes. FM10 or greater

Format:

```
Set rc = eUtl.TrackChangesRejectAll(docObject};
```

Table 67: TrackChangesRejectAll{} Options

Option Name	Option Description
docObject	The Document Object.
rc	The return code of the function. This should be 0 for success.

. . .

UpdateMenus{}

The UpdateMenus method updates the menus. FM10 or greater

Format:

```
Set rc = eUtl.UpdateMenus();
```

Table 68: UpdateMenus{} Options

Option Name	Option Description
rc	The return code of the function. This should be 0 for success.

. . .

NewBook{}

The NewBook method creates a new empty book. FM10 or greater

Format:

```
Set bookObject = eUtl.NewBook();
```

Table 69: NewBook{} Options

Option Name	Option Description
bookObject	The book object returned.

. . .

DeleteUnusedFormats{}

The DeleteUnusedFormats method deletes unused formats. FM10 or greater

Format:

```
Set rc = eUtl.DeleteUnusedFormats(docObject,objectName};
```

Table 70: DeleteUnusedFormats{} Options

Option Name	Option Description
docObject	The Document Object.
objectName	The name (String) of the format object type.
rc	The return code of the function. This should be 0 for success.

. . .

GetConditionalExpression{}

The GetConditionalExpression method returns the expression for the given conditional tag. **FM 2015 or greater**

Format:

```
Set expr = eUtl.GetConditionalExpression{bookObject, condTag};
```

Table 71: GetConditionalExpression{} Options

Option Name	Option Description
bookObject	The Book Object.
condtag	The condition tag.
expr	The returned conditional expression

GetWorkSpace{}

The GetWorkSpace method returns the name of the current workspace. **FM 2017 or greater**

Format:

```
Set wrkName = eUtl.GetWorkSpace{};
```

Table 72: GetWorkSpace{} Options

Option Name	Option Description
wrkName	The returned current workspace name.

SetWorkSpace{}

The SetWorkSpace method sets the name of the current workspace. **FM 2017 or greater**

Format:

```
Set ret = eUtl.SetWorkSpace{wrkName};
```

Table 73: SetWorkSpace{} Options

Option Name	Option Description
wrkName	The workspace name to set.
ret	The return code.

ApplyFitToFrame{}

The SetWorkSpace method sets the name of the current workspace. **FM 2017 or greater**

Format:

```
Set ret = eUtl.ApplyFitToFrame{opCode};
```

Table 74: ApplyFitToFrame{} Options

Option Name	Option Description
opcode	The code: **1 FitFrameToFrame.** **2 FitFrameToImageProportional** **3 FitImageToFrame**
ret	The return code.

Chapter 18

EActiveXObject

The EActiveXObject object represents an external COM dispatch object. These objects are not part of ElmScript. They are written by 3rd party vendors and reside not only outside of ElmScript but outside of FrameMaker as well. Vendors write them as an interface to some functionality. For instance, Microsoft (the developer of the COM technology) has written many COM objects mostly as a way to access their own products, such Microsoft Word, Microsoft Excel, etc. Microsoft has also developed an object (MSXML) that processes XML files. Other vendors have also developed COM objects.

Each COM object is identified by a ProgId. This is a text string that uniquely identifies the object. These are defined in the Microsoft Windows registry. In ElmScript, when you created an EActiveXObject object, you must provide this name. This name is assigned by the creator of the object type. For example, the name of the MSXML object from Microsoft is called 'Msxml2.DOMDocument'.

Like the other EslObjects, you interact with EActiveXObject objects by accessing its properties and calling its methods after creating an instance of them with the **New** command. When you are done with them, you use the Delete Object command to remove the instance of them. Unlike the other EslObjects, EActiveXObjects have no unique properties (except for ObjectName) or methods. The properties and methods for an EActiveXObject object is determined by the particular object created. In other words, the Microsoft Word COM Object has its own defined properties and methods. The Microsoft Excel COM Object has its own set of properties and methods. Other COM types have their own set of properties and methods.

> **IMPORTANT:** The EActiveXObject is an advanced feature of ElmScript and should only be used by experienced programmers and scripters.

Creating the Object

You create an EActiveXObject object by using the **New** command and providing the ProgId of the object. The ProgId is the descriptive name of the object. The vendor will provide this. The following is an example of creating the Dom object of the MsXml control.

```
New EActiveXObject NewVar(gvMyXmlDom) ProgId('Msxml2.DOMDocument');
```

The Dom object represents an XML document. Note: ElmScript does not support the *servername* option which is available in some systems.

