Open Object Rexx™

Windows Extensions Reference

Version 4.1.0 Edition
Draft - SVN Rev 6346
November 2010

W. David Ashley
Rony G. Flatscher
Mark Hessling
Rick McGuire
Mark Miesfeld
Lee Peedin
Jon Wolfers

Open Object Rexx™: Windows Extensions Reference

by
W. David Ashley
Rony G. Flatscher
Mark Hessling
Rick McGuire
Mark Miesfeld
Lee Peedin
Jon Wolfers

Version 4.1.0 Edition
Published November 2010

Thanks to Julian Choy for the ooRexx logo design.

Table of Contents

List of Tables

About This Book

This book describes extensions to the Open Object Rexx Interpreter that are specific to the Windows operating system. The extensions are in three main categories.

The first category is a number of classes implemented in a library package, `winSystm.cls`. These classes are used to interact with Windows system objects like the event log and the clipboard. The second category is OLE Automation . The last category is the Windows Scripting Host engine.

These extensions are currently only available on Windows. The Windows Scripting Host and OLE Automation can only be implemented on Windows. Some of the classes, such as the WindowsEventLog and the WindowsRegistry classes must be, by their nature, Windows specific. Some of the other classes, such as the MenuObject or WindowObject classes could certainly be enhanced to be cross-platform. However, at this time there are no plans to do so.

This book is intended for people who plan to develop applications using ooRexx and one or more of the Windows specific classes. In general no special knowledge of Windows programming is needed to use the Windows extensions. Therefore this book is applicable for users ranging in experience from the novice ooRexx programmer, to the experienced application developer.

This book is a reference rather than a tutorial. It assumes the reader has some exposure to object-oriented programming concepts and Rexx programming.

The use and syntax of all the classes and their methods is covered in this book. A brief overview of OLE Automation and the Windows Scripting Host Engige is given. Many of the descriptions of class methods also include example code snippets.

1. Related Information

See also: *Open Object Rexx: Reference*

2. How to Read the Syntax Diagrams

Throughout this book, syntax is described using the structure defined below.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The >>--- symbol indicates the beginning of a statement.

  The ---> symbol indicates that the statement syntax is continued on the next line.

  The >--- symbol indicates that a statement is continued from the previous line.

  The --->< symbol indicates the end of a statement.

  Diagrams of syntactical units other than complete statements start with the >--- symbol and end with the ---> symbol.

- Required items appear on the horizontal line (the main path).

```
>>-STATEMENT--required_item-----------------------------------><
```

- Optional items appear below the main path.

```
>>-STATEMENT--+---------------+--------------------------------><
              +-optional_item-+
```

- If you can choose from two or more items, they appear vertically, in a stack. If you must choose one of the items, one item of the stack appears on the main path.

```
>>-STATEMENT--+-required_choice1-+----------------------------><
              +-required_choice2-+
```

- If choosing one of the items is optional, the entire stack appears below the main path.

```
>>-STATEMENT--+------------------+----------------------------><
              +-optional_choice1-+
              +-optional_choice2-+
```

- If one of the items is the default, it appears above the main path and the remaining choices are shown below.

```
              +-default_choice--+
>>-STATEMENT--+-----------------+----------------------------><
              +-optional_choice-+
              +-optional_choice-+
```

- An arrow returning to the left above the main line indicates an item that can be repeated.

```
              +-----------------+
              V                 |
>>-STATEMENT----repeatable_item-+----------------------------><
```

  A repeat arrow above a stack indicates that you can repeat the items in the stack.

- A set of vertical bars around an item indicates that the item is a fragment, a part of the syntax diagram that appears in greater detail below the main diagram.

```
>>-STATEMENT--| fragment |-----------------------------------><
```

  fragment:

```
|--expansion_provides_greater_detail----------------------------|
```

- Language keywords appear in uppercase (for example, `SAY`). They must be spelled exactly as shown but you can type them in upper, lower, or mixed case. Variables appear in all lowercase letters (for example, `parmx`). They represent user-supplied names or values.
- Class and method names appear in mixed case (for example, `.Object~new`). They must be spelled exactly as shown but you can type them in upper, lower, or mixed case.
- If punctuation marks, parentheses, arithmetic operators, or such symbols are shown, you must enter them as part of the syntax.

The following example shows how the syntax is described:

```
        +-,------+
        V        |
>>-MAX(----number-+--)-----------------------------------------><
```

3. A Note About Program Examples in this Document

The program examples in this document are rendered in a mono-spaced font that is not completely compatible for cut-and-paste functionality. Pasting text into an editor could result in some characters outside of the standard ASCII character set. Specifically, single-quote and double-quote characters are sometimes converted incorrectly when pasted into an editor.

4. Getting Help

The Open Object Rexx Project has a number of methods to obtain help for ooRexx. These methods, in no particular order of preference, are listed below.

4.1. The Rexx Language Association Mailing List

The *Rexx Language Association* (http://www.rexxla.org/) maintains a mailing list for its members. This mailing list is only available to RexxLA members thus you will need to join RexxLA in order to get on the list. The dues for RexxLA membership are small and are charged on a yearly basis. For details on joining RexxLA please refer to the *RexxLA Home Page* (http://rexxla.org/) or the *RexxLA Membership Application* (http://www.rexxla.org/rexxla/join.html) page.

4.2. The Open Object Rexx SourceForge Site

The Open Object Rexx Project (http://www.oorexx.org/) utilizes *SourceForge* (http://sourceforge.net/) to house the *ooRexx Project* (http://sourceforge.net/projects/oorexx) source repositories, mailing lists and other project features. Here is a list of some of the most useful facilities.

The ooRexx Forums

The ooRexx project maintains a set of forums that anyone may contribute to or monitor. They are located on the *ooRexx Forums* (http://sourceforge.net/forum/?group_id=119701) page. There are currently three forums available: Help, Developers and Open Discussion. In addition, you can monitor the forums via email.

The Developer Mailing List

You can subscribe to the oorexx-devel mailing list at *ooRexx Mailing List Subscriptions* (http://sourceforge.net/mail/?group_id=119701) page. This list is for discussing ooRexx project development activities and future interpreter enhancements. It also supports a historical archive of past messages.

The Users Mailing List

You can subscribe to the oorexx-users mailing list at *ooRexx Mailing List Subscriptions* (http://sourceforge.net/mail/?group_id=119701) page. This list is for discussing using ooRexx. It also supports a historical archive of past messages.

The Announcements Mailing List

You can subscribe to the oorexx-announce mailing list at *ooRexx Mailing List Subscriptions* (http://sourceforge.net/mail/?group_id=119701) page. This list is only used to announce significant ooRexx project events.

The Bug Mailing List

You can subscribe to the oorexx-bugs mailing list at *ooRexx Mailing List Subscriptions* (http://sourceforge.net/mail/?group_id=119701) page. This list is only used for monitoring changes to the ooRexx bug tracking system.

Bug Reports

You can create a bug report at *ooRexx Bug Report* (http://sourceforge.net/tracker/?group_id=119701&atid=684730) page. Please try to provide as much information in the bug report as possible so that the developers can determine the problem as quickly as possible. Sample programs that can reproduce your problem will make it easier to debug reported problems.

Request For Enhancement

You can suggest ooRexx features at the *ooRexx Feature Requests* (http://sourceforge.net/tracker/?group_id=119701&atid=684733) page.

Patch Reports

If you create an enhancement patch for ooRexx please post the patch using the *ooRexx Patch Report* (http://sourceforge.net/tracker/?group_id=119701&atid=684732) page. Please provide as much information in the patch report as possible so that the developers can evaluate the enhancement as quickly as possible.

Please do not post bug patches here, instead you should open a bug report and attach the patch to it.

4.3. comp.lang.rexx Newsgroup

The comp.lang.rexx (news:comp.lang.rexx) newsgroup is a good place to obtain help from many individuals within the Rexx community. You can obtain help on Open Object Rexx or on any number of other Rexx interpreters and tools.

Chapter 1. The WindowsProgramManager Class

The `WindowsProgramManager` class allows the programmer to interact with the Windows Program Manager. This class can be use to create program groups and shortcuts to access your programs.

The WindowsProgramManager class is defined in the file `winSystm.cls` To use this class in a program, place a `::requires` statement in the program file:

```
::requires "winSystm.cls"
```

A sample program `desktop.rex` is provided in the `samples\winsystem` directory.

Methods of the WindowsProgramManager class are:

Table 1-1. Methods Available to the WindowsProgramManager Class

Method...	...link
new (Class method)	init (Class method)
addDeskTopIcon	addDesktopIcon
addGroup	addGroup
addItem	addItem
addShortCut	addShortCut
deleteDesktopIcon	deleteDesktopIcon
deleteGroup	deleteGroup
deleteItem	deleteItem
showGroup	showGroup

1.1. new (Class method)

```
>>--new-----------------------------------------------------><
```

Creates an instance of the WindowsProgramManager class.

1.2. addDesktopIcon

```
>>-addDesktopIcon--(--name--,--program--+--------------------------------+-->
                                        |                    +-0------+ |
                                        +-,--+----------+--,--+--------+-+
                                             +-iconfile-+     +-iconnr-+

>--+-------------------------------------------------------------------------+-)-><
   |                +-"PERSONAL"-+                         +-"NORMAL"----+ |
   +-,-+---------+-,-+------------+-,-+------+-,-+--------+-,-+-------------+-+
```

```
+-workdir-+   +-"COMMON"---+   +-args-+   +-hotkey-+   +-"MAXIMIZED"-+
                                                       +-"MINIMIZED"-+
```

Adds a shortcut to the Windows desktop. A sample program `DESKICON.REX` is provided in the `ooRexx\SAMPLES` directory.

Arguments:

The arguments are:

name

The name of the shortcut, displayed below the icon.

program

The program file launched by the shortcut.

iconfile

The name of the icon used for the shortcut. If not specified, the icon of *program* is used.

iconnr

The number of the icon within the *iconfile*. The default is 0.

workdir

The working directory of the shortcut.

location

Either of the following locations:

"PERSONAL"

The shortcut is personal and displayed only on the desktop of the user.

"COMMON"

The shortcut is common to all users and displayed on the desktop of all users.

args

The arguments passed to the program that the shortcut refers to.

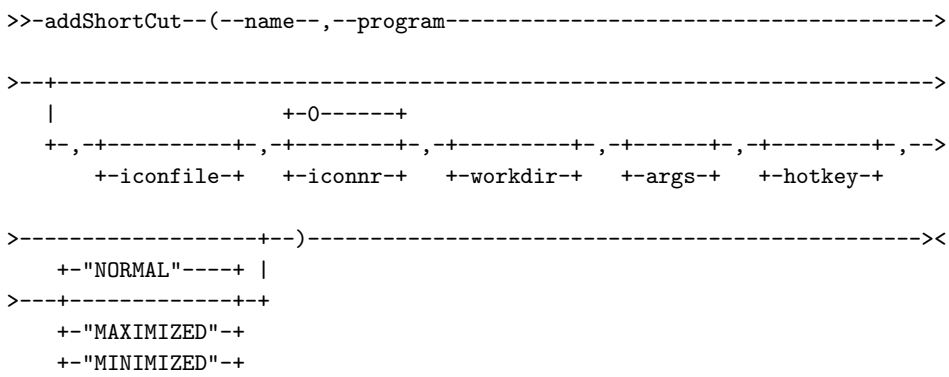
hotkey

The virtual key to be used as a hotkey to open the shortcut. For a list of the key names, see Symbolic Names for Virtual Keys.

run

Specifies one of the options listed in the syntax diagram. The default is "NORMAL".

1.3. addShortCut



Creates a shortcut within the specified folder.

Arguments:

The arguments are:

name

The full name of the shortcut.

program

The program file launched by the shortcut.

iconfile

The name of the icon used for the shortcut. If not specified, the icon of *program* is used.

iconnr

The number of the icon within the *iconfile*. The default is 0.

workdir

The working directory of the shortcut.

args

The arguments passed to the program that the shortcut refers to.

hotkey

The virtual key to be used as a hotkey to open the shortcut. For a list of the key names, see Symbolic Names for Virtual Keys.

run

Specifies one of the options listed in the syntax diagram. The default is "NORMAL".

Example:

The following example creates a shortcut named "My NotePad" to the Notepad editor within the directory c:\temp:

```
pm = .WindowsProgramManager~new
 if pm~InitCode \= 0 then exit
 pm~addShortCut("c:\temp\My Notepad","%SystemRoot%\system32\notepad.exe")
 ::requires "winsystm.cls"
```

1.4. addGroup

```
>>-addGroup(-group-)------------------------------------------><
```

Adds a program group to the Programs group of the desktop. If the group already exists, it is opened. The *group* argument specifies the name of the program group to be added. Example:

```
addGroup("Object Rexx Redbook")
```

Note: The name that you specify for the *group* argument must not contain any brackets or parenthesis. Otherwise, this method fails.

Return value:

0

The method was successful.

1

The method failed.

1.5. addItem

```
>>-addItem--(--shortcut,program--------------------------------->

>--+-------------------------------------------------------+----->
   +-,--+----------+--+-----------------------------------+-+
        +-iconfile-+  +-,--+------------+--+----------+-+
                           +-iconnumber-+  +-,workdir-+

>--)-------------------------------------------------------------><
```

Adds a shortcut to a program group. The shortcut is placed into the last group used with either AddGroup or ShowGroup. Example:

```
AddItem("OODialog Samples", ,
"rexx oodialog\samples\sample.rex", ,
```

```
"oodialog\samples\oodialog.ico")
```

Note: The name that you specify for the *group* argument must not contain characters that are not valid, such as brackets or parenthesis. Otherwise, this method fails. Some characters are changed, for example / to _.

Return value:

0

The method was successful.

1

The method failed.

1.6. deleteDesktopIcon

```
                                  +-"PERSONAL"-+
>>-deleteDesktopIcon--(--name--,---+------------+-)------------><
                                  +-"COMMON"---+
```

Deletes a shortcut from the Windows desktop that was previously created with AddDesktopIcon.

The arguments are:

name

The name of the shortcut to be deleted.

location

Either of the following locations:

"PERSONAL"

The shortcut was previously created with AddDesktopIcon and the location option "PERSONAL". This is the default.

"COMMON"

The shortcut was previously created with AddDesktopIcon and the location option "COMMON".

Return codes:

0

Shortcut deleted successfully.

2

Shortcut not found.

3

Path to shortcut not found.

5

Access denied or busy.

26

Not a DOS disk.

32

Sharing violation.

36

Sharing buffer exceeded.

87

Does not exist.

206

Shortcut name exceeds range error.

Note:: Return code 2 is also returned when a "PERSONAL" should be deleted that was previously created with "COMMON" and vice versa.

Example:

```
pm = .WindowsProgramManager~new
if pm~InitCode \= 0 then exit

rc = pm~deleteDesktopIcon("MyNotepad1", ,
     "%SystemRoot%\system32\notepad.exe")
if rc \= 0 then do
  say "Error deleting shortcut: My Notepad 1"
  exit
end

exit

::requires "winsystm.cls"
```

1.7. deleteGroup

```
>>-deleteGroup(-group-)----------------------------------------><
```

Deletes a program group from the desktop. The *group* argument specifies the name of the program group to be deleted.

Return value:

0

The method was successful.

1

The method failed.

1.8. deleteItem

```
>>-deleteItem(shortcut)-----------------------------------------><
```

Deletes a shortcut from a program group.

Return value:

0

The method was successful.

1

The method failed.

1.9. showGroup

```
>>-showGroup(-group,-+-----+-)---------------------------------><
                    +-MIN-+
                    +-MAX-+
```

Opens a program group. The *group* argument specifies the name of the program group to be opened. If MIN or MAX is specified, the program group is opened minimized or maximized.

Return value:

0

The method was successful.

1

The method failed.

1.10. Symbolic Names for Virtual Keys

Table 3 shows the symbolic names and the keyboard equivalents for the virtual keys used by Object Rexx.

Table 1-2. Symbolic Names for Virtual Keys

Symbolic Name	Mouse or Keyboard Equivalent
LBUTTON	Left mouse button
RBUTTON	Right mouse button
CANCEL	Control-break processing
MBUTTON	Middle mouse button (three-button mouse)
BACK	BACKSPACE key
TAB	TAB key
CLEAR	CLEAR key
RETURN	ENTER key
SHIFT	SHIFT key
CONTROL	CRTL key
MENU	ALT key
PAUSE	PAUSE key
CAPITAL	CAPS LOCK key
ESCAPE	ESC key
SPACE	SPACEBAR
PRIOR	PAGE UP key
NEXT	PAGE DOWN key
END	END key
HOME	HOME key
LEFT	LEFT ARROW key
UP	UP ARROW key
RIGHT	RIGHT ARROW key
DOWN	DOWN ARROW key
SELECT	SELECT key
EXECUTE	EXECUTE key
SNAPSHOT	PRINT SCREEN key
INSERT	INS key
DELETE	DEL key
HELP	HELP key

Symbolic Name	Mouse or Keyboard Equivalent
0	0 key
1	1 key
2	2 key
3	3 key
4	4 key
5	5 key
6	6 key
7	7 key
8	8 key
9	9 key
A	A key
B	B key
C	C key
D	D key
E	E key
F	F key
G	G key
H	H key
I	I key
J	J key
K	K key
L	L key
M	M key
N	N key
O	O key
Q	Q key
R	R key
S	S key
T	T key
U	U key
V	V key
W	W key
X	X key
Y	Y key
Z	Z key
NUMPAD0	Numeric keypad 0 key
NUMPAD1	Numeric keypad 1 key
NUMPAD2	Numeric keypad 2 key

Symbolic Name	Mouse or Keyboard Equivalent
NUMPAD3	Numeric keypad 3 key
NUMPAD4	Numeric keypad 4 key
NUMPAD5	Numeric keypad 5 key
NUMPAD6	Numeric keypad 6 key
NUMPAD7	Numeric keypad 7 key
NUMPAD8	Numeric keypad 8 key
NUMPAD9	Numeric keypad 9 key
MULTIPLY	Multiply key
ADD	Add key
SEPARATOR	Separator key
SUBTRACT	Subtract key
DECIMAL	Decimal key
DIVIDE	Divide key
F1	F1 key
F2	F2 key
F3	F3 key
F4	F4 key
F5	F5 key
F6	F6 key
F7	F7 key
F8	F8 key
F9	F9 key
F10	F10 key
F11	F11 key
F12	F12 key
F13	F13 key
F14	F14 key
F15	F15 key
F16	F16 key
F17	F17 key
F18	F18 key
F19	F19 key
F20	F20 key
F21	F21 key
F22	F22 key
F23	F23 key
F24	F24 key
NUMLOCK	NUM LOCK key

Symbolic Name	Mouse or Keyboard Equivalent
SCROLL	SCROLL LOCK key

Chapter 2. The WindowsClipboard Class

The WindowsClipboard class provides methods to interact with a clipboard. Typically a clipboard is used to transfer data back and forth between different windows in a graphical user interface.