See the demo script XmlToFmConsoleDemo.fsl for an example of creating and using this object.

Deleting the object

To delete an instance of this object use the DELETE Object command. Delete the object when you do not need it anymore.

Delete Object

The Delete method deletes the `EActiveXObject` object.

Format:

```
Delete Object(gvMyComObj);
```

EActiveXObject Properties

The properties of EActiveXObject objects are defined in their respective documentation. You access the properties using the familiar dot notation (e.g. objectVar.PropertyName).

EActiveXObject Methods

The methods of EActiveXObject objects are defined in their respective documentation. The preferred way to run EActiveXObject methods is the use the helper function EslRunMethod. The syntax for this is as follows:

```
Set returnValue = objectVar.EslRunMethod{'methodname'[,arg1] ... [,argN]};
```

> **IMPORTANT:** The arguments must be in the correct order. The current implementation of Active X methods does not support named arguments.

Using EActiveXObject objects

It is beyond the scope of this document to try and describe the properties and methods of all these objects. These are published by the vendor and subject to change as they update their products. There are many examples of using ActiveX objects on the internet. The syntax for a ElmScript EActiveXObject is similar to the equivalent syntax for JavaScript. If you adjust the JavaScript examples on the web to fit the ElmScript syntax, it should work most of the time, at least for the types of data that ElmScript supports.

Looking for Information

Use a search engine to look for ActiveX related technologies. For example, search for "Msxml2.DOMDocument" for information on the MsXml object. Search for "Word.Application" for information on the Microsoft Word object and "Excel.Application" for information on Microsoft Excel.

> **IMPORTANT:** ElmScript does not support all the options and data types of all COM objects. They must be taken on a case by case basis.

Chapter 19

OE/EVM

Like JavaScript[1], ElmScript uses object types (classes) but cannot create them. Classes are usually provided by some enclosing program. For example, JavaScript in a web browser uses the DOM (Document Object Model) provided by the brower itself, usually dealing with updating the web page. ElmScript uses a DOM based on FrameMaker objects. ElmScript also uses objects provided by the Ms Windows OS, including ODBC and systems functions amoung others.With ElmScript version 6, it is now possible to create and use object types. This is done using an interface into the OE/Evm development system.

OE is an object oriented computer language. The Evm is a virtual machine (ElmSoft Virtual Machine) which can compile, load and run programs/files written in OE. In fact, it can also run specially formatted dlls written in C/C++, which can create classes and methods/properties. ElmScript 6 has an interface that can create EVMs, load OE (or C/C++) files into it in order to create (and/or load) classes and to run methods. When ElmScript 6 creates an Evm, it also adds a DOM (a set of classes) to interact directly with FrameMaker objects.

Detailed information about creating programs(and class files) in OE/EVM is available at the ElmSoft downloads website (www.ElmSoftOnline.com/downloads).

> **IMPORTANT:** Does this mean that you have to learn a new language to use this functionality? Actually, the answer is no.The OE/EVM system comes with many useful classes (read and write zip files, regular expressions (Pcre), (X)Html input/output, EPub output). These classes can be accessed via ElmScript. You can retrieve and/or set the values of properties and run methods in Evm objects by using the standard ElmScript commands and syntax that you already know. However, if you want to build your own classes then you will have to know how to program with OE.

EvmVM object

The EvmVm object represents an instance of an Evm (ElmSoft Virtual Machine). There is a default Evm available in a global variable called EDefEvmVM. Since it is rare to need more than one Evm in a ElmScript script, using the EDefEvmVM global variable is a convenient short cut.

Like the other EslObjects, you interact with EvmVM objects by accessing its properties and calling its methods. Unlike the other EslObjects, EvmVM have no unique properties (except for ObjectName) or methods. The properties and methods for an EvmVM object is determined by what classes (or methods or properties) have been imported.

Creating the Object

If you need more than one Evm, you create another EvmVM object by using the **New** command. It will automatically locate and connect to the Evm system, if it has been installed.

```
New EvmVM NewVar(gvEvm);
```

. . . . . .

1. To be fair, javascript has the ability to create minimal objects using the this keyword.

There are several sample/demo scripts that illustrate using the Evm. Many of the new object types available are classes in the Evm.