The WindowsClipboard class is not a built-in class. It is defined in the `winSystm.cls` file. This means, you must use a `::requires` statement to use its functionality, as follows:

```
::requires "winSystm.cls"
```

Methods the WindowsClipboard Class Defines

- copy
- makeArray
- paste
- empty
- isDataAvailable

2.1. copy

```
>>-copy--(--text--)-----------------------------------------><
```

Empties the clipboard and copies the specified text to it.

2.2. makeArray

```
>>-makeArray-----------------------------------------------><
```

If the content of the clipboard is a string with newline characters in it, makeArray can be used to split up the string into individual lines. An array is returned containing those lines.

2.3. paste

```
>>-paste---------------------------------------------------><
```

Retrieves the text data stored on the clipboard.

2.4. empty

```
>>-empty---------------------------------------------------><
```

Empties the clipboard.

2.5. isDataAvailable

```
>>-isDataAvailable-------------------------------------------><
```

Returns 1 if the text data is available on the clipboard. If no data is available, 0 is returned.

Chapter 3. The WindowsRegistry Class

The `WindowsRegistry` class allows the programmer to interface with the operating sytem APIs that are used to access the *registry*. The class can be used to query the registry and modify, add, and delete entries.

In the Windows operating systems, the *registry* is a system-defined database in which applications and system components store and retrieve configuration data. The data stored in the registry varies according to the version of Microsoft Windows. Applications use the registry APIs to retrieve, modify, or delete registry data.

You should not edit registry data that does not belong to your application unless it is absolutely necessary. If there is an error in the registry, your system may not function properly. If this happens, you can restore the registry to the state it was in when you last started the computer successfully. For more information, see the help for your operating system.

The registry stores data in a tree format. Each node in the tree is called a key. Each key can contain both subkeys and data entries called values. Sometimes, the presence of a key is all the data that an application requires; other times, an application opens a key and uses the values associated with the key. A key can have any number of values, and the values can be in any form.

Each key has a name consisting of one or more printable characters. Key names are not case sensitive. Key names cannot include a backslash (\), but any other printable or unprintable character can be used. The name of each subkey is unique with respect to the key that is immediately above it in the hierarchy. Key names are not localized into other languages, although values may be.

Note: Windows provides a command line user tool named `regedit` that displays the registry and its tree structure on the local machine. The tool can be very helpful in picturing the layout of the registry. To use it, merely type `regedit` at the command prompt of a console window, or use the *run* option of the `Start` menu.

Most of the operating system functions that manipulate the registry require the open handle of a parent key. As a convenience to the programmer, the `WindowsRegistry` class usually allows this handle to be omitted as an argument in its methods. The class keeps track of the most recently opened handle and supplies this handle when the programmer omits the parent handle argument in a method. This mechanism is implemented through the current_key attribute. When a method has a parent handle argument and the programmer omits the arguement, the current key handle is used.

The WindowsRegistry class is not a built-in class; it is defined in the file winSystm.cls.

Use a `::requires` statement to use the class in a program.

```
::requires "winSystm.cls"
```

A sample program, `registry.rex`, is provided in the `samples\winsystem` directory.

Methods the WindowsRegistry Class Defines

- new (Class method)
- classes_root (Attribute [get])
- close
- create

- current_key (Attribute [get])
- current_key= (Attribute [set])
- current_user (Attribute [get])
- delete
- deleteKey
- deleteValue
- flush
- getValue
- list
- listValues
- load
- local_machine (Attribute [get])
- open
- query
- replace
- restore
- save
- setValue
- unload
- users (Attribute [get])

3.1. new (Class method)

```
>>-new-------------------------------------------------------><
```

Creates an instance of the WindowsRegistry class. The current key is set to HKEY_LOCAL_MACHINE.

3.2. classes_root (Attribute [get])

```
>>-classes_root-----------------------------------------------><
```

Returns the handle of the root key HKEY_CLASSES_ROOT. This handle is maintained by the operating system, it can not be changed.

3.3. close

```
>>-close(-+------------+-)-------------------------------------><
```

```
          +-key_handle-+
```

Closes a previously opened key specified by its handle. Example:

```
rg~close(objectrexxkey)
```

It can take several seconds before all data is written to disk. You can use FLUSH to empty the cache.

If *key_handle* is omitted, CURRENT_KEY is closed.

3.4. connect

```
>>-connect(-key,computer-)------------------------------------><
```

Opens a key on a remote computer. This is supported only for HKEY_LOCAL_MACHINE and HKEY_USERS.

3.5. create

```
>>-create(-+--------+-,subkey)--------------------------------><
           +-parent-+
```

Adds a new named subkey to the registry and returns its handle. The first argument is the parent key handle. The second argument is the name of the new subkey.

Example:

```
newKey = rg~create(rg~local_machine, "myOwnKey")
```

3.6. current_key (Attribute [get])

```
>>-current_key-------------------------------------------------><
```

Returns the handle of the current key. The current key is set by the new() method to `HKEY_LOCAL_MACHINE`. It's value is then updated by every call to create() and open(). Therefore its value is always that of the handle of the most recently opened key unless the programmer sets it to some other value.

Most registry operations require an open handle to the parent key of the subkey being operated on. In the `WindowsRegistry` class most methods that require the parent key allow the programmer to omit the parent key. When the parent key is omitted, the `current_key` handle is used.

3.7. current_key= (Attribute [set])

```
>>-current_key=-------------------------------------------------><
```

Sets the handle of the current key. The `WindowsRegistry` class maintains this key, but the programmer can set it to any value at any time.

3.8. current_user (Attribute [get])

```
>>-current_user-------------------------------------------------><
```

Returns the handle of the root key HKEY_CURRENT_USER. This handle is maintained by the operating system and can not be changed.

3.9. delete

```
>>-delete(-+------------+-,-subkeyName-)----------------------><
          +-keyHandle--+
```

Deletes a subkey and all its descendants. The method will remove the key, all of the key's values, and all of its subkeys from the registry. To delete a key only if the key does not have subkeys values, use the deleteKey() method.

Arguments

The two arguments are:

keyHandle [optional]

A handle to an open registry key. The key must have been opened with the DELETE access right. If this argument is omitted then the CURRENT_KEY attribute is used.

subkeyName [required]

The name of the subkey to be deleted. The name is case insensitive.

Return

O on success, otherwise the Windows system error code. A generic description of the error can be obtained by using the Rexx Utility function, `SysGetErrorText()` .

3.10. deleteKey

```
>>-deleteKey(-+------------+-,-subkeyName-)----------------------><
             +-keyHandle--+
```

Deletes a subkey and its values from the registry. The subkey to be deleted must not have subkeys. To delete a key and all its subkeys, you need to enumerate the subkeys and delete them individually. To delete keys recursively, use the delete() method.

Arguements

The two arguments are:

keyHandle [optional]

A handle to an open registry key. The key must have been opened with the DELETE access right. If this argument is omitted then the CURRENT_KEY attribute is used.

subkeyName [required]

The name of the subkey to be deleted. The name is case insensitive.

Return

O on success, otherwise the Windows system error code. A generic description of the error can be obtained by using the Rexx Utility function, `SysGetErrorText()` .

3.11. deleteValue

```
>>-deleteValue(-+------------+-+--------+-)--------------------><
               +-key_handle-+ +-,value-+
```

Deletes the named value for a given key. If *key_handle* is omitted, CURRENT_KEY is used. If *value* is blank or omitted, the default value is deleted.

3.12. flush

```
>>-flush(-+------------+-)-----------------------------------><
         +-key_handle-+
```

Forces the system to write the cache buffer of a given key to disk. If *key_handle* is omitted, CURRENT_KEY is flushed.

3.13. getValue

```
>>-getValue(-+------------+-+--------+-)----------------------><
             +-key_handle-+ +-,value-+
```

Retrieves the data and type for a named value of a given key. The result is a compound variable with suffixes data and type. If *key_handle* is omitted, CURRENT_KEY is used. If named *value* is blank or omitted, the default value is retrieved. Example:

```
myval. = rg~getvalue(,"filesystem")        /* current key */
say "Type is" myval.type
if myval.type = "NORMAL" then say "Value is" myval.data
myval. = rg~getvalue(mykey)
say "my default value is:" myval.data
myval. = rg~getvalue(mykey,"")
say "my default value is:" myval.data
```

Possible types: NORMAL, EXPAND, MULTI, NUMBER, BINARY, NONE, OTHER.

3.14. list

```
>>-list(-+------------+-,stem.)--------------------------------><
         +-key_handle-+
```

Retrieves the list of subkeys for a given key in a stem variable. The name of the stem variable must include the period. The keys are returned as stem.1, stem.2, and so on. Example:

```
rg~LIST(objectrexxkey,orexxkeys.)
do i over orexxkeys.
say orexxkeys.i
end
```

3.15. listValues

```
>>-listValues(-+------------+-,variable.)----------------------><
               +-key_handle-+
```

Retrieves all value entries of a given key into a compound variable. The name of the variable must include the period. The suffixes of the compound variable are numbered starting with 1, and for each number the three values are the name (var.i.name), the data (var.i.data), and the type (var.i.type). The type is NORMAL for alphabetic values, EXPAND for expandable strings such as a path, NONE for no specified type, MULTI for multiple strings, NUMBER for a 4-byte value, and BINARY for any data format.

If *key_handle* is omitted, the values of CURRENT_KEY are listed.

Example:

```
qstem. = rg~QUERY(objectrexxkey)
rg~LISTVALUES(objectrexxkey,lv.)
do i=1 to qstem.values
say "name of value:" lv.i.name "(type="lv.i.type")"
if lv.i.type = "NORMAL" then
say "data of value:" lv.i.data
end
```

3.16. load

```
>>-load(-+------------+-,subkeyname, filename)-----------------><
         +-key_handle-+
```

Load creates a named subkey under the open key key_handle and loads registry data from the file filename (created by SAVE) and stores the data under the newly created subkey.

key_handle can only be HKEY_USERS or HKEY_LOCAL_MACHINE. Registry information is stored in the form of a hive - a discrete body of keys, subkeys, and values that is rooted at the top of the registry hierarchy. A hive is backed by a single file.

If *key_handle* is omitted, the subkey is created under HKEY_LOCAL_MACHINE.

Use UNLOAD to delete the subkey and to unlock the registry data file filename.

3.17. local_machine (Attribute [get])

```
>>-local_machine-------------------------------------------------><
```

Returns the handle of the root key HKEY_LOCAL_MACHINE. This handle is maintained by the operating system, it can not be changed.

3.18. open

```
>>-open(-+--------------+--+----------+--+----------+-)---------------><
         +-parentHandle-+  +-,-subkey-+  +-,-access-+
```

Opens a named subkey with the specified access rights and returns its handle. When the programmer is done with the handle it should be closed using the close() method.

Note: The default for the *access* argument is ALL access. As Microsoft has tightened up the security in it operating systems, it has made access to the registry more restrictive than it was when the `WindowsRegistry` class was first introduced. Opening a registry key with more access rights than the user running the Rexx program has will result in failure. Best practice is to open a key with the least rights needed for the operation being performed.

For instance, in a Rexx program where the function is to read the values of a single key, the key should be opened with just the INQUIRE right. This is not only less likely to fail if the program user is not an Administrator, but is more secure in every case.

Arguments

The two arguments are:

parentHandle [optional]

A handle to the open parent key. If this argument is omitted then the current_key attribute is used.

subkey [optional]

The name of the subkey to be opened. If this argument is omitted or the empty string, a new handle to the key identified by *parentHandle* is opened.

subkey [optional]

A string consisting of one or more of the following key words. The keywords are not case sensitive. The default is the ALL keyword.

ALL

Opens the key with all possible access. Although this is the default, as discused above some forethought should be given to using this access.

WRITE

Combines the rights to create subkeys, set key values, and to read a key's access rights.

READ

Combines the rights to query key values, enumerate subkeys, read keys access rights, and the notify right. Note that READ access and EXECUTE access are exactly the same.

QUERY

Combines the right to query values of a registry key (specified by the INQUIRE keyword here,) with the right to enumerate subkeys.

INQUIRE

Required to query the values of a registry key.

ENUMERATE

Required to enumerate the subkeys of a registry key.

SET

Required to create, delete, or set a registry value.

DELETE

Exactly equivalent to SET access. The keyword is a convenience to make programs more readable.

CREATE

Required to create a subkey of a registry key.

NOTIFY

Required to request change notifications for a registry key or for subkeys of a registry key.

EXECUTE

Exactly equivalent to READ access.

LINK

The Microsoft documentation states that this right is reserved for system use. The keyword is listed here simply because it was documented in previous versions of ooRexx and Object Rexx. The programmer is advised not to use it.

Return

A handle to the opened key on success, otherwise 0. If the key was opened correctly, the value of the current_key attribute is set to this handle. If the method fails, `current_key` is left unchanged.

3.19. query

```
>>-query--(--+------------+--)---------------------------------><
             +-key_handle-+
```

Retrieves information about a given key in a compound variable. The values returned are *class* (class name), *subkeys* (number of subkeys) *values* (number of value entries), *date* and *time* of last modification. If *key_handle* is omitted, CURRENT_KEY is queried. Example:

```
myquery. = rg~query(objectrexxkey)
say "class="myquery.class "at" myquery.date
say "subkeys="myquery.subkeys "values="myquery.values
```

3.20. replace

```
>>-replace(-+------------+-,-+------------+-,newfilename,oldfilename-)-><
            +-key_handle-+   +-subkeyname-+
```

Replaces the backup file of a key or subkey with a new file. Key must be an immediate descendant of HKEY_LOCAL_MACHINE or HKEY_USERS. If *key_handle* is omitted, the backup file of CURRENT_KEY is replaced. The values in the new file become active when the system is restarted. If *subkeyname* is omitted, the key and all its subkeys will be replaced.

3.21. restore

```
>>-restore(-+------------+-,filename-+-------------+-)---------><
            +-key_handle-+           +-,"VOLATILE"-+
```

Restores a key from a file. If *key_handle* is omitted, CURRENT_KEY is restored. Example:

```
rg~restore(objectrexxkey,"\objrexx\orexx")
```

The VOLATILE keyword creates a new memory-only set of registry information that is valid only until the system is restarted.

3.22. save

```
>>-save(-+------------+-,filename)-----------------------------><
         +-key_handle-+
```

Saves the entries of a given key into a file. If *key_handle* is omitted, CURRENT_KEY is saved. Example:

```
rg~SAVE(objectrexxkey,"\objrexx\orexx")
```

On a FAT system, do not use a file extension in *filename*.

3.23. setValue

```
>>-setValue(-+------------+-,-+------+-,value-+---------+-)----><
             +-key_handle-+   +-name-+        +-,NORMAL-+
                                               +-,EXPAND-+
                                               +-,MULTI--+
                                               +-,NUMBER-+
                                               +-,BINARY-+
                                               +-,NONE---+
```

Sets a named value of a given key. If *name* is blank or omitted, the default value is set. Examples:

```
rg~SETVALUE(objectrexxkey, ,"My default","NORMAL")
rg~SETVALUE(objectrexxkey,"Product_Name","Object Rexx")
rg~SETVALUE(objectrexxkey,"VERSION","1.0")
```

3.24. unload

```
>>-unload(-+------------+-,subkey)-----------------------------><
           +-key_handle-+
```

Removes a named subkey (created with LOAD) and its dependents from the registry, but does not modify the file containing the registry information. If *key_handle* is omitted, the subkey under CURRENT_KEY is unloaded. Unload also unlocks the registry information file.

3.25. users (Attribute [get])

```
>>-users-----------------------------------------------------><
```

Returns the handle of the root key HKEY_USERS. This handle is maintained by the operating system and can not be changed.

Chapter 4. The WindowsEventLog Class

The `WindowsEventLog` class provides functionaliy to interact with the Windows system event log.

The WindowsEventLog class is not a built-in class. It is defined in the file `winSystm.cls`. To use the class, place a `::requires` statement in the program file:

```
::requires "winSystm.cls"
```

A sample program `eventLog.rex` is provided in the `samples\winsystem` directory.

Methods:

The WindowsEventLog class implements the class and instance methods listed in the following table.

Table 4-1. WindowsEventLog Methods

Method	Category
new	Class method
events	Attribute
minimumReadBuffer	Attribute
minimumReadMin	Attribute
minimumReadMax	Attribute
clear	Instance method
close	Instance method
getFirst	Instance method
getLast	Instance method
getLogNames	Instance method
getNumber	Instance method
isFull	Instance method
minimumRead	Instance method
minimumRead=	Instance method
open	Instance method
Deprecated read	**Deprecated** instance method
readRecords	Instance method
write	Instance method

4.1. Using WindowsEventLog

In Windows the Event Log service provides a central facility for both the operating system and applications to log important events. The primary purpose for logging an event is to give administrators a way to determine the cause of errors and to prevent future errors. The Event Log service provides several standard logs: Application, Security, and System. The service also allows for applications to register and create Custom logs. Each event is logged as a single event log record in a single log.

The ooRexx `WindowsEventLog` class has methods that allow the programmer to query, read from, write to, back up, and clear event logs. The class can access logs on both the local machine and on remote machines accessed through the network. Full access to any log is governed by the security settings of the system. Therefore an ooRexx program that interacts with the Event Log service will be restricted to the privilege level of the user running the program.

The Event Log service uses information stored registry. This information controls how the service operates. The following list discusses some of the event logging elements to help the programmer better understand the methods and method arguments of the `WindowsEventLog` class:

Eventlog key

The Eventlog key is the key in the registry where all information for the Event Log service is stored. There are several subkeys under the EventLog key. Each subkey names an event *log*. The following shows the structure of the Eventlog key. The *Application, Security* and *System* subkeys below name the standard logs provided by the system. The actual name(s) and the number of *Custom logs* are dependent on the system.

```
HKEY_LOCAL_MACHINE
  System
    CurrentControlSet
      Services
      EventLog
        Application
        Security
        System
        CustomLog
```

Server

Many of `WindowsEventLog` instance methods have a *server* argument. This argument indentifies which machine contains the desired event log. The argument is always optional, with the default server being the local machine. In all cases, using the empty string is the same as omitting the argument.

To work with a log on a remote system, the server name must be in Universal Naming Convention (UNC) format. For instance, `\\Osprey`.

Note that if there is an open event log, the *server* argument is ignored.

Event Source

The *event source* is the name of the software or driver that logs the event. Event source names are usually the name of the application, or a component of the application if the application is large, or the driver name. Applications normally use the Application log, while drivers normally use the System log. Event source names are stored in the registry as subkeys of the log they are used in. Take the following registry example:

```
HKEY_LOCAL_MACHINE
  System
    CurrentControlSet
      Services
      EventLog
        Application
          WinApp1
```

```
    LoadPerf
  Security
    Security
  System
    Dhcp
    atapi
```

WinApp1, LoadPerf, Security, Dhcp, and *atapi* are all event sources.

Like the server argument, many of the WindowsEventLog instance methods have a *source* argument. This argument specifies the event source and therefore determines exactly which event log is used. The argument is always optional, the default is Application, and the empty string is the same as omitting the argument. In the same manner as the server argument, if there is an open event log, the *source* argument is ignored.

Note that if the Event Log service can not find the event source name in the registry, then the service also uses Application for the source.

When opening, querying, or reading event logs, using an event source name is no different than using the log name itself. For the above registry example, using:

```
eventLog~open( , "WinApp1")
```

or:

```
eventLog~open( , "LoadPerf")
```

is exactly the same as:

```
eventLog~open( , "Application")
```

However, when writing to an event log, the event source is included as part of the event log record. Therefore:

```
eventLog~open( , "WinApp1")
```

produces a different result than using:

```
eventLog~open( , "LoadPerf")
```

Although both event records will be written to the System log, the records will show the event source as WinApp1 in the first record and LoadPerf for the source in the second record.

Event Log Record

Each event is stored in an event log as a single record. The information in the record includes things like: time, type, category, record number, etc.. Each record contains the same fields, although some fields, like the binary data field, are not always filled in.

Each record has a record number. The first record written to a log is number 1 and records are then written consecutively. This makes the record with the lowest record number the oldest record. Likewise the highest record number makes that record the youngest record. The user can set a property for an event log to overwrite records when the maximum log size is reached. Because of this, the oldest record is not always record number 1. The getFirst() and getLast() methods can be used to get the absolute record numbers of the oldest and youngest records.

Record numbers can be used in the readRecords() method to read portions of the event log rather than the entire log.

When the readRecords() method reads a record it converts the information in the record to a string with a defined format that makes it easy to parse. The parts (fields) of the string are as follows, in order:

Table 4-2. Event Record Fields

Field	Description	Format
type	The event type	Single word
date	The date the event was written	Single word
time	The time the event was written	Single word
source	The The event source	Enclosed in single quotes
id	The event ID	Single word
userID	The user ID, if applicable	Single word
computer	The machine generating the event	Single word
description	A descripiton of the event	Enclosed in single quotes
data	Binary data associated with the event	Enclosed in single quotes

Assuming that *rec* is the event record string, the following shows how to parse the string into its component fields:

```
parse var rec type date time "’" source "’" id userID computer "’" description "’" "’" data "’"
```

Event Type

Each event recorded in an event log is a single type. There are five types of events that can be logged. Each event type has well-defined common data and some optional data that is specific to the event. When an event is logged the event type is included.

When the `WindowsEventLog` instance reads a record, the event type is indicated by a keyword. When the programmer writes to the event log using a `WindowsEventLog` object, she specifies the event type with its numeric value. The following table contains information on the five event types and shows the event type keywords and numeric values.

Table 4-3. Event Types

Type	Description	Keyword	Value
Error	An event that indicates a significant problem such as loss of data or loss of functionality. For example, if a service fails to load during startup, an Error event is logged.	Error	1 (0x01)
Warning	An event that is not necessarily significant, but may indicate a possible future problem. For example, when disk space is low, a Warning event is logged. If an application can recover from an event without loss of functionality or data, it can generally classify the event as a warning event.	Warning	2 (0x02)

Type	Description	Keyword	Value
Information	An event that describes the successful operation of an application, driver, or service. For example, when a network driver loads successfully, it may be appropriate to log an Information event. Note that it is generally inappropriate for a desktop application to log each time it starts.	Information	4 (0x04)
Information*	* The Windows API allows the numeric value of 0. This is not a separate event type, but rather a mapping of 0 to the Infomation event type. The `WindowsEventLog` also allows the use of 0 for the numeric value of an event type and maps it to Information.	Information	0 (0x00)
Success Audit	An event that records an audited security access attempt that is successful. For example, a user's successful attempt to log on to the system is logged as a Success Audit event.	Success	8 (0x08)
Failure Audit	An event that records an audited security access attempt that fails. For example, if a user tries to access a network drive and fails, the attempt is logged as a Failure Audit event.	Failure	16 (0x10)

Event ID

The event identifier value is specific to the event source for the event. It is used with source name to locate a description string in the message file for the event source.

Opened event log

When an event log has been opened using the open() method, that opened log is always used until it has been closed. The log can be closed using the close() method, or by another call to the `open()` method. This means that if there is an open log, the *server* and *source* arguments are **always** ignored. The only exception to this is the write() method. Each time a record is written to a log, the log is specifically opened for writing and then closed.

Note that when there is not an open event log, then all the instance methods behave as the `write()` method. That is, methods like readRecords, isFull, etc., will open the log specified in the method call and then explicitly close the log before returning.

Event Category

Categories are used to organize events so that an event viewer can filter them. Each event source can define its own numbered categories and the text strings to which they are mapped.

The categories must be numbered consecutively, beginning with the number 1. The categories themselves are defined in a message file and the category number maps to a text string in the message file.

4.2. new (Class method)

```
>>--new---------------------------------><
```

Creates an instance of the WindowsEventLog class.

4.3. minimumReadMin (Attribute)

```
>>--minimumReadMin----------------------><
```

The programmer can set the size of the minimum read buffer, within limits. `minimumReadMin` is the lowest acceptable size of the minimum read buffer.

~minimumReadMin= (set minimumReadMin)

`minimumReadMin=` is a private method, not intended to be changed by the programmer.

~minimumReadMin (get minimumReadMin)

The minimum number of kilobytes that the minimum read buffer can be set to.

Example:

The use of the attribute is straight-forward.

```
eventLog = .WindowsEventLog~new
say "Smallest possible read buffer is" eventLog~minimumReadMin "kilobytes"

::requires 'winSystm.cls'

/* Output might be:

Smallest possible read buffer is 16 kilobytes

*/
```

4.4. minimumReadMax (Attribute)

```
>>--minimumReadMax----------------------><
```

The programmer can set the size of the minimum read buffer, within limits. The `minimumReadMax` value is the largest acceptable size for the minimum read buffer.

~minimumReadMax= (set minimumReadMax)

`minimumReadMax=` is a private method, not inteneded to be changed by the programmer.

~minimumReadMax (get minmumReadMax)

The maximum number of kilobytes that the minimum read buffer can be set to.

Example:

This example displays the maximum size the programmer can set the minimum read buffer to.

```
eventLog = .WindowsEventLog~new
say "Largest possible minimum read buffer is" eventLog~minimumReadMax "kilobytes"

::requires 'winSystm.cls'

/* Output might be:

Largest possible minimum read buffer is 256 kilobytes

*/
```

4.5. minimumReadBuffer (Attribute)

```
>>--minimumReadBuffer------------------><
```

Returns the current size of the minimum read buffer in bytes. The programmer can adjust the size of the minimum read buffer. The value of this attribute reflects that size.

~minimumReadBuffer= (set minimumReadBuffer)

`minimumReadBuffer=` is a private method, not intended to be changed by the programmer. The programmer changes the size of the buffer using the minimumRead= method.

~minimumReadBuffer (get minimumReadBuffer)

The current size in bytes of the minimum read buffer.

Example:

This example displays the size of the minimum read buffer when a new WindowsEventLog object is created and then displays the size after the programmer has changed the minimum.

```
eventLog = .WindowsEventLog~new
say "Current size of the minimum read buffer is" eventLog~minimumReadBuffer "bytes"

eventLog~minimumRead = 64
 say "Adjusted size of the minimum read buffer to" eventLog~minimumReadBuffer "bytes"

::requires 'winSystm.cls'

/* Output might be:

Current size of the minimum read buffer is 16384 bytes
Adjusted size of the minimum read buffer to 65536 bytes

*/
```

4.6. events (Attribute)

```
>>--events-----------------------------><
```

The `events` attribute is an array that holds the event log records that are read from the event log during a call to readRecords(). The array is empty if no call to `readRecords()` has been made. Each time `readRecords()` is called the array is first emptied.

Each index in the array holds one event record in the form of a string with a fixed format.

~events= (set events)

`events=` is a private method, not intended to be changed by the programmer.

~events (get events)

Returns the array holding the event log records from the last readRecords() call. The array will be empty if no call to `readRecords()` has been made.

Example:

This example displays the number of event log records that were read from the System log.

```
eventLog = .WindowsEventLog~new
if eventLog~readRecords("BACKWARDS", , "System") == 0 then do
  say 'The System log has' eventLog~events~items 'records'
end

::requires 'winSystm.cls'

/* Output might be:

The System log has 1983 records

*/
```

4.7. open

```
>>--open(--+---------+--+------------+--)----------------------><
           +-server--+  +-,--source--+
```

Opens the specified event log. Once an event log is opened, other methods of the `WindowsEventLog` instance will use that opened log until it has been closed.

If an event log is already open, then it is first closed before the specified log is opened.

Arguments:

The arguments are:

server

Optional. The name of the server where the event log resides.

source

Optional. The event source.

Return value:

This method returns 0 on success, and the operating system error code on failure.

Example:

The following two code snippets are equivalent. They both open the Application log on the local machine if they succeed:

```
eventLog1 = .WindowsEventLog~new
ret = eventLog1~open
if ret \== 0 then do
  say 'Failed to open the event log:'
  say '  Error' ret':'SysGetErrortext(ret)
  ...
end

eventLog2 = .WindowsEventLog~new
ret = eventLog2~open( , "Application")
if ret \== 0 then do
  say 'Failed to open the event log'
  say '  Error' ret':'SysGetErrortext(ret)
  ...
end
```

The following example opens the System log on SERVER01:

```
eventLog = .WindowsEventLog~new
ret = eventLog~open("\\SERVER01", "System")
if ret == 0 then do
  -- Do something with the event log
  ...
  eventLog~close
  ...
end
else do
  -- Handle the error in some way
  ...
end
```

4.8. close

```
>>-close--------------------------------><
```

Closes an open event log. If no log is open, this method does nothing.

Arguments:

There are no arguments.

Return value:

This method returns 0 on success. If there is an error closing the event log the operating system error code is returned. An error is highly unlikely.

Example:

The following code snippet opens the default event log (the Application log,) displays some information about the log, then closes the open log.

```
log = .WindowsEventLog~new

ret = log~open
if ret == 0 then do
  say "  Total records:     " log~getNumber
  say "  First record number" log~getFirst
  say "  Last record number " log~getLast
  say "  Log is full?       " log~isFull
  log~close
end

/* Output might be:

  Total records:      1827
  First record number 1
  Last record number  1827
  Log is full?        0

/*
```

4.9. read (deprecated)

Note: This method is deprecated. It is replaced by the functionally equivalent readRecords() method. Do not use this method in new code. Try to migrate existing code to to the `readRecords()` method. This method may not exist in future versions of ooRexx.

4.10. readRecords

```
>>--readRecords(-+-----------+-+----------+-+----------+-+---------+-+---------+--)-->
                 +-direction-+ +-,-server-+ +-,-source-+ +-,-start-+ +-,-count-+
```

Reads the desired event records from the specified event log. Each record is stored in the events array. After a successful read, all records will be contained in the `events` array in the order there were read. Prior to starting the read opereation the array is emptied.

Details:

This method will raise syntax errors if the *start* or *count* arguments are used incorrectly. These arguments, if used, must specify records actually contained in the event log. Use any combination of the getFirst(), getLast(), or getNumber() methods to determine the absolute record numbers contained in the log.

During a read operation, if a single event record is larger than the read buffer an execution error will be raised. The text of the error will read: An event log record is too large (*recordSize*) for the read buffer (*bufferSize*.) Where *recordSize* is the size of the record and *bufferSize* is the size of the read buffer at the time of the error.

The minimum size of the read buffer can be increased by using the minimumRead= method. If this error occurred, the minimum read buffer should be set larger than the size of the offending record.

Note well: It seems inconceivable that the read buffer could be smaller than a single event record. The minimum possible size of the buffer is 16 KB and the average size of an event record is between 100 and 200 bytes. The ooRexx programmer should **not** worry about this. This unlikely possibility is simple documented for the sake of completeness.

Arguments:

The arguments are:

direction

Optional. The direction to read the from the event log, forwards or backwards. The default is to read forwards. If this argument is not omitted, it must be exactly one of the keywords, BACKWARDS or FORWARDS. Case is not significant.

server

Optional. The name of the server where the event log resides

source

Optional. The event source.

start

Optional. The starting record number for the read operation. The start and the count arguments must be used together. Either both must be used or neither. If both arguments are omitted, the entire log is read. When both arguments are used, the read begins with the record number specified by *start* and reads in the direction specified for *count* records.

count

Optional. The count of records to be read during the read operation. The start and the count arguments must be used together. Either both must be used or neither. If both arguments are omitted, the entire log is read. When both arguments are specified, the read begins with the record number specified by *start* and reads in the direction specified for *count* records.

Return value:

This method returns 0 on success, and the operating system error code on failure.

Example:

This example reads the 5 most recent event records in the System event log and displays them to the console. (If there are less than 5 records in the log, then all the records are read.)

```
  log = .WindowsEventLog~new

  startRec = log~getLast( , "System")
  count = log~getNumber~min(5)

  ret = log~readRecords("BACKWARDS", , "System", startRec, count)

  if ret == 0 then do
    c = displayRecords(log~events)
    say 'Displayed' c 'records'
  end
  else do
    say "Error reading the System event log rc:" ret "-" SysGetErrorText(ret)
  end

::requires 'winSystm.cls'

/* Routine to display the event log records */
::routine displayRecords
  use strict arg records

  do record over records
    say "================================================================================"
    parse var record type date time "'" sourcename"'" id userid computer "'" string "'" "'" data "'"
    say 'Type     : 'type
    say 'Date     : 'date
    say 'Time     : 'time
    say 'Source   : 'sourcename
    say 'ID       : 'id
    say 'UserId   : 'userid
    say 'Computer : 'computer
    say 'Detail   : 'string
    say 'Data     : 'data
  end
  say "================================================================================"
return records~items

/* The output (shortened to 2 records) might be:

================================================================================
Type     : Information
Date     : 02/14/09
Time     : 11:32:21
Source   : WinHttpAutoProxySvc
ID       : 12503
UserId   : N/A
Computer : OSPREY
Detail   : The WinHTTP Web Proxy Auto-Discovery Service has been idle for
15 minutes, it will be shut down.
```

```
Data     :
================================================================================
Type     : Information
Date     : 02/14/09
Time     : 11:15:51
Source   : Service Control Manager
ID       : 7036
UserId   : N/A
Computer : OSPREY
Detail   : The WinHTTP Web Proxy Auto-Discovery Service service entered
the running state.

Data     :
================================================================================
Displayed 5 records

*/
```

4.11. write

```
                                                                    +-,--------+
                                                                    V          |
>>-write(-+------+-+-------+-+-------+-+------------+-+------+-+--------+-+----------+-)--
><
          +-srvr-+ +-,-src-+ +-,type-+ +-,-category-+ +-,-id-+ +-,-data-+ +-,-string-+
```

Description

Arguments:

The arguments are:

srvr

Optional. The name of the server where the event log resides

src

Optional. The event source.

type

Optional. The event type for the record. The default is the *Error* (1) event type. When used, this argument must be the numeric value of a valid event type.

category

Optional. The event category for the record. The default is 0, which is the same as no category (none.)

id

Optional. The event identifier for the record.. The default is 0.

data

Optional. The binary data for the record. The default is none. This is binary information specific to the event being logged and to the source that generated the entry. It could for example be the contents of the processor registers when a device driver got an error, a dump of an invalid packet that was received from the network, etc..

string

Optional. The default is no string. This last argument can be repeated any number of times. Each additional argument is a string used as a substitution string in the description string.

The event identifier together with the event source name identify a description string contained in a message file that describes the event in more detail. The description string can contain substitution place holders. The substitution strings named by this argument are used to replace the substitution place holders in the description string.

Return value:

This method returns 0 on success, and the operating system error code on failure.