Deleting the object

If you wish to delete an instance of this object use the DELETE Object command. With ElmScript 6, you do not have to delete esl objects anymore. Once they go out of scope they will be automatically deleted for you.

Delete Object

The Delete method deletes the `EvmVM` object.

Format:

```
Delete Object(gvEvm);
```

EvmVM Properties

The properties available in EvmVM objects are determined by what class files has been imported into the Evm. You access the properties using the familiar dot notation (e.g. EDefEvmVM.PropertyName). At first, the properties available are all the built-in class objects in the dataspace. These include the standard objects (e.g. Object, Integer, String, etc.) as well as the FrameMaker objects (FmDoc, FmBook, etc.). When you run the built-in import method, you can import classes into the Evm and make other class properites available.

EvmVM Methods

The methods available in EVmVM objects are determined by what class files has been imported into the Evm. There is one exception to this and that is the RunEvmScript method: You can use this to run OE programs from a disk file.

```
Set returnValue = EDefEvmVM.RunEvmScript{'filename'};
```

Using EvmVM objects

To get the full advantage of the OE/EVM system, you would need to learn to program using this language. However, if you want to primarily use ElmScript and access previously created Evm objects (by ElmSoft or third party providers), most of the time you would do the following steps.

1. Import one or more class files into the Evm.
   Once this is done you can access any **`class`** methods that are available from the imported classes. For example, if you wanted to use one of the class methods in the `System` class to compute a UUID value, you could do the following:

   ```
   EDefEvmVM.import{'System'};
   Set gvUUid=EDefEvmVM.System.computeUUID{};
   ```

   Note the following:
   - You only have to import a class file once for an Evm.
   - 'System' identifies the class file, which is in the standard library. When imported it also happens to create a class called System. This is a coincidence. The class file name and the class(es) it creates do not have to be named the same. In fact, in most cases this will not occur.

- Identifiers that go to the Evm are case sensitive (import,System and computeUUID must be the case shown.
- Since computeUUID is a class method (in the System class), it means that you do not have to create an instance of the class System in order to use it.
- The computeUUID method will return a standard ElmScript string value.
- The computeUUID method has empty braces ({}). This is necessary for ElmScript

2. Once classes have been imported, you can use their class methods (as in the above example).You can also create instances from the classes using the `make` method.

```
EDefEvmVM.import{'TempFiles'};
var vTempFileMgr=ETempFileMgr.make{'MyCompany','MyProject'}; // Create an instance
...
var vTempFileName=vTempFileMgr.getTempFileName{};
...
vTempFileMgr.deleteAllTempFiles{};
```

This set of commands performs the following:

- It imports the TempFiles class file, which creates the ETempFileMgr class.
- It creates an instance of the ETempFileMgr class and puts in into the vTempFileMgr local variable.
- OE uses the make method of a class object to create an instance of that object type. Other object oriented systems (including ElmScript) generally use the New keyword to do this.
- It calls the instance method (getTempFileName) to create a new unique temporary file name.
- It calls the deleteAllTempFiles instance method to delete all the temp files created up to that point.

3. Since classes in OE are just objects, you can get them as properties and use them later. Here is an example of using the regular expressions using the Pcre object.

```
Set gvRc=EDefEvmVM.import{'Pcre'};
Set gvPcre=EDefEvmVM.Pcre;  // Get the Pcre class object,
                            //  to make it easier to run some class methods
Set gvRegExp='(the quick) (brown fox)';  // Make a small regular expression string
// Build a string to search
Set gvSearchStr='The quick brown fox'+CHARCR+CHARLF;
Set gvSearchStr=gvSearchStr+'The quick brown FOX'+CHARCR+CHARLF;
Set gvSearchStr=gvSearchStr+'About the quick brown fox doing the foxtrot?' +
                            CHARCR + CHARLF;
Set gvSearchStr=gvSearchStr+'What do you know about THE QUICK BROWN FOX?' +
                            CHARCR+CHARLF;

// Run the subStr method to find the first matching substring in the search string.
// It uses the case insensitive and multi-line options. It returns the string itself.
Set gvMatchList=gvPcre.search{gvRegExp,gvSearchStr,'miZg'};
Write Console 'PCRE TEST--Pcre search-'+gvMatchList.toS{};
If gvMatchList
   // Replace each match with the replace string where the 2 subpatterns are
   // switched and a dash(-) placed between them.
   Set gvNewStr=gvMatchList.replace{gvSearchStr,'\2-\1'};
   Write Console 'New String='+gvNewStr);
EndIf;
```

This example illustrates many ways to interface with the Evm from ElmScript.