Example:

This example writes some fictious data to an event log.

```
log = .WindowsEventLog~new

source     = "MyApplication"
type       = 4  -- Information
category   = 22
id         = 33
binaryData = "01 1a ff 4b 0C"x

ret = log~write( , source, type, category, id, binaryData, "String1", "String2")
if ret == 0 then
  say "Record" source "successfully written"
else
  say "Error writing record" source "rc:" ret ":" SysGetErrorText(ret)


::requires 'winSystm.cls'
```

4.12. clear

```
>>--clear(--+---------+--+----------+--+------------------+--)----------------><
           +-server--+  +-,-source-+  +-,-backupFileName-+
```

Clears (removes) all event records from the log. Optionally will back up the log first. When the optional backup file name is supplied and for some reason the back up fails, then the event records are not cleared.

Arguments:

The arguments are:

srvr

Optional. The name of the server where the event log resides

src

Optional. The event source.

backupFileName

Optional. The path name to a back up file. If this argument is specified, the event log is first backed up before it is cleared. If the back up fails, the log is not cleared. The back will fail if the file name specified already exists.

The back up file will be created on the system that the event log file itself is on. This means that if an event log on a remote system is specified, the log will be created on that remote system. The file name must therefore be a valid file name on the remote system.

If the file name does not contain an extension, the the normal extension for event log back ups, `.evt`, will be used.

Return value:

This method returns 0 on success, and the operating system error code on failure.

Example:

This example backs up the Application event log on the remote Eagle system and then clears the log. If the back up fails, the log will not be cleared. The back up file will be named eagle_application.evt and will be located *on* the Eagle system, not on the local machine.

```
log = .WindowsEventLog~new

ret = log~open("\\Eagle", "Application")
if ret == 0 then do
  ret = log~clear( , , "C:\eagle_application")
  if ret == 0 then do
    say 'Backed up the Application event log on Eagle to:'
    say '  C:\eagle_application.evt on the Eagle system.'
  end
  else do
    say 'Failure backing up event log:' ret ":" SysGetErrorText(ret)
  end
end

::requires 'winSystm.cls'
```

4.13. minimumRead

```
>>--minimumRead-------------------------><
```

Determines the current minimum size, in kilobytes, of the buffer used to read event log records. The minimum size of this buffer can be adjusted by the programmer.

Arguments:

There are no arguments to this method.

Return value:

The size in kilobytes of the minimum read buffer. For example if the minimum buffer size is 32,768, this method will return 32. (32 KB.)

Example:

This example displays the current value of the minimum read buffer size.

```
log = .WindowsEventLog~new
say 'Current minimum size of the read buffer is:' log~minimumRead "KB"

::requires 'winSystm.cls'

/* Output might be:

Current minimum size of the read buffer is: 16 KB

*/
```

4.14. minimumRead=

```
>>--minimumRead-=-sizeKB----------------><
```

Adjusts the minimum size of the read buffer in increments of 1024 bytes. **Note** that the programmer need not worry about the read buffer. This method is documented because it does exist and for the sake of the rare Rexx programmer that might need to change the minimum size of the read buffer.

The read buffer is used by the underlying implementation during the readRecords() method only. During a read operation, the `WindowsEventLog` attempts to allocate a buffer that is big enough to read in all the records at once. The size of the buffer is guessed at by using the number of records in the event log. The size is constrained by a minimum and maximum. The buffer will never be larger than the maximum and never smaller than the minimum. The maximum value is fixed. The minimum value can be adjusted by the programmer by this, the `minimumRead()` method.

In almost all cases, the size of the buffer will be set towards the maximum constraint and the minimum constraint will not come into play at all. There is only one circumstance where the Rexx programmer would need to change the minimum constraint, which is this:

The Windows Event Log Service will only place whole records into the buffer. If a record is bigger in size than the buffer, the record can not be read and an execution error will be raised by the `WindowsEventLog` object. In this case the minimum constraint for the buffer size would need to be set to a size bigger than the record size. The text of the error message lists both the record size and the buffer size. To read the record, the programmer would set the minimum constraint larger than the record size.

Again, it *must be stressed* that the above scenario is extremely unlikely.

Arguments:

The single argument is:

sizeKB

The minimum size to allocate the read buffer, in kilobytes.

Return value:

There is no return.

Example:

This method is straight forward to use:

```
log = .WindowsEventLog~new
log~minimumRead = 64
say 'Current minimum read is' log~minimumRead 'KB.'

::requires 'winSystm.cls'

/* Output might be:

Current minimum size of the read buffer is: 64 KB

*/
```

4.15. isFull

```
>>--isFull(--+---------+-+----------+--)----------------------><
             +--server-+ +-,-source-+
```

Determines if the event log is full.

Arguments:

The arguments are:

server

Optional. The server where the event log resides.

source

Optional. The event source.

Return value:

The method returns `.true` or `.false`. True if the event log is full, otherwise false.

Example:

This example is a snippet of code from an application that monitors the system log. When the log gets full, the log is backed up and cleared.

```
::routine checkLog
  use strict arg sysLog, monitor

  if sysLog~isFull then do
    success = monitor~backupLog(sysLog)
    if \ success then monitor~notifyAdmin
  end
return success
```

4.16. getNumber

```
>>--getNumber(--+---------+-+----------+--)---------------------><
               +--server-+ +-,-source-+
```

Determines the number of records in the event log.

Arguments:

The arguments are:

server

Optional. The server where the event log resides.

source

Optional. The event source.

Return value:

On success, the count of event records in the log. On error, the return is the negated system error code.

Example:

This example opens the system log on the Osprey server. It then checks that there are at least 10 records before reading the log:

```
  log = .WindowsEventLog~new
  log~open("\\Osprey", "System")
  if log~getNumber > 10 then do
    log~readRecords
    say 'Read' log~events~items 'records.'
  end

::requires 'winSystm.cls'
```

4.17. getLogNames

```
>>--getLogNames(--names--)-------------><
```

Obtains a list of all the event log names on the current system.

Arguments:

The single argument is:

names

An array object. On return the array will contain the names of the event logs on the current system. This will include any custom logs, if there are any. The array is emptied before the names are added. If an error happens, the array will be empty.

Return value:

This method returns 0 on success, and the operating system error code on failure.

Example:

This example displays the names of all the event logs on the current system.

```
logNames = .array~new
ret = log~getLogNames(logNames)

if ret == 0 then do name over logNames
  say "Log:" name
end

::requires 'winSystm.cls'

/* Output might be:

Log: Application
Log: Internet Explorer
Log: Security
Log: System

*/
```

4.18. getLast

```
>>--getLast(--+---------+-+----------+--)---------------------><
              +--server-+ +-,-source-+
```

Determines the absolute record number of the last record in the event log.

Arguments:

The arguments are:

server

Optional. The server where the event log resides.

source

Optional. The event source.

Return value:

On success, the record number of the last (most recently written) event record. On error, the return is the negated system error code.

Example:

This example displays the last record written to application log.

```
log = .WindowsEventLog~new~~open
log~readRecords( , , , log~getLast, 1)
rec = log~events[1]
if rec \== .nil then do

  parse var rec type date time "'" src"'" id user computer "'" string "'" "'" data "'"
  say 'Type     : 'type
  say 'Date     : 'date
  say 'Time     : 'time
  say 'Source   : 'src
  say 'ID       : 'id
  say 'UserId   : 'user
  say 'Computer : 'computer
  say 'Detail   : 'string
  say 'Data     : 'data
end

::requires 'winSystm.cls'

/* Output might be:

Type     : Error
Date     : 02/14/09
Time     : 16:55:08
Source   : Windows Search Service
ID       : 3083
UserId   : N/A
Computer : OSPREY
Detail   : The protocol handler Search.Mapi2Handler.1 cannot be loaded. Error
description: Class not registered.
Data     :

*/
```

4.19. getFirst

```
>>--getFirst(--+---------+-+----------+--)---------------------><
              +--server-+ +-,-source-+
```

Determines the absolute record number of the first record in the event log.

Arguments:

The arguments are:

server

Optional. The server where the event log resides.

source

Optional. The event source.

Return value:

On success, the first record number in the event log. On error, the return is the negated system error code.

Example:

This example displays the first record written to the application log.

It is somewhat interesting to note that this first record was written right after the operating system had been installed, prior to the computer being added to a work group and given the Osprey name. This can be seen when the record is displayed, the Computer field is MACHINENAME.

```
log = .WindowsEventLog~new~~open("\\Osprey", "System")
log~readRecords( , , , log~getFirst, 1)
rec = log~events[1]
if rec \== .nil then do

  parse var rec type date time "'" src"'" id user computer "'" string "'" "'" data "'"
  say 'Type     : 'type
  say 'Date     : 'date
  say 'Time     : 'time
  say 'Source   : 'src
  say 'ID       : 'id
  say 'UserId   : 'user
  say 'Computer : 'computer
  say 'Detail   : 'string
  say 'Data     : 'data
end

::requires 'winSystm.cls'

/* Output might be:

Type     : Information
Date     : 08/16/08
```

```
Time     : 04:27:01
Source   : EventLog
ID       : 6009
UserId   : N/A
Computer : MACHINENAME
Detail   : 5.02. 3790 Service Pack 1 Multiprocessor Free
Data     :

*/
```

Chapter 5. The WindowsManager Class

The WindowsManager class provides methods to query, manipulate, and interact with windows on your desktop. Currently, this class is specifically for the Windows operating sytsem and is not available on other operating systems.

The WindowsManager class is not a built-in class, it is defined in the file `winSystm.cls`. To use the class, add a `::requires` statement to the program file:

```
::requires "winSystm.cls"
```

Methods the WindowsManager Class Defines

- desktopWindow
- find
- foregroundWindow
- windowAtPosition
- consoleTitle
- consoleTitle=
- sendTextToWindow
- pushButtonInWindow
- processWindowCommand
- broadcastSettingChanged

5.1. desktopWindow

```
>>-desktopWindow-------------------------------------------><
```

Returns an instance of the WindowObject class that represents the Desktop window. The Desktop window is the parent of all top-level windows and therefore the ancestor of every window on the system. If some error happens, `.nil` is returned. (This is extremely unlikely.)

5.2. find

```
>>-find--(--title--)---------------------------------------><
```

Searches for a top-level window (not a child window) on your desktop with the specified *title*.

If this window already exists, an instance of the WindowObject class is returned. Otherwise, .Nil is returned.

5.3. foregroundWindow

```
>>-foregroundWindow-------------------------------------------><
```

Returns an instance of the WindowObject class that is associated with the current foreground window.

5.4. windowAtPosition

```
>>-windowAtPosition--(--x--,--y--)----------------------------><
```

Returns an instance of the WindowObject class that is associated with the window at the specified position (*x,y*). The coordinates are specified in screen pixels. This method does not retrieve hidden or disabled windows. If you are interested in a particular child window, use method childAtPosition.

5.5. consoleTitle

```
>>-consoleTitle-----------------------------------------------><
```

Returns the title of the current console.

5.6. consoleTitle=

```
>>-consoleTitle=title-----------------------------------------><
```

Sets the title of the current console.

5.7. sendTextToWindow

```
>>-sendTextToWindow--(--title--,--text--)----------------------><
```

Sends a case-sensitive *text* to the window with the specified *title.*.

5.8. pushButtonInWindow

```
>>-pushButtonInWindow--(--title--,--text--)--------------------><
```

Selects the button with label *text* in the window with the specified *title*. If the button's label contains a mnemonic (underscored letter), you must specify an ampersand (&) in front of it. You can also use this method to select radio buttons and to check or uncheck check boxes.

Example:

```
winmgr~pushButtonInWindow("Testwindow","List &Employees")
```

5.9. processMenuCommand

```
>>-processMenuCommand--(--+-------+--,--+-------+--,------------>
                          +-title-+     +-popup-+

   +-,-------+
   V         |
>----submenu-+--,--menuItem--)--------------------------------><
```

Selects an item of the menu or submenu of the specified window *title*. You can specify as many submenus as necessary to get to the required item.

5.10. broadcastSettingChanged

```
>>-broadcastSettingChanged--+-----------+---------------------><
                            +-(timeOut)-+
```

Causes the Windows operating system to send a message, (the WM_SETTINGCHANGE message,) to every top-level window on the Desktop informing them that a system-wide setting has changed. Well-written applications will then reload any system settings that they use.

An example of one use for this might be an installer program setting an environment variable, such as the PATH. Then a call to broadcastSettingChanged would cause all open applications to update their reference to the environment, without the necessity of a reboot.

There are two variations of calling this method. When called with no arguments, the message is broadcast and returns immediately. When called with the time out parameter, the message is broadcast and does not return until every window on the Desktop has acknowledged the message, or timed out.

The problem with using a time out and waiting for acknowledgment is that, if a window is not responding, or several windows are slow to respond, it may take a very long time to return. The problem with not using a time out and returning immediately is that the caller will have no way of knowing when every window has received the message. Generally this is not a problem, but it is up to the programmer to decide how she wants to use this method.

The time out value is specified in milliseconds. For each window, the operating system will wait up to the time out for a response before going on to the next window. Typically a time out value of 5000 (5 seconds) is used, and this is the default.

The single optional argument is:

timeOut

The time, in milliseconds, to wait for each window to acknowledge it received the setting changed message. Specifying 0 or a negative number will cause the default time out of 5000 to be used. (5000 is a typical value used by applications.)

Return value:

0

The setting changed message was broadcast successfully. If no time out argument was used, then this is all it means. If a time out value was used, then all top-level windows have acknowledged receiving the message.

-1

The setting changed message was broadcast, but one or more windows timed out. This return can only happen when the time out parameter is used.

-x

A number less than -1 indicates a system error occurred. This value is the negation of the system error code. I.e., if the return is -1400, the system error code was 1400. System error codes can be looked up in the MSDN library or the Windows Platform SDK. Microsoft makes these references available on the Internet.

+x

A number greater than 0 would be a window result of broadcasting the setting changed message and would not be an error. It is unlikely that this would occur.

Example:

```
ret = winmgr~broadcastSettingChanged(1000)
```

Chapter 6. The WindowObject Class

The WindowObject class provides methods to query, manipulate, and interact with a particular window or one of its child windows.

Access to the WindowObject class requires that the following directive appear in the Rexx program.

```
::requires 'winSystm.cls'
```

Note. Prior to the release of ooRexx 4.0.0, the WindowsObject class was implemented using the original external function API. That API required that the external functions be registered with the interpreter. For the most part this was done transparently to the Rexx programmer. However, with the WindowsObject class there was one scenario where the registration was not done and prior documentation provided a work around.

Starting with ooRexx 4.0.0, that work around is not needed. There no longer is any need for the programmer to register external functions at all. Requiring `winSystm.cls` is all that is needed from ooRexx 4.0.0 and on. Disregard the previous documentation concerning external functions.

Methods the WindowObject Class Defines

- assocWindow
- childAtPosition
- coordinates
- disable
- enable
- enumerateChildren
- findChild
- first
- firstChild
- focusItem
- focusNextItem
- focusPreviousItem
- getStyle
- handle
- hide
- id
- isMenu
- last
- maximize
- menu
- minimize

- moveTo
- next
- owner
- previous
- processMenuCommand
- pushButton
- resize
- restore
- sendChar
- sendCommand
- sendKey
- sendKeyDown
- sendKeyUp
- sendMenuCommand
- sendMessage
- sendMouseClick
- sendSyscommand
- sendText
- state
- systemMenu
- title
- title=
- toForeground
- wclass

6.1. assocWindow

```
>>-assocWindow--(--handle--)---------------------------------><
```

Assigns a new window *handle* to the WindowObject instance.

6.2. handle

```
>>-handle-----------------------------------------------------><
```

Returns the handle of the associated window.

6.3. title

```
>>-title-------------------------------------------------------><
```

Returns the title of the window.

6.4. title=

```
>>-title=newTitle----------------------------------------------><
```

Sets a new title for the window.

6.5. wclass

```
>>-wclass------------------------------------------------------><
```

Returns the class of the window associated with the WindowObject instance.

6.6. id

```
>>-id----------------------------------------------------------><
```

Returns the numeric ID of the window.

6.7. coordinates

```
>>-coordinates-------------------------------------------------><
```

Returns the upper left and the lower right corner positions of the window in the format "left,top,right,bottom".

6.8. state

```
>>-state-------------------------------------------------------><
```

Returns information about the window state. The returned state can contain one or more of the following constants:

- "Enabled" or "Disabled"
- "Visible" or "Invisible"
- "Zoomed" or "Minimized"

- "Foreground"

6.9. getStyle

```
>>--getStyle-------------------------------------------------><
```

Returns the style and extended style flags of the window. This method is intended for use by programmers that have some knowledge of the Windows API and would not be much use to Rexx programmers that do not have any understanding of that API.

The styles are returned in a string of two words. The first word is the window style and the second word is the extend window style. Each word is in the format: `0xAAAAAAAA` where A represents any hexadecimal digit. If an error happens, the numerical system error code is returned instead of a string with two words.

Example:

```
-- This function will return an array with all matching windows.  An empty array
-- signals no match.
windows = fuzzyFindWindows(deskTop, text)

if windows~items > 0 then do wnd over windows
  say 'Found this window.'
  say '  Title:   ' wnd~title
  say '  Class:   ' wnd~wClass
  say '  Position:' wnd~coordinates
  say '  Styles:  ' wnd~getStyle
  say
end

/* Output might be:

Found this window.
  Title:    GetMenuState Function - MSDN Library - Microsoft Document Explorer
  Class:    wndclass_desked_gsk
  Position: 0,0,1152,800
  Styles:   0x16cf0000 0xc0040900

Found this window.
  Title:    C:\work.ooRexx\3.x\main
  Class:    ExploreWClass
  Position: 0,25,1150,804
  Styles:   0x16cf0000 0xc0000900

*/
```

6.10. restore

```
>>-restore---------------------------------------------------><
```

Activates and displays the associated window. If the window is minimized or maximized, it is restored to its original size and position.

6.11. hide

```
>>-hide-------------------------------------------------------><
```

Hides the associated window and activates another window.

6.12. minimize

```
>>-minimize----------------------------------------------------><
```

Minimizes the associated window and activates the next higher-level window.

6.13. maximize

```
>>-maximize----------------------------------------------------><
```

Maximizes the associated window.

6.14. resize

```
>>-resize--(--width--,--height--)------------------------------><
```

Resizes the associated window to the specified width and height. The width and height are specified in screen coordinates.

6.15. enable

```
>>-enable------------------------------------------------------><
```

Enables the associated window if it was disabled.

6.16. disable

```
>>-disable-----------------------------------------------------><
```

Disables the associated window.

6.17. moveTo

```
>>-moveTo--(--x--,--y--)---------------------------------------><
```

Moves the associated window to the specified position (*x,y*). Specify the new position in screen pixels.

6.18. toForeground

```
>>-toForeground-------------------------------------------------><
```

Makes the associated window the foreground window.

6.19. focusNextItem

```
>>-focusNextItem------------------------------------------------><
```

Sets the input focus to the next child window of the associated window.

6.20. focusPreviousItem

```
>>-focusPreviousItem--------------------------------------------><
```

Sets the input focus to the previous child window of the associated window.

6.21. focusItem

```
>>-focusItem--(--wndObject--)-----------------------------------><
```

Sets the input focus to the child window associated with the specified WindowObject instance *wndObject*.

Example:

The following example sets the input focus to the last child window:

```
dlg = wndmgr~find("TestDialog")
   if dlg \= .Nil then do
       fChild = dlg~firstChild
       lChild = fChild~last
       dlg~focusItem(lChild)
   end
```

6.22. findChild

```
>>-findChild--(--label--)----------------------------------------><
```

Returns an instance of the WindowObject class associated with the child window with the specified *label*. If the associated window does not own such a window, the .Nil object is returned.

6.23. childAtPosition

```
>>-childAtPosition--(--x--,--y--)-----------------------------><
```

Returns an instance of the WindowObject class associated with the child window at the specified client position (*x,y*). The coordinates that are relative to the upper left corner of the associated window must be specified in screen pixels. To retrieve top-level windows, use method windowAtPosition.

6.24. next

```
>>-next-------------------------------------------------------><
```

Returns an instance of the WindowObject class associated with the next window of the same level as the associated window. If the associated window is the last window of a level, the .Nil object is returned.

6.25. previous

```
>>-previous---------------------------------------------------><
```

Returns an instance of the WindowObject class associated with the previous window of the same level as the associated window. If the associated window is the first window of a level, the .Nil object is returned.

6.26. first

```
>>-first------------------------------------------------------><
```

Returns an instance of the WindowObject class associated with the first window of the same level as the associated window.

6.27. last

```
>>-last-------------------------------------------------------><
```

Returns an instance of the WindowObject class associated with the last window of the same level as the associated window.

6.28. owner

```
>>-owner-------------------------------------------------------><
```

Returns an instance of the WindowObject class associated with the window that owns the associated window (parent). If the associated window is a top-level window, the .Nil object is returned.

6.29. firstChild

```
>>-firstChild--------------------------------------------------><
```

Returns an instance of the WindowObject class associated with the first child window of the associated window. If no child window exists, the .NIL object is returned.

6.30. enumerateChildren

```
>>-enumerateChildren-------------------------------------------><
```

Returns a stem that stores information about the child windows of the associated window. "Stem.0" contains the number of child windows. The returned stem contains as many records as child windows. The first record is stored at "Stem.1" continued by increments of 1. Each record contains the following entries, where each entry starts with an exclamation mark (!):

!Handle

The handle of the window.

!Title

!Class

The window class.

!State

!Coordinates

!Children

1 if the window has child windows, 0 if is has none.

!Id

Example:

```
wo = winmgr~find("TestDialog")
  enum. = wo~enumerateChildren
  do i = 1 to enum.0   /* number of children */
     say "---"
     say "Handle:" enum.i.!Handle
     say "Title:" enum.i.!Title
     say "Class:" enum.i.!Class
     say "Id:" enum.i.!Id
     say "Children:" enum.i.!Children
     say "State:" enum.i.!State
     say "Rect:" enum.i.!Coordinates
  end
```

6.31. sendMessage

```
>>-sendMessage--(--message--,--wParam--,--lParam--)------------><
```

Sends a message to the associated window.

6.32. sendCommand

```
>>-sendCommand--(--command--)---------------------------------><
```

Sends a WM_COMMAND message to the associated window. WM_COMMAND is sent, for example, when a button is pressed, where *command* is the button ID.

6.33. sendMenuCommand

```
>>-sendMenuCommand--(--id--)----------------------------------><
```

Selects the menu item *id* of the associated window. Method idOf returns the ID of a menu item.

6.34. sendMouseClick

```
                      +-"LEFT"---+     +-"DBLCLK"-+
>>-sendMouseClick--(--+-"RIGHT"--+--,--+-"UP"-----+----,--x--,--y-->
                      +-"MIDDLE"-+     +-"DOWN"---+

>--+-----------------------------+--)-------------------------><
   |       +----------------+    |
   |       V                |    |
   +-,--"----+-LEFTDOWN---+-+--"-+
             +-RIGHTDOWN--+
             +-MIDDLEDOWN-+
```

```
+-SHIFT------+
+-CONTROL----+
```

Simulates a mouse click event in the associated window.

Arguments:

The arguments are:

which

Specifies which mouse button is simulated. LEFT is the default.

kind

Selects the simulated mouse action. DBLCLK is the default.

x,y

Specifies the coordinates of the mouse click event, in screen coordinates, relative to the upper left corner of the window.

ext

Can be one or more of the following strings:

LEFTDOWN

Simulates the pressed left mouse button.

RIGHTDOWN

Simulates the pressed right mouse button.

MIDDLEDOWN

Simulates the pressed middle mouse button.

SHIFT

Simulates the pressed Shift key.

CONTROL

Simulates the pressed Control key.

6.35. sendSyscommand

```
>>-sendSyscommand--(--"--+-SIZE--------+--"--)-----------------><
                         +-MOVE--------+
                         +-MINIMIZE----+
                         +-MAXIMIZE----+
                         +-NEXTWINDOW--+
                         +-PREVWINDOW--+
                         +-CLOSE-------+
```

```
+-VSCROLL-----+
+-HSCROLL-----+
+-ARRANGE-----+
+-RESTORE-----+
+-TASKLIST----+
+-SCREENSAVE--+
+-CONTEXTHELP-+
```

Sends a WM_SYSCOMMAND message to the associated window. These messages are normally sent when the user selects a command in the Window menu.

Argument:

The only argument is:

command

One of the commands listed in the syntax diagram:

SIZE

Puts the window in size mode.

MOVE

Puts the window in move mode.

MINIMIZE

Minimizes the window.

MAXIMIZE

Maximizes the window.

NEXTWINDOW

Moves to the next window.

PREVWINDOW

Moves to the previous window.

CLOSE

Closes the window.

VSCROLL

Scrolls vertically.

HSCROLL

Scrolls horizontally.

ARRANGE

Arranges the window.

RESTORE

Restores the window to its normal position and size.

TASKLIST

Activates the Start menu.

SCREENSAVE

Executes the screen-saver application specified in the [boot] section of the SYSTEM.INI file.

CONTEXTHELP

Changes the cursor to a question mark with a pointer. If the user then clicks on a control in the dialog box, the control receives a WM_HELP message.

6.36. pushButton

```
>>-pushButton--(--label--)-------------------------------------><
```

Selects the button with the specified *label* within the associated window and sends the corresponding WM_COMMAND message. If the button's label contains a mnemonic (underscored letter), you must specify an ampersand (&) in front of it. You can also use this method to select radio buttons and check or uncheck check boxes.

6.37. sendKey

```
>>-sendKey--(--keyName--+-----------------------+--)----------><
                        +-,--+-----+--+--------+-+
                             +-alt-+  +-,--ext-+
```

Sends all messages (CHAR, KEYDOWN, and KEYUP) that would be sent by pressing a specific key on the keyboard. Character keys (a to z) are not case-sensitive.

If the *alt* argument is 1, the Alt key flag is set, which is equal to pressing the specified key together with the Alt key.

The *Ext* argument must be 1 if the key is an extended key, such as a right Ctrl or a right Shift.

For a list of key names, refer to Symbolic Names for Virtual Keys.

6.38. sendChar

```
>>-sendChar--(--character--+--------+--)----------------------><
                           +-,--alt-+
```

Sends a WM_CHAR message to the associated window. If the *alt* argument is 1, a pressed Alt key is simulated.

6.39. sendKeyDown

```
>>-sendKeyDown--(--keyName--+--------+--)----------------------><
                            +-,--ext-+
```

Sends a WM_KEYDOWN message to the associated window. The *ext* argument must be 1 if the key is an extended key, such as a right Ctrl or a right Shift.

For a list of key names, refer to Symbolic Names for Virtual Keys.

6.40. sendKeyUp

```
>>-sendKeyUp--(--keyName--+--------+--)------------------------><
                          +-,--ext-+
```

Sends a WM_KEYUP message to the associated window. The *ext* argument must be 1 if the key is an extended key, such as a right Ctrl or a right Shift.

For a list of key names, refer to Symbolic Names for Virtual Keys.

6.41. sendText

```
>>-sendText--(--text--)-----------------------------------------><
```

Sends a (case-sensitive) text to the associated window by sending a sequence of WM_CHAR, WM_KEYDOWN, and WM_KEYUP messages.

6.42. menu

```
>>-menu----------------------------------------------------------><
```

Returns an instance of the MenuObject class that refers to the menu of the associated window.

6.43. systemMenu

```
>>-systemMenu----------------------------------------------------><
```

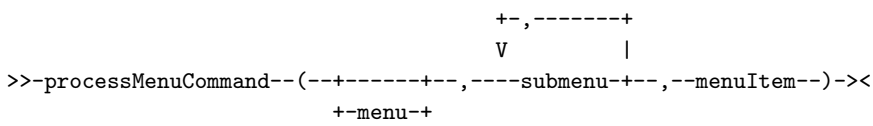
Returns an instance of the MenuObject class that refers to the system menu of the associated window.

6.44. isMenu

```
>>-isMenu--------------------------------------------------------><
```

Returns 1 if the associated window is a menu, otherwise 0.

6.45. processMenuCommand



Selects an item of the menu or submenu of the associated window. You can specify as many submenus as necessary to get to the required item.

Chapter 7. The MenuObject Class

The MenuObject class provides methods to query, manipulate, and interact with the menu or submenu of a window.

Use of the MenuObjects requires that the following directive appear in the Rexx program.

```
::requires 'winSystm.cls'
```

Methods the MenuObject Class Defines

- findItem
- findSubmenu
- idOf
- ischecked
- isMenu
- isSubMenu
- isSeparator
- items
- processItem
- submenu
- textOf(id)
- textOf(position)

7.1. isMenu

```
>>-isMenu-----------------------------------------------------><
```

Returns 1 if the associated window is a menu, otherwise 0.

7.2. isSubMenu

```
>>-isSubMenu(--position--)-------------------------------------><
```

Returns `.true` if the menu item at the position specified is a submenu of this menu, otherwise `.false`. Menu items are zero-based, so the first menu item is at position 0.

7.3. isChecked

```
>>-isChecked(--position--)-------------------------------------><
```

Returns .true if the menu item at the position specified is checked, otherwise .false. Menu items are zero-based, so the first menu item is at position 0. Submenus and separators can be be checked.

This method can not be 100% reliable. Some applications do not set the check mark for a menu item until the menu is displayed. To be confident of the result, the programmer should first test how a specific application behaves.

7.4. isSeparator

```
>>-isSeparator(--position--)---------------------------------><
```

Returns .true if the menu item at the position specified is a separator line, otherwise .false. Menu items are zero-based, so the first menu item is at position 0.

7.5. items

```
>>-items-----------------------------------------------------><
```

Returns the number of menu items contained in the associated menu.

7.6. idOf

```
>>-idOf--(--position--)---------------------------------------><
```

Returns the ID of the menu item at the specified *position*, starting with 0.

7.7. textOf(position)

```
>>-textOf--(--position--)-------------------------------------><
```

Returns the text of the menu item at the specified *position*, starting with 0. A mnemonic (underscored letter) is represented by a leading ampersand (&). If the menu item contains an accelerator, it is separated by a tab.

7.8. textOf(id)

```
>>-textOf--(--id--)-------------------------------------------><
```

Returns the text of menu item *id*. A mnemonic is represented by a leading ampersand (&). If the menu item contains an accelerator, it is separated by a tab.

7.9. submenu

```
>>-submenu--(--position--)-------------------------------------><
```

Returns an instance of the MenuObject class that is associated with the submenu at the specified *position*, starting with 0. If no submenu exists at this position, the .Nil object is returned.

Example:

```
sub = menu~submenu(5)
    if sub \= .Nil then do
         say "Items:" sub~items
    end
```

7.10. findSubmenu

```
>>-findSubmenu--(--label--)------------------------------------><
```

Returns an instance of the MenuObject class that is associated with the submenu with the specified *label*. If the associated menu does not contain such a submenu, the .Nil object is returned.

7.11. findItem

```
>>-findItem--(--label--)----------------------------------------><
```

Returns the ID of the menu item *label*. If the specified label does not include an accelerator, the comparison excludes the accelerators of the menu items. If no menu item is found that matches the specified label, 0 is returned.

Example:

```
f = menu~findItem("&Tools" || "9"x || "Ctrl+T")
    if f \= 0 then menu~processItem(f)
```

7.12. processItem

```
>>-processItem--(--id--)----------------------------------------><
```

Selects the menu item *id*. This causes a WM_COMMAND to be sent to the window owning the menu.

Chapter 8. OLE Automation

OLE (Object Linking and Embedding) automation is a subset of COM (Component Object Model). These technologies were first developed on Windows and are deeply embedded in the Windows operating system. Although COM is not tied to the Windows operating system, in practice it is not seen much on other operating systems. Because of this, the ooRexx classes supporting OLE Automation are currently Windows only classes.

8.1. Overview of OLE Automation

OLE (Object Linking and Embedding) is an implementation of COM (Component Object Model). OLE automation makes it possible for one application to manipulate objects implemented in another application, or to expose objects so they can be manipulated. ooRexx provides two classes, OLEObject and OLEVariant that allow the programmer to take advantage of this ability to manipulate objects that are exposed as OLE objects.

An automation client is an application that can manipulate exposed objects belonging to another application. An automation server is an application that exposes the objects. The OLEObject class enables Rexx to be an OLE automation client. In addition, some automation servers have an event mechanism that allows them to invoke methods in the OLE automation client. The OLEObject class also supports this mechanism.

Applications can provide OLE objects, and OLE objects that support automation can be used by a Rexx script to remotely control the object through the supplied methods. This lets you write a Rexx script that, for example, starts a Web browser, navigates to a certain page, and changes the display mode of the browser.

Every application that supports OLE places a unique identifier in the registry. This identifier is called the class ID (CLSID) of the OLE object. It consists of several hexadecimal numbers separated by the minus symbol.

Example: CLSID of Microsoft® Internet Explorer (Version 5.00.2014.0216):

```
"{0002DF01-0000-0000-C000-000000000046}"
```

The CLSID number can prove inconvenient when you want to create or access a certain object, so a corresponding easy-to-remember entry is provided in the registry, and this entry is mapped to the CLSID. This entry is called the ProgID (the program ID), and is a string containing words separated by periods.

Example: ProgID of Microsoft Internet Explorer: `"InternetExplorer.Application"`

To find the ProgID of an application, you can use the sample script `OLEINFO.REX` or the Microsoft OLEViewer, you can consult the documentation of the application, or you can search the registry manually.

Several sample programs are provided in the Open Object Rexx installation directory under `samples\ole`

- The apps directory contains examples of how to use Rexx to remote-control other applications.
- The oleinfo directory is a sample Rexx application that can be used to browse through the information an OLE object provides.

- In the adsi directory there are eight examples of how to use the Active Directory Services Interface with the Rexx OLE interface.
- The methinfo directory contains a very basic example of how to access the information an OLE object provides.
- Finally, the wmi directory contains five examples of how to work with the Windows Management Instrumentation.

8.2. OLE Events

Some, but not all, OLE automation objects support *events*. The most prevalent use of OLE is for the automation server (the OLE object) to implement methods that the automation client (the ooRexx OLEObject) invokes. However, it is also possible for the automation client (the ooRexx OLEOjbect) to implement methods that the automation server (the OLE object) invokes.

The methods that the automation client implements are called event methods and the automation server that suports event methods is called a *connectable* object. The connectable object defines the events it supports by defining the name of the method and its arugments, but does not implement the method. Rather the automation client implements the method. The client asks the automation server to make a *connection*. If the connection is established, then from that point on whenever one of the defined events occurs, the server invokes the event method on the connected client.

In effect, what is happening is that the automation server is *notifying* the automation client that some event has occurred and giving the client a chance to react to the event. Any number of clients can be connected to the same connectable object at the same time. Each client will recieve a notification for any event they are interested in. There is no need for the client to receive notifications for evey event. When the client is not interested in an event, the client simply does not implement a method for that event.

The original implementation of OLEObject allowed the Rexx programmer to use events in this way: The programmer defines and implements a subclass of the OLEObject. Within the subclass, the programmer defines and implements the event methods for which she wants to recieve notifications. The progammer has the client make a connection to the automation server at the time the OLEObject object is instantiated by using the `WITHEVENTS` keyword for the *events* argument. If the `WITHEVENTS` keyword is not used during instantiation, then no event connection can be made.

This is relatively easy to understand and a simple example should make this clear. In the following, rather than create a new OLEObject, the programmer defines a subclass of the OLEObject, a WatchedIE class. The WatcheID object is instantiated with events. This tells the OLEObject to make an event connection, if possible. In the subclass, the programmer implements the events he is interested in receiving notifications for.