- As before, it imports the class file (Pcre, standard library) needed to create the classes used for regular expressions. The main class is the Pcre class, but there are other classes created as well.

- The second line gets the Pcre property from the Evm. This is the class object. We don't have to get this property here. We can get later when you ready to use it. But since it will probably used many times (if we have a script that does a lot of regular expressions) we might as well get it now.
- The gvPcre object returned is an `EvmObject` (see below) EslObject.
- The gvPcre.search method performs a regular expression search. It returns (if successful) an instance of the class Pcre.PcreMatchList. This is also an EvmObject type of EslObject.
- The Pcre.PcreMatchList object has a replace method which returns a string replacing the original string with the subpattern inserted.

4. Summary
   - The EvmVM object is the starting point for any interface.
   - The EvmVM object can return object handles (EvmObject) which can be used to access properties and methods. These are objects in the Evm that have no equivalent in ElmScript.
   - Although identifiers in ElmScript are case insenitive, identifiers in the Evm are case sensitive. Make sure to use the correct casing when accessing properties and/or methods in the Evm.
   - Methods with no arguments still must have empty braces ({}) to identify it as a method.

EvmObject object

When you get a property or run a method, the Evm returns a value. For standard data items (such as Integer, String, Number values), the Evm converts the object into a ElmScript data type. However, when a property (or the results of a method call) returns a non-standard object (class or instance of some non-built-in class), the object returned is a handle to that class. That handle is represented in ElmScript by an EvmObject eslobject. You can use this object to get and set properties and to run methods of that object.

You cannot create or delete an object of this type. They are only returned from a get property or method call.

ElmScript vs OE

When you use Evm objects in ElmScript you use the ElmScript syntax for the commands and functions. You don't have to know the OE syntax unless you are writing classes in OE. However, there are some things to remember when interfacing with Evm objects from ElmScript.

- OE names are case sensitive.

  When you get or set properties or call methods, the name you use has to be the same case as described in the class documentation. While ElmScript names are case insensitive and you can use these case insensitive names for ElmScript variable names that hold Evm Objects, any name identifying a property or method insdie the Evm virtual machine must be the correct case.

- In ElmScript, the comparison equality operator is a single equal character (=) character. OE uses the more standard double equal (==).

  In ElmScript, you can do the following and it will display the message.

```
Set gvTest=0;
If gvTest=0
   Display 'Test is True';
EndIf
```

  In OE, if you can do the following and it will *not* display the message, because even inside an if statement, a single equal character is an assignment operator. It will assign a value of zero to the vTest variable, then return a false condition.

```
var vTest=0;
if vTest=0       // Single equal sign, assignment operator
   alert('Test is True');
end
```

  The correct way to do a comparison in OE is the following.

```
var vTest=0;
if vTest==0      // double equal sign, comparison operator
   alert('Test is True');
end
```

  This is a potentially error producing situation that has plagued many a C, C++, Java and JavaScript programmer. However, it is the only way to distinguish an assignment from a comparison. ElmScript has special syntax the allows this duality at the cost of more consistent expressions.

- Arrays indexes start with 0 (zero) in OE instead.

  Standard arrays in ElmScript start with 1 as the first member. OE indexes start with 0 as the first member. You can access members of arrays using the standard bracket notation, as follows.

```
Set gvArr=EDefEvmVM.Array.make; // Create an Evm array
Set gvRc=gvArr.push{2,4,6,9,10,19,23}; // Add integer members to the array
Set gvFirstMember=gvArr[0]; // Get the value in the first member
Set gvArr[3]=9999; // Set the value of the 4th member
```

- Callback functions.

  You can call methods in Evm as described in the examples above. You can also have methods in the Evm call ElmScript methods (as callback functions). This is done in two steps. The first is to create a ElmScript SubVar object that creates a reference to the function that you wish a method in the Evm to call. Then you set this subvar object to a property (or use it as an argument in a method call) sending the subvar to the Evm. It can then be called by the Evm (using the run method). Of course the object in the Evm must be programmed to call this method when desired.