Example:

```
-- Instantiate an instance of the subclassed OLEObject
myIE = .WatchedIE~new("InternetExplorer.Application", "WITHEVENTS")
...

-- This class is derived from OLEObject and contains several methods
-- that will be called when certain events take place.
::class 'WatchedIE' subclass OLEObject
```

```
-- This is an event of the Internet Explorer */
::method titleChange
  use arg Text
  say "The title has changed to:" text

-- This is an event of the Internet Explorer
::method beforeNavigate2
  use arg pDisp, URL, Flags, TargetFrameName, PostData, Headers, Cancel
  ...

-- This is an event of the Internet Explorer */
::method onQuit
  ...
```

However, the process described above only allows using events with OLEObject objects that are directly instantiated by the programmer. There are a number of OLE objects that support events, where the OLEObject object is not instantiated by the programmer, but rather is returned from a method invocation. Prior to ooRexx 4.0.0, events could not be used with these objects. In 4.0.0, methods were added to the OLEObject class that allow using events with any OLE object that supports events.

This second process works this way: With an already instantiated object, the programmer can create method objects for any events of interest and use the addEventMethod() method to add the method to the instantiated object. Then the connectEvents() method is used to connect the automation client (the instantiated object in this case) to the connectable OLE automation server.

The following example demonstrates this second process that is available in ooRexx 4.0.0 and onwards.

Example:

```
wordApp = .OLEObject~new("Word.Application")
wordApp~visible = .true
document = wordApp~documents~Add

-- Use the isConnectable method to ensure the object supports connections.
if document~isConnectable then do

  -- Create a method for the OLEEvent_Close event. From the Word documentation
  --and experimentation, it is known that this event has no arguments.
  mArray = .array~new
  mArray[1] = 'say "Received the OLEEvent_Close event."'
  mArray[2] = 'say "  Event has" arg() "parameters."'
  mArray[3] = 'say'

  mClose = .Method~new("OLEEvent_Close", mArray)

  -- Now add this method to the document object.
  document~addEventMethod("OLEEvent_Close", mClose)

  -- Tell the object to make an events connection.
  document~connectEvents
end
```

The preceding example brings up one last point that is important to note when defining event methods. It is possible for an event method to have the same name as a normal invocation method of the OLE object. This gives rise to this scenario:

The programmer adds an ooRexx event method to the OLEObject with that name. Then the programmer tries to invoke the normal method. However, the invocation will no longer get forwarded to the unknown() method. Instead the event method by the same name is invoked. This is the case in the above example. The document object has a `close`() method that is used to close the document. The document also has the `close`() event method that is used to notify clients that the document is about to close.

To prevent this scenario, when an event method of an OLE object has the same name as a normal method name, the programmer must prepend `OLEEvent_` to the method name. The implementation of OLEObject assumes the programer has done so. If the programmer does not name the event methods using this convention, the results are unpredicatable.

Note that *only* the event method names that have matching normal event names can be prepended with the OLEEvent_prefix. Other event names must not have the prefix. One way to check for this is to use the getKnownEvents() method. This method will return the correct names for all events the OLE object supports.

Example:

This example is a compelete working program. To run it, Microsoft OutLook must be installed. The program demonstrates some of the various methods of the OLEObject that deal with events. The interface to the program is simplistic, but workable.

Once the program starts, the user controls it by creating specific named files in working directory of the program. This could be done for example using echo:

```
echo " " > stop.monitor
```

The three specific file names are: `stop.monitor`, `pause.monitor`, and `restart.monitor`. The stop file ends the program. The pause file has the program stop monitoring for new mail, but keep running. The restart file has the program restart monitoring from the paused state.

```
/* Monitor OutLook for new mail */
  say; say; say 'ooRexx Mail Monitor version 1.0.0'

  outLook = .oleObject~new("Outlook.Application")

  inboxID = outLook~getConstant(olFolderInBox)
  inboxItems = outLook~getNameSpace("MAPI")~getDefaultFolder(inboxID)~items

  if \ inboxItems~isConnectable then do
    say 'Inbox items is NOT connectable, quitting'
    return 99
  end

  inboxItems~addEventMethod("ItemAdd", .methods~printNewMail)
  inboxItems~connectEvents

  if \ inboxItems~isConnected then do
    say 'Error connecting to inbox events, quitting'
    return 99
```

```
  end

  monitor = .Monitor~new
  say 'ooRexx Mail Monitor - monitoring ...'
  do while monitor~isActive
    j = SysSleep(1)
    status = monitor~getStatus

    select
      when status == 'disconnect' then do
        inboxItems~disconnectEvents
        say 'ooRexx Mail Monitor - paused ...'
      end
      when status == "reconnect" then do
        inboxItems~connectEvents
        say 'ooRexx Mail Monitor - monitoring ...'
      end
      otherwise do
        nop
      end
    end
    -- End select
  end
  say 'ooRexx Mail Monitor version 1.0.0 ended'

return 0

::method printNewMail unguarded
  use arg mailItem
  say 'You have mail'
  say 'Subject:' mailItem~subject

::class 'Monitor'
::method init
  expose state active

  state = 'continue'
  active = .true
  j = SysFileDelete('stop.monitor')
  j = SysFileDelete('pause.monitor')
  j = SysFileDelete('restart.monitor')

::method isActive
  expose active
  return active

::method getStatus
  expose state active

  if SysIsFile('stop.monitor') then do
    j = SysFileDelete('stop.monitor')
    active = .false
    state = 'quit'
```

```
  return state
end

if SysIsFile('pause.monitor') then do
  j = SysFileDelete('pause.monitor')
  if state == "paused" then return "continue"

  if state \== 'quit' then do
    state = "paused"
    return 'disconnect'
  end
end

if SysIsFile('restart.monitor') then do
  j = SysFileDelete('restart.monitor')
  if state == 'continue' then return state

  if state \== 'quit' then do
    state = 'continue'
    return 'reconnect'
  end
end

return 'continue'
```

8.3. The OLEObject Class

The OLEObject class is a built-in class. No `::requires` directive is needed to use the class.

Methods available to the OLEObject class:

new (Class method)
addEventMethod
connectEvents
disconnectEvents
dispatch
getConstant
getKnownEvents
getKnownMethods
getObject(Class method)
getOutParameters
isConnected
isConnectable
removeEvents
removeEventHandler
unknown

Note: It is somewhat useful to think of the Rexx OLEObject object as a proxy to the real OLE object. The real OLE object has its own methods. Which methods it has is dependent on its individual implementation. These methods are then accessed transparently through the unknown() method mechanism of the OLEObject by invoking a method of the same name on the OLEObject object.

8.3.1. new (Class method)

```
>>--new(--classID--+-----------+--+--------------+--)------------------><
                   +-,-events--+  +-,-getObject--+
```

Instantiates a new OLEObject as a proxy for a COM / OLE object with the specified *classID* (the `ProgID` or `CLSID`). If the COM / OLE object can not be accessed or created, an error will be raised. See the list of OLE specific errors in the Open Object Rexx Reference document.

Arguments:

The arguments are:

classID

The `ProgID` or `CLSID` that identifies the COM / OLE object to proxy for.

events

Controls how the event methods of the COM / OLE object are handled:

If the argument is omitted completely, then no action concerning the event methods is taken.

If the argument is NOEVENTS then the COM / OLE object is queried to determine if it is a connectable object. If it is, an internal table is constructed listing all the event methods. But the object is not connected.

If the argument is WITHEVENTS then the COM / OLE object is queried to determine if it is a connectable object. If it is, an internal table is constructed listing all the event methods, and an event connection is established.

getObject

A flag asking to first try to get an already instantiated OLE object, rather than instantiate a new object. Some OLE automation servers register themselves with the operating system when an object is first created, but not all do. If this flag is true, then the OLEObject first tries to proxy for an already running OLE / COM object. If this fails, then a new OLE / COM object is instantiated.

If the flag is omitted, or `.false` then no attempt to look for an already running OLE / COM object is made.

Example:

```
myOLEObject = .OLEObject~new("InternetExplorer.Application")
```

8.3.2. dispatch

```
>>-dispatch(methodname--+-------------+--)----------------------><
                        |  +------+   |
                        |  V      |   |
                        +----,arg-+--+
```

Dispatches a method with the optionally supplied arguments.

8.3.3. getConstant

```
>>-getConstant(-+--------------+-)----------------------------><
                +-ConstantName-+
```

Retrieves the value of a constant that is associated with this OLE object. If no constant of that name exists, the .Nil object will be returned. You can also omit the name of the constant; this returns a stem with all known constants and their values. In this case the constant names will be prefixed with a "!" symbol.

Example 1:

```
myExcel = .OLEObject~new("Excel.Application")
say "xlCenter has the value" myExcel~getConstant("xlCenter")
myExcel~quit
exit
```

Possible output:

```
xlCenter has the value -4108
```

Example 2:

```
myExcel = .OLEObject~new("Excel.Application")
constants. = myExcel~getConstant
myExcel~quit

do i over constants.
  say i"="constants.i
end
```

Possible output:

```
!XLFORMULA=5
!XLMOVE=2
!XLTEXTMAC=19
...
```

8.3.4. getKnownEvents

```
>>-getKnownEvents------------------------------------------><
```

Returns a stem with information on the events that the connectable OLE object supports. It collects this information from the type library of the OLE object. A type library provides the names, types, and arguments of the provided methods. The OLEObject object does not need to be currently connected to connectable OLE object.

This method will return the event methods for any connectable object. Prior to ooRexx 4.0.0, only OLEObjects created directly, and created with the 'event' flag (WITHEVENTS or NOEVENTS) would return any known events. This fact had not been fully documented. Therefore, if the user did not create the OLEObject correctly, .nil would be returned for objects that did support event connections.

In 4.0.0, the behavior is fixed (or enhanced depending on the point of view) so that the known events are returned for all connectable objects under all circumstances.

The stem provides the following information:

Table 8-1. Stem Information

stem.0	The number of events.
stem.n.!NAME	Name of n-th event.
stem.n.!DOC	Description of n-th event (if available).
stem.n.!PARAMS.0	Number of parameters for n-th event.
stem.n.!PARAMS.i.!NAME	Name of i-th parameter of n-th event.
stem.n.!PARAMS.i.!TYPE	Type of i-th parameter of n-th event.
stem.n.!PARAMS.i.!FLAGS	Flags of i-th parameter of n-th event; can be "in", "out", "opt", or any combination of these.

If no information is available, the .NIL object is returned. This indicates that the OLE object does support events.

Example script:

```
myIE = .OLEObject~new("InternetExplorer.Application","NOEVENTS")
events. = myIE~getKnownEvents

if events. == .nil then
  say "Sorry, this object does not have any events."
else do
  say "The following events may occur:"
  do i = 1 to events.0
    say events.i.!NAME
  end
end

exit
```

Sample output:

```
The following events may occur:
ONTHEATERMODE
ONFULLSCREEN
ONSTATUSBAR
...
```

For an example of how to use events, see examples `samples\ole\apps\samp12.rex` and `samples\ole\apps\samp13.rex`. The samples directory is installed as part of the normal Windows installation.

8.3.5. connectEvents

```
>>--connectEvents-----------------------------------------------><
```

The connectEvents() method is used to connect the instantiated automation client (the OLEObject subclass object) to the automation server (the OLE object) at any time. The method returns `.true` if the connection was made, otherwise `.false`. Remember, not all OLE objects support events. The programmer can determine if the OLE object supports events by using the isConnectable() method.

Example:

```
wordApp = .OLEObject~new("Word.Application")
wordApp~visible = .true
document = wordApp~documents~Add

wordApp~connectEvents
```

8.3.6. isConnected

```
>>--isConnected-------------------------------------------------><
```

Determines if the OLEObject instance is currently connected to a connectable OLE automation server. Returns `.true` if the instance is connected and `.false` if not.

Example:

```
wordObj = .oleObject~new("Word.Application", "WITHEVENTS")
if wordObj~isConnected then do
  ...
end
else do
  ...
end
```

8.3.7. isConnectable

```
>>--isConnectable------------------------------------------------><
```

Determines if the OLE object is a connectable object. In other words, does the OLE object support event methods and will it accept connections at this time. Not all OLE objects support events, probably the majority do not support events. This method returns `.true` if the object is connectable, otherwise `.false`.

Example:

```
outLook = .oleObject~new("Outlook.Application")

-- This searches all folders for the 'Mailbox - .. ' folder.  Which is
-- usually the default folder in a business installation of Outlook.
nameSpace = outLook~getNameSpace('MAPI')
folders = nameSpace~folders
```

```
do i = 1 to folders~count
  if folders~item(i)~name~caselessPos("Mailbox") <> 0 then do
    theMailBoxFolder = folders~item(i)
    leave
  end
end

-- Now that we have the Mailbox folder, get the collection of folders that
-- are contained in the Mailbox folder.
folders = theMailBoxFolder~folders

if folders~isConnectable then do
  -- Add event methods to the folders object.
  ...
end
```

8.3.8. disconnectEvents

```
>>--disconnectEvents-----------------------------------------><
```

This method disconnects from the connectable OLE object. The method returns `.false` if there is not a current connection, otherwise `.true`. After this method is called, the OLE object will no longer invoke the event methods, in effect stopping event notifications.

The internal data structures used to manage events remain intact. The programmer can use the connectEvents() method to reconnect at any time. Since the internal data structures do not need to be rebuilt, this will save some small amount of processor time. To completely remove the internal data structures use the removeEventHandler() method.

Example:

This example shows some code snippets from a program that monitors the user's inbox in OutLook. When a new mail item arrives, the user is notified. The interface for the program allows the user to turn off the notifications when she wants, then turn them back on later. When the interface signals the program to stop the notifications, the program simply disconnects the events from the OutLook object. When the user wants to resume notifications, the program reconnects the events.

```
outLook = .oleObject~new("Outlook.Application")
inboxID = outLook~getConstant(olFolderInBox)
inboxItems = outLook~getNameSpace("MAPI")~getDefaultFolder(inboxID)~items

...

inboxItems~addEventMethod("ItemAdd", .methods~printNewMail)
inboxItems~connectEvents

...

  select
    when status == 'disconnect' then do
      inboxItems~disconnectEvents
```

```
      say 'ooRexx Mail Monitor - paused ...'
    end
    when status == "reconnect" then do
      inboxItems~connectEvents
      say 'ooRexx Mail Monitor - monitoring ...'
    end
    otherwise do
      nop
    end
  end
  -- End select
```

8.3.9. removeEventHandler

```
>>--removeEventHandler-----------------------------------------><
```

Removes the event handler and cleans up the internal data structures used to manage events. No event methods will be invoked after this method is called. See the disconnectEvents() method for a way to temporarily disconnect from event notifications.

Example:

```
inboxItems~removeEventHandler
inboxItems~removeEventMethod("ItemAdd")
```

8.3.10. addEventMethod

```
>>--addEventMethod(--name-,-methodObject--)--------------------><
```

`addEventMethod` adds a new method to this object's collection of methods. The *name* argument specifies the name of the new method and the *methodObject* argument defines the method. The acceptable values for *methodObject* are the same as those for the second argument to the `setMethod` method of the `.Object` class. That is, it can be a method object, a string containing a method source line, or an array of strings containgin individual method source lines.

The purpose of this method is to add an event method to a OLEObject after the object has been instantiated. See the OLE Events section for more details on events.

Example:

Note that in this example, the printNewMail method is defined as a floating method. See the documentation for the `.methods` directory in the Open Object Rexx Reference book for more details if needed.

```
inboxID = outLook~getConstant(olFolderInBox)
inboxItems = outLook~getNameSpace("MAPI")~getDefaultFolder(inboxID)~items

inboxItems~addEventMethod("ItemAdd", .methods~printNewMail)
```

```
...

::method printNewMail unguarded
  use arg mailItem
  say 'You have mail'
  say 'Subject:' mailItem~subject
```

8.3.11. removeEventMethod

```
>>--removeEventMethod(--name--)----------------------------------><
```

Removes the event method with the specified *name* that has been previously added to this object by the addEventMethod() method.

Example:

```
  inboxID = outLook~getConstant(olFolderInBox)
  inboxItems = outLook~getNameSpace("MAPI")~getDefaultFolder(inboxID)~items

  inboxItems~addEventMethod("ItemAdd", .methods~printNewMail)
  inboxItems~connectEvents

  ...

::method doneWithItemEvents private
  expose inboxItems

  inboxItems~removeEventHandler
  inboxItems~removeEventMethod("ItemAdd")
```

8.3.12. getKnownMethods

```
>>-getKnownMethods-------------------------------------------><
```

Returns a stem with information on the methods that the OLE object supplies. It collects this information from the type library of the object. A type library provides the names, types, and arguments of the provided methods. Parts of the supplied information have only informational character as you cannot use them directly.

The stem provides the following information:

Table 8-2. Stem Information

stem.0	The number of methods.
stem.!LIBNAME	Name of the type library that describes this object.
stem.!LIBDOC	A help string describing the type library. Only set when the string is available.

stem.!COCLASSNAME	COM class name of this object.
stem.!COCLASSDOC	A string describing the COM class. Only set when the string is supplied by the type library.
stem.n.!NAME	The name of the n-th method.
stem.n.!DOC	A help string for the n-th method. If this information is not supplied in the type library this value will not be set.

stem.n.!INVKIND	A number that represents the invocation kind of the method: 1 = normal method call, 2 = `property get`, 4 = `property put`. A normal method call is used with brackets; for a `property get` only the name is to be specified; and a `property set` uses the "=" symbol, as in these examples:
	`object~methodCall(a,b,c)`
	`object~propertyPut="Hello" say object~propertyGet`

stem.n.!RETTYPE	The return type of the n-th method. The return type will be automatically converted to a Rexx object (see Type Conversion in the description of the UNKNOWN method of the OLEObject class).
stem.n.!MEMID	The MemberID of the n-th method. This is only used internally to call the method.
stem.n.!PARAMS.0	The number of parameters of the n-th method.
stem.n.!PARAMS.i.!NAME	The name of the i-th parameter of the n-th method.
stem.n.!PARAMS.i.!TYPE	The type of the i-th parameter of the n-th method.
stem.n.!PARAMS.i.!FLAGS	The flags of the i-th parameter of the n-th method; can be "in", "out", "opt", or any combination of these (for example: "[in, opt]").

If no information is available, the .NIL object is returned.

Note that it is not required that an OLE object supply a type library. The methods of OLE objects that do not supply a type library can still be invoked by name, but there is no way for `getKnownMethods` to look up the methods. To use these OLE objects the Rexx programmer would need to consult the documentation for the OLE object.

In addition all OLE objects have methods that can only be used internally. There are mechanisms to *hide* these methods from the user, because they can not be used by the automation client. It is possible that these are not hidden properly and will be listed when using `getKnownMethods`. The following methods can not be used by an instance of the OLEObject:

AddRef
GetTypeInfoCount
GetTypeInfo
GetIDsOfNames
QueryInterface
Release

Example script:

```
myOLEObject = .OLEObject~new("InternetExplorer.Application")
methods. = myOLEObject~getKnownMethods

if methods. == .nil then
  say "Sorry, no information on the methods available!"
else do
  say "The following methods are available to this OLE object:"
  do i = 1 to methods.0
    say methods.i.!NAME
  end
end

exit
```

Sample output:

```
The following methods are available to this OLE object:
GoBack
GoForward
GoHome
...
```

8.3.13. getObject (Class method)

```
>>-getObject(Moniker-+--------+-)------------------------------><
                     +-,class-+
```

This is a class method that allows you to obtain an OLE object through the use of a moniker. A moniker is a string and is similar to a nickname. Monikers are used by OLE to connect to and activate OLE objects. OLE returns the object that the moniker identifies.

If the object is already running, OLE will find it in memory. If the object is stored passively on disk, OLE will locate a server for the object, run the server, and have the server bring the object into the running state. This makes monikers very easy for the automation client to use. OLE hides all the details from the client. However, since the OLEObject also hides all the details when a new OLE object is instantiated, for the Rexx programmer there is not much difference between using the `getObject` method and using the `new` method.

Note that file system names are monikers. Therefore, if a file is assoicated with an application that is an OLE automation server, a new OLE object can be instantiated by using the file name as the moniker. Obviously, this is not true of every file. It is true for files like `.xls` and `.doc` files, for example, because Word and Excel are OLE automation applications.

The optional *class* argument can be used to specify a subclass of OLEObject, and can be used to obtain an OLE object that supports events (the'`WITHEVENTS`' option will be used in this case). This method is similar to the `new` method where the programmer supplies a ProgID or CLSID. In this case the programmer supplies a moniker.

Example:

```
/* create a Word.Document by opening a certain file */
myOLEObject = .OLEObject~GetObject("C:\DOCS\HELLOWORLD.DOC")
```

8.3.14. getOutParameters

```
>>-getOutParameters-------------------------------------------><
```

Returns an array containing the results of the single `out` parameters of the OLE object, or the .NIL object if it does not have any. `Out` parameters are arguments to the OLE object that are filled in by the OLE object. As this is not possible in Rexx due to data encapsulation, the results are placed in the array mentioned above.

Example:

Consider an OLE object method with the following signature:

```
aMethod([in] A, [in] B, [out] sumAB)
```

The resulting `out` parameter of the method invocation will be placed in the `out` array at position one; the "normal" return value gets processed as usual. In this case the method will return the .NIL object:

```
resultTest = myOLEObject~aMethod(1, 2, .NIL)
say "Invocation result  :" resultTest
say "Result in out array:" myOLEObject~getOutParameters~at(1)
```

The output of this sample script will be:

```
The NIL object
3
```

`Out` parameters are placed in the `out` array in order from left to right. If the above OLE method looked like this:

```
aMethod([in] A, [in] B, [out] sumAB, [out] productAB),
```

then the `out` array would contain the sum of A and B at position one, and the product at position two.

8.3.15. unknown

```
>>-unknown(messageName--+----------------+--)------------------><
                        +--,messageArgs--+
```

The `unknown` message is the central mechanism through which methods of the OLE object are called.

For further information on the details on how an `unknown` method works, see *Defining an unknown Method* in the Open Object Rexx Reference.

The programmer can invoke the methods of the real OLE object by simply invoking the methods on the the Rexx (proxy) OLEObject object like this:

```
myOLEObject~OLEMethodName
```

This calls the method "OLEMethodName" of the real OLE object for any message (method) *that does not exist* in the Rexx OLEObject object through the *unknown* method mechanism. The implementation for the unknown() method in the OLEObject class does this by dispatching the method call to the real OLE object.

This presents a problem when an OLE object has a method with a name that is identical to a method defined for the OLEObject object. When this situation happens, the programmer has two choices.

One choice is for the programmer to call the unknown method directly. E.g., take an OLE object that has the method copy used to copy something from a source to a destination. Since copy is a method of the Object class, the copy method of the OLE object is a method name already defined for the OLEObject. The programmer can invoke the unknown method directly, like this:

```
msgArgs = .array~of("C:\open\myFile.txt", "C:\processDir\")
val = myOLEObject~unknown("copy", msgArgs)
```

This causes the implementation of the unknown() method in the OLEObject object to invoke the copy method of the OLE object with the arguments of C:\open\myFile.txt and C:\processDir\.

The other thing the Rexx programmer can do is use the dispatch() method. Since, in OLE automation terms, the act of invoking a method on the OLE object is commonly referred to as *dispatching* a message to the OLE object, this may make the code a little easier to understand. In the above example the dispatch method would be used like this:

```
val = myOLEObject~dispatch("copy", "C:\open\myFile.txt", "C:\processDir\")
```

8.3.16. Type Conversion

Unlike Rexx, OLE uses strict typing of data. Conversion to and from these types is done automatically, if conversion is possible. OLE types are called variants, because they are stored in one structure that gets flagged with the type it represents. The following is a list of all variant types valid for use with OLE Automation and the Rexx objects that they are converted from or into.

Table 8-3. OLE/Rexx Types

VARIANT type	Rexx object
VT_EMPTY	.NIL
VT_NULL	.NIL
VT_ERROR	.NIL
VT_I1	Rexx string (a whole number)
VT_I2	Rexx string (a whole number)
VT_I4	Rexx string (a whole number)
VT_I8	Rexx string (a whole number)
VT_UI1	Rexx string (a whole, positive number)
VT_UI2	Rexx string (a whole, positive number)
VT_UI4	Rexx string (a whole, positive number)
VT_UI8	Rexx string (a whole, positive number)

VARIANT type	Rexx object
VT_INT	Rexx string (a whole number)
VT_UINT	Rexx string (a whole, positive number)
VT_DECIMAL	Rexx string (a decimal number)
VT_R4	Rexx string (a real number)
VT_R8	Rexx string (a real number)
VT_CY	Rexx string (currency, a fixed-point number with 15 digits to the left of the decimal point and 4 digits to the right)
VT_DATE	Rexx string (a date)
VT_BSTR	Rexx string
VT_DISPATCH Rexx	OLEObject
VT_BOOL	.TRUE or .FALSE
VT_VARIANT	Any Rexx object that can be represented as a VARIANT
VT_UNKNOWN	OLEObject
VT_ARRAY *	Rexx Array
VT_BYREF *	Any Rexx object

* VT_ARRY and VT_BYREF are combined with any of the other variant types and never used alone. VT_ARRAY and another variant type are used for a SAFEARRAY datatype, an array of the other variant type. VT_BYREF and another variant type are used to pass the other variant type to or from an OLE object by reference. The programmer need not worry about this passing by reference, the OLE support handles this transparently.

8.4. The Windows OLEVariant Class

The OLEVariant class enhances the support for OLE Automation provided by the OLEObject class and is used in conjunction with that class. An OLEVariant object is used as a parameter in a method call of an OLEObject object. In the OLEObject's role as a proxy for a OLE / COM object, the parameters in method calls are forwarded on to the actual OLE / COM object. (OLE / COM objects will be referred to simply as COM objects.)

There are two areas where the OLEVariant adds to the capabilities of OLEObject method calls.

- Parameters forwarded on to COM objects must be converted to and from the proper datatypes. This conversion is done automatically (see OLEObject Type Conversion.) Occasionally this automatic conversion is incorrect. The OLEVariant allows the ooRexx programmer to override the automatic conversion by specifying how the conversion should be done.
- COM objects can return data to the caller in "out" parameters ([OUT] parameters.) The OLEVariant can be used to transport this returned data back to the calling ooRexx program.

In general, the automatic type conversion in the OLE support uses type libraries to determine how to format the parameters being sent to an OLE object in a method call. The information in a type library specifies the variant type an ooRexx object, used as a parameter, needs to be converted to. Type libraries also detail how a parameter is to be flagged when it is sent to the COM object.

However, COM objects are not required to supply type libraries. When there is no type library, ooRexx uses an educated guess to determine this information. On rare occasions this guess is wrong and the method call fails. In theses cases, if the ooRexx programmer knows what the correct information is, the programmer can use an OLEVariant to specify this information. The programmer can supply either or both of these pieces of information by specifying the variant type for the converted ooRexx object and the parameter flags.

The following is a real world example where the automatic conversion in the OLE support does not work and shows how the OLEVariant is used to specify the correct conversion. The snippet comes from code to automate a CICS client. In this case the variant type that the ooRexx object needs to be converted to is specified. The parameter flags are omitted. The fourth parameter to the ~link method call is the parameter where the default conversion was failing.

```
connect  = .OLEObject~new("Ccl.Connect")
flow     = .OLEObject~new("Ccl.Flow")
buffer   = .OLEObject~new("Ccl.Buffer")

uow      = .OLEVariant~New(.nil, VT_DISPATCH)
...
connect~link(flow, "FOO", buffer, uow)
```

> **Note:** It is extremely rare that the OLE support fails to do the right thing with its automatic conversion. 99.999% of the time the ooRexx programmer does not need to use an OLEVariant object to specify the type conversion. This use of the OLEVariant is provided for those few times when it is necessary to override the default conversion. Furthermore, if the ooRexx programmer does not know what variant type to specify, this usage will not be much help. Normally the ooRexx programmer would know what type to specify through the documentation for the COM class the programmer is using.

The next example shows how the OLEVariant can be used to transport the data returned in an "out" parameter back to the calling ooRexx program. This usage will be more common and does not require that the ooRexx have a lot of detailed knowledge of the COM object. Obviously, the programmer does need to know that the parameter is an out parameter. This example comes from updating a MS Access database where the number of records affected by the update is returned in an "out" parameter. Here the out parameter is the second parameter in the ~execute method call.

```
sql = "update myTable set id=id*3 where id > 7"
param = .OLEVariant~new(0)
conn~execute(sql, param)
count = param~!varValue_
say count "record(s) were affected."
```

Finally an example where the OLE support does not use the correct parameter flags for the method call. The Windows Management Instrumentation, Win32_Process COM class does not supply a type library. The fourth parameter in the ~create method call is an "out" parameter. That information is known by the ooRexx programmer through the documentation of the class. However, without a type library, ooRexx

has no way to know that. Here the variant type specification is omitted (signaling ooRexx to go ahead and use its automatic conversion) and the parameter flags are specified. Since this an out parameter, the OLEVariant object is also used to transport the returned data back to the calling program.

```
objProcess = .oleObject~getObject("WinMgmts:Win32_Process")

param = .OLEVariant~new( 0, , "IN,OUT" )
ret = objProcess~create('notepad.exe', .nil, .nil, param)
if ret == 0 then do
  pid = param~!varValue_
  say 'The notepad process was created and its PID is' pid
end
```

Methods available to the OLEVariant class

new
!varValue_
!varValue_=
!varType_
!varType_=
!paramFlags_
!paramFlags_=

> **Note:** A possible future enhancement of the OLEVariant class requires that its method names be unique, which is the reason for the method name style. In normal usage the ooRexx programmer would only be concerned with the new and the !varValue methods. Therefore the slightly unorthodox method names should not present a problem.

8.4.1. new Class method

```
>>-new(valueObject-+----------+--+-------------+-)-------------><
                   +-,varType-+  +-,paramFlags-+
```

Instantiates a new OLEVariant object to be used as a parameter in an OLEObject method call. The first argument is the ooRexx object to be converted to a variant type for the method call. It is the object to be used in the method call. This argument is required. The varType and paramFlags arguments are optional.

The varType argument is used to specify the type of the variant that the valueObject is to be converted to. If this argument is omitted or is .nil then ooRexx will use the default conversion for the valueObject. If it is not omitted it must be a valid OLE Automation variant type and ooRexx will attempt to convert the valueObject to this variant type.

The valid variant type symbols are listed in Table OLE/Rexx Types. In addition any of those symbols can be combined with the VT_BYREF or the VT_ARRAY symbol. When symbols are combined a comma is used to separate the two symbols. This of course necessitates that the argument be quoted. Case does not matter for this argument. For example vt_bool, VT_bool, or VT_BOOL are all treated the same.

The paramFlags argument is used to specify the flags for the parameter. The flags are separated by a comma. Although any combination of valid PARAMFLAGS as defined for OLE Automation will be

accepted, in practice the ooRexx programmer will probably only need to use "IN,OUT" for this argument.

The PARAMFLAGS defined for OLE Automation:

PARAMFLAG_NONE
PARAMFLAG_FIN
PARAMFLAG_FOUT
PARAMFLAG_FLCID
PARAMFLAG_FRETVAL
PARAMFLAG_FOPT
PARAMFLAG_FHASDEFAULT
PARAMFLAG_FHASCUSTDATA

The ooRexx programmer should only use the last portion of the symbol. I.e., NONE, IN, OUT, LCID, RETVAL, OPT, HASDEFAULT, or HASCUSTOMDATA. Case also does not matter for this argument and "in,out" is equivalent to "IN,OUT"

If the paramFlags argument is omitted or .nil, (the normal case,) ooRexx will determine the flags for the parameter through its default mechanism. If the argument is not omitted, ooRexx will use the specified flags unconditionally.

> **Note:** If either the varType or paramFlags arguments are used, and not the .nil object, they must be valid variant types or param flags for OLE Automation. If they are not valid, a syntax error will be raised.

```
manager = .oleObject~new("com.sun.star.ServiceManager", "WITHEVENTS")
cf = manager~createInstance("com.sun.star.reflection.CoreReflection")
...
classSize = .cf~forName("com.sun.star.awt.Size")

param = .OLEVariant~new(.nil, "VT_DISPATCH,VT_BYREF", "IN,OUT")

retVal = classSize~createObject(param)
```

8.4.2. !VARVALUE_

```
>>-!VARVALUE_-------------------------------------------------><
```

Returns the value object set within an instance of an OLEVariant. If the parameter in a COM method call that the OLEVariant was used for is an "out" parameter, than the value object of the instance will be the data returned by the COM object. Otherwise, the value object is that set by the ooRexx programmer.

```
manager = .oleObject~new("com.sun.star.ServiceManager", "WITHEVENTS")
cf = manager~createInstance("com.sun.star.reflection.CoreReflection")
...
classSize = .cf~forName("com.sun.star.awt.Size")

param = .OLEVariant~new(.nil, "VT_DISPATCH,VT_BYREF", "IN,OUT")

retVal = classSize~createObject(param)
```

```
size = param~!varValue_
```

8.4.3. !VARVALUE_=

```
>>-!VARVALUE_=-------------------------------------------------><
```

Sets the value object an instance of an OLEVariant contains.

8.4.4. !VARTYPE_

```
>>-!VARTYPE_---------------------------------------------------><
```

Returns the variant type specification of the OLEVariant instance.

8.4.5. !VARTYPE_=

```
>>-!VARTYPE_=-------------------------------------------------><
```

Sets the variant type specification of an OLEVariant instance. This serves the same purpose as the second argument to the new method and follows the same rules as specified in the documentation of the new method. I.e., the value must be a valid variant type used in OLE Automation, or .nil. If not a syntax error is raised.

8.4.6. !PARAMFLAGS_

```
>>-!PARAMFLAGS_------------------------------------------------><
```

Returns the parameter flags specification of the OLEVariant instance.

8.4.7. !PARAMFLAGS_=

```
>>-!PARAMFLAGS_=----------------------------------------------><
```

Sets the flags specification of an OLEVariant instance. This serves the same purpose as the third argument to the new method and follows the same rules as specified in the documentation of the new method. I.e., the value must be a valid combination of PARAMFLAG types as documented for use in OLE Automation, or .nil. If not a syntax error is raised.

Chapter 9. Windows Scripting Host Engine

This chapter describes the use of Object Rexx as a Windows Scripting Host (WSH) engine.

9.1. Object Rexx as a Windows Scripting Host Engine

Object Rexx is automatically enabled as an engine for Windows Scripting Host at installation. This chapter gives a brief description of WSH and how Object Rexx interacts with it, and shows you how you can best use this feature.

The easiest part of this feature to understand and to become immediately productive with is its ability to use Object Rexx as a scripting language for Microsoft's Web browser, Internet Explorer. To go quickly to using this technique, see Invocation by the Browser.

9.1.1. Windows Scripting Host Overview

Windows Scripting Host (WSH) is a unified scripting environment for all Microsoft products. It is usable by any macro language that follows its specification. WSH is the mechanism that allows users to customize and dynamically control the products that support its hosting standard.

The Windows Scripting Host engine for Object Rexx enables users to drive Microsoft's products, notably Internet Explorer. Other products that can be driven include the components of the Office suite, like Word, Excel, and so on.

The difference between WSH and the OLE support that Object Rexx provides is the context in which the script resides. OLE scripts are exterior to the product, and WSH scripts can be embedded in the files that the product uses. The advantage of embedding the script is that the user has fewer files to manage. The Object Rexx engine for WSH enables users to accomplish this in a seamless fashion.

There are two components to WSH. The first is the host - the product that can be scripted. The second is the engine - the product that interprets the script.

Object Rexx supplies the engine component of WSH.

9.1.1.1. The Gestation of WSH

As with many new technologies today, WSH introduces several new concepts and terms. The best way to describe these is to start with an overview of the problem that WSH addresses, and its history.

Until recently, Microsoft provided users simply with a COM (Common Object Model) interface to their office products. COM is a binary, as opposed to text, command input system. These commands drove the product - by, for example, telling Microsoft Word to print the current document - and did not contain any logic or decision-making capabilities. Users who wanted such capabilities developed them in programs external to the COM object. Accessing the interface required the user to develop the logic to drive the COM object at first in C++, then later in Visual Basic. The investment for the user, in development time, was quite significant.

In order to satisfy customer demand, a particular version of a scripting language (based on Visual Basic) was developed for each Microsoft product. In addition, the emergence of scripting languages such as

JavaScript™, with their ability to dynamically control Web browsers, led Microsoft to develop two more scripting languages, VBScript and JScript.

WSH is a consolidation of the scripting language proliferation. Borrowing heavily from the browser paradigm, the host interprets a language-independent XML file that contains one or more scripts where each script is encapsulated in a script (script tag) that denotes the language of the script, and any other necessary environmental parameters. The host extracts the script from the file, and passes it to the appropriate interpreter.

9.1.1.2. Hosts Provided by Microsoft

Microsoft provides three fully-implemented scripting hosts. They are Microsoft Internet Explorer, CScript, and WScript. As an expansion on the concept of using a scripting language to drive external products, CScript and WScript were developed to control the Windows operating system. The two modules are so similar that they are sometimes referred to as C/WScript. CScript is intended to be used from the command line, and WScript is best used in the Windows environment. Both provide their services to the script through the WScript object. Using the default method for output `WScript~Echo()`, CScript sends the output to a console screen in the same manner as the Object Rexx command `Say`, whereas `WScript~Echo()` in a script controlled by WScript will create a pop-up box in which the user must click the OK button to make it disappear.

9.2. Scripting in the Windows Style

Each flavor of WSH has an associated file type. This section gives a brief example of scripting for each file type, and suggestions that are appropriate in each case. If you need to, see the appropriate documentation for the exact syntax of WSH's XML format, and the syntax of an HTML file.

9.2.1. Invocation by the Browser

Invocation by the Web browser is probably the easiest scripting technique to illustrate, and the most familiar use of WSH. The following is a small HTML file that shows Object Rexx as the scripting language. There are three paragraphs that have the animating power of Object Rexx behind them. Each uses an Internet Explorer pop-up window to denote a particular mouse action. The appropriate activity takes place when the mouse is rolled over the first paragraph, when it leaves the second, and when it is used to click the third.

```
<HTML>
<!--
/*****************************************************************/
/* DISCLAIMER OF WARRANTIES. The following [enclosed]            */
/* code is sample code created by Rexx Language Association. This */
/* sample code is not part of any standard or RexxLA             */
/* product and is provided to you solely for the                 */
/* purpose of assisting you in the development of your           */
/* applications.  The code is provided "AS IS", without          */
```

```
/* warranty of any kind.  RexxLA shall not be liable for          */
/* any damages arising out of your use of the sample              */
/* code, even if they have been advised of the                    */
/* possibility of such damages.                                   */
/******************************************************************/
 !-->

<HEAD>
  <TITLE>A simple event</TITLE>
  <script language="Object Rexx" >
::Routine Display Public
   Window~Alert(Arg(1))
   Return   "something to keep the mouseover function call happy"
   </script>
</HEAD>

<BODY BGCOLOR="#ffffff">
  <FONT FACE="Arial, Helvetica" COLOR="#f00000">
  <H1>How to use events</H1>

  <FONT COLOR="#000000">
  <P>Moving the cursor over the following paragraphs will cause two
  events, respectively: one when you move onto the text, and one when
  you leave it. At both times a pop-up message will inform you about this.</P>

  <!-- in both cases the "alert" function of the object "window" is called !-->
  <FONT COLOR="#0000ff">
  <P onmouseout="alert("Cursor left paragraph")" LANGUAGE="Object Rexx">
  Event takes place when cursor leaves this paragraph.</P>

  <P onmouseover="a = Display('Cursor is over paragraph')"
LANGUAGE="Object Rexx">
Event takes place when cursor moves over this paragraph.</P>

  <FONT COLOR="#000000">
  <P>The following paragraph reacts when you click it:</P>

  <FONT COLOR="#0000ff">
  <P onclick="call Display "Thank you! The current time is" time()"," date()"
  LANGUAGE="Object Rexx">Click me!</P>
  </FONT>

</BODY>

</HTML>
```

The important things to note in this example are:

- The `LANGUAGE="Object Rexx"` attribute on each tag that contains code.
- The `<script>` tag in the `<HEAD>` section defines a function that can be called from any other code section in this HTML file.

- The Object Rexx keyword PUBLIC must be on the ::ROUTINE statement, or Object Rexx will not be able to make that name accessible outside of that script block.
- The Window object is accessible, even though it was not declared and the ::ROUTINE statements have the variable scope of an external routine.
- Some text was put on the RETURN statement simply as a precaution. Those familiar with Object Rexx know that routines called as functions demand a return value.

- All of the code for the mouseout= is completely contained within the and tags.
  - Also note the lack of the leading "Window~" on the Alert(). See Changes in Object Rexx due to WSH.
- The second event references the routine that was defined earlier as a function. The return value is assigned to the variable "a", and discarded as soon as the event finishes processing. Unlike the situation in JScript, function return values in WSH must be used in an expression, or assigned to a variable.
- The third event also references the routine that was defined earlier, but this time as a procedure and not as a function. The CALL statement forces this kind of access.
  - CALL statements do not produce an error if no value is returned. If a value is returned, and CALL was used to activate the routine, the value can be obtained from the special variable RESULT.

Additional examples can be found in the Samples\WSH subdirectory of your Object Rexx for Windows installation directory.

9.2.2. WSH File Types and Formats

Two main file types are used by WSH. Both follow an XML format that wraps the script code. The XML tags are interpreted by C/WScript, and direct it to the correct scripting engine to process the code inside. The file type .wsf is used to define scripts that are executed as commands. This is similar to the conventional way of invoking Object Rexx in the Windows environment. The file type .wsc is used to define scripts that are treated as COM objects. The XML tags here denote the properties, methods, and events of the COM object, as well as the correct engine to invoke for scripts.

Note that these XML files are well formed, but not valid. There is no associated Document Type Definition (DTD).

9.2.2.1. .wsf

The .wsf file type is as easy to invoke as HTML, and is very similar in appearance, with only minor differences. The .wsf file is used to drive the operating system in the same way that an HTML file is used to drive the browser. The file is an Object Rexx script file with an XML wrapper.

The following sample prints the version of the JScript engine and the version of the scripting host. If this file had the name "SimpleORexx.wsf", the command to invoke it would be "CScript //nologo SimpleORexx.wsf", or "WScript //nologo SimpleORexx.wsf".

```
<?xml version="1.0"?>
<?job error="true" debug="true" ?>
```

```
<package id="wstest">

<!--
/*****************************************************************/
/* DISCLAIMER OF WARRANTIES. The following [enclosed]            */
/* code is sample code created by Rexx Language Association. This */
/* sample code is not part of any standard or RexxLA             */
/* product and is provided to you solely for the                 */
/* purpose of assisting you in the development of your           */
/* applications.  The code is provided "AS IS", without          */
/* warranty of any kind.  RexxLA shall not be liable for         */
/* any damages arising out of your use of the sample             */
/* code, even if they have been advised of the                   */
/* possibility of such damages.                                  */
/*****************************************************************/
 !-->

<!--  Just a small file to demonstrate the *.wsf file format, and
    ---  what Windows provides by default.
    -->
<job idid="RunByDefault">

<!---
  ---   These functions are provided by WSH.
  -->
 <script language="JScript"><![CDATA[


function GetScriptEngineInfo(){
   var s;
   s = ""; // Build string with necessary info.
   s += ScriptEngine() + " Version ";  //  Except this function.  It can
                                       // only be accessed from JScript
                                       // or VBScript.
   s += ScriptEngineMajorVersion() + ".";
   s += ScriptEngineMinorVersion() + ".";
   s += ScriptEngineBuildVersion();
   return(s);
   }


]]></script>
<!---
  ---  Not all of the script needs to be within one tag, or use the
  ---  same language.
  -->
 <script language="Object Rexx"><![CDATA[

Say "This is "GetScriptEngineInfo()
Ver = "Accessing the version info from Object Rexx yields"
Ver = Ver ScriptEngineMajorVersion()"."
Ver = Ver||ScriptEngineMinorVersion()"."ScriptEngineBuildVersion()
Say Ver
```

```
WScript~Echo("Done!")

]]></script>
</job>

</package>
```

The important things to note in this example are:

- Accept the two XML tags (<? ... ?>) at the beginning as boilerplate, although the debug="true" can also be debug="false" without any detrimental effect.
- All XML tag names and attributes are in lower case.
- All XML tags have a beginning and an end tag. The beginning tag looks like <tag>, and the end tag </tag>. Where the tag contains only attributes, and there is no content between the beginning and the end tag, it is acceptable to abbreviate <tag attribute=""></tag> to <tag attribute=""/>.
- Comments are the same as in HTML.
- Following the <script> tag is the tag <![CDATA[, and preceding the <script/> tag is]]>. This tells the XML parser to ignore this text. If this is not done, many of the operators and special characters in the script will confuse the XML parser, and it will abort the script.
- There are several <script> tags; here Object Rexx is invoking a JScript function.
- The functions that begin with ScriptEngine... and the WScript object are not declared, yet Object Rexx finds them. They are implicit, and their scope is global.

Additional examples can be found in the Samples\WSH subdirectory of your Object Rexx for Windows installation directory.

9.2.2.2. .wsc

The .wsc file type is much more elaborate than the .wsf type. Since a .wsc file is used as a COM object, the XML must describe the object in a way that is independent of the script language. Consider the following example.

```
<?xml version="1.0"?>
<?component error="true" debug="true" ?>

<package id="SimpleObjectRexxCOMScriptTest">

<!--
/*****************************************************************/
/* DISCLAIMER OF WARRANTIES. The following [enclosed]            */
/* code is sample code created by Rexx Language Association. This */
/* sample code is not part of any standard or RexxLA             */
/* product and is provided to you solely for the                 */
/* purpose of assisting you in the development of your           */
/* applications.  The code is provided "AS IS", without          */
/* warranty of any kind.  RexxLA shall not be liable for         */
/* any damages arising out of your use of the sample             */
```

```
/* code, even if they have been advised of the                    */
/* possibility of such damages.                                   */
/******************************************************************/
!-->


<!---
  ---  An example script to demonstrate the features provided by the
  ---  COM structure.  Register our own typelib, create methods,
  ---  and create a property.
  !-->


<!---

  ---    This section registers the script as a COM
  ---  object when Register is chosen from the list of commands
  ---  that appear when this file is right-clicked.
  ---
  ---    The value of progid= is how the world will find us.
  ---  Two GUID's are needed, one for the COM object, and one
  ---  for the Typelib that will be generated. The routine's
  ---  Register and Unregister mimic those required in a COM
  ---  *.dll.  Even within these routines, there is full
  ---  Object Rexx capability.
  !-->
<component id="SimpleORexxCOM">
  <registration
     progid="SimpleObjectRexx.Com"
     description="Test of the COM scriptlet interface as seen by Object
     Rexx."
     version="1.0"
     clsid="{6550bac9-b31d-11d4-9306-b9d506515f14}">
   <script language="Object Rexx"><![CDATA[
::Routine Register Public
  Shell = .OLEObject~New("WScript.Shell")
  Typelib = .OLEObject~New("Scriptlet.TypeLib")
  Shell~Popup("We are registering, n o w . . . .")
     /*
      *  Please note that the name that follows must match
      * our file name exactly, or this fails when registering
      * with an "OLE exception", Code 800C0005 or Code 800C0009.
      */
     Typelib~AddURL("SimpleORexxCOM.wsc")
     Typelib~Path= "SimpleORexxCOM.tlb"
     Typelib~Doc = "Test component typelib for Simple Object Rexx.Com"
     Typelib~Name = "Test component typelib for Simple Object Rexx.Com"
     Typelib~MajorVersion = 1
     Typelib~MinorVersion = 0
     Typelib~GUID = "{6550bac5-b31d-11d4-9306-b9d506515f14}"
     Typelib~Write()
     Typelib~Reset()
     Shell~Popup("We've really done it n o w . . . .")
```

```
   ::Routine Unregister Public
     Shell = .OLEObject~New("WScript.Shell")
     Shell~Popup("We are outa here!")

     ]]></script>
    </registration>

<!---
  ---    This section is what describes this COM object to the outside
  ---  world. There is one property, and there are two methods named.
  ---  One of the methods is the default, since its dispid is 0.
  ---  Object Rexx does not support calling the default in a shorthand
  ---  manner. All calls are as follows:
  ---
  ---  Obj = .OLEObject~New("SimpleObjectRexx.Com")
  ---  Obj~DefaultMethod("Some Parm")
  ---
!-->
<public>
  <property name="ExternalPropertyName"
internalName="InternalPropertyName" dispid="3">
     </property>
  <method name="NamedRoutine">
     <parameter name="NamedParameter"/>
     </method>
  <method name="DefaultMethod" dispid="0">
     <parameter name="ReallyForTheOutsideWorld" />
     </method>
  </public>


<!---
  ---    This is the actual script code. Note that the property
  ---  is declared at the highest scope. If this is not done,
  ---  then the property will not be found, and the script
  ---  will not abend when the property is referenced.
!-->
  <script language="Object Rexx" ><![CDATA[
InternalPropertyName = "Sample Property"

::Routine NamedRoutine Public
say "There are "Arg()" args."
a = RxMessageBox("Is executing, now.","NamedRoutine","OK",)
  Return

::Routine DefaultMethod Public
say "There are "Arg()" args."
a = RxMessageBox("Is executing, now.","DefaultMethod","OK",)
WShell = .OLEObject~New("WScript.Shell")
a = WShell~Popup("A message via an implicit COM object.");
  Return  "a value"
```

```
  ]]></script>

</component>

</package>
```

The important things to note are:

- There are three distinct sections in this file, and two of them contain Object Rexx code.
  - The first section identifies this as a COM object. The `progid=`, `version=`, and `clsid=` attributes of the <`registration`> tag are given so that this file can be entered into the Windows Registry as a COM object. This is one of the sections that has code. The code here generates the Typelib when the script is registered as a COM object.
  - The second section lists all of the entry points to this object, their parameters, and any data that is being externalized. When the Typelib is generated, this information is used to create its contents. This is more of a designer's wish list than something that is enforced. The designer states what he or she believes to be the minimal number of parameters. The designer must then enforce this within the subroutine. However, be aware that other routines calling these listed here may pass more, or fewer, parameters than this section suggests. This is especially true for procedures named with <`method`> tags. WSH passes the named parameter `THIS`, which Object Rexx passes on to the routine.
  - The third section is the actual code.
- Read the comments before each section; they contain important information about that particular section.
- Any code that is put in the same scope as the property being assigned its value is called immediate code. Immediate code is executed when the COM object is loaded, before any of its pieces (methods, properties, or events) are accessed. It executes even if none of the external pieces are accessed.

Additional examples can be found in the Samples\WSH subdirectory of your Object Rexx for Windows installation directory.

9.2.3. Invocation from a Command Prompt

Invocation from a command prompt covers many possible means:

- Opening a DOS window to type the command into;
- Selecting Start->Run from the Windows taskbar;
- Starting from a file association made in Windows Explorer.

A conventional Object Rexx file is one in which every line is valid Object Rexx syntax, and makes no assumptions about global objects. It contains no XML wrapper as described in the section on `.wsf` files.

Consider what happens when a file named `WSH.rex` contains the single line: `'WScript~Echo("WSH is available.")'`; another file named `WSH.wsf` contains the same line of code in the .wsf wrapper described above; and another file, `Safe.rex`, contains the line `"Say 'Conventional Rexx file' Arg(1)"`.

9.2.3.1. As a Conventional Object Rexx File

From a command prompt, "Rexx WSH.rex", will stop with an error 97: Object "WScript" does not understand message "Echo".

From a command prompt, "Rexx WSH.wsf", will stop with an error 35: Invalid expression detected at "<".

From a command prompt, "Rexx Safe.rex GREAT!", produces one line of output, "Conventional Rexx File GREAT!".

9.2.3.2. As a Windows Scripting Host File

Both CScript and WScript will invoke a file from the command line. All of their parameters begin with a double slash. Two useful parameters are: //nologo and //e:. The //nologo parameter prevents the banner from being displayed, and //e: tells WSH not to interpret this file, and to pass the complete contents to the named engine. Enter CScript or WScript with no parameters or file names to see a complete list of parameters.

WScript converts all WScript~Echo() output into pop-up text boxes, whereas with CScript they are displayed as output lines in a DOS window. If CScript is executed from outside a DOS window (either from Start->Run, or from the use of Windows Explorer), a DOS window will be created for the output. Note, however, that it is removed when the script is complete. Usually, this means that the lifetime of the DOS window is long enough for a person to detect it, but not to actually read it.

From a command prompt, "cscript //e:"Object Rexx" WSH.rex" produces one line of output, "WSH is available." From a command prompt, "wscript //e:"Object Rexx" WSH.rex", produces a pop-up box that contains the title "Windows Script Host", an OK button, and the text "WSH is available."

From a command prompt, "cscript //e:"Object Rexx" WSH.wsf" will stop with an error 35: Invalid expression detected at "<". From a command prompt, "wscript //e:"Object Rexx" WSH.wsf", will seem as if it produced no output at all. Though Object Rexx is still generating the error message, WScript does not detect the output to STDOUT, and no DOS window is created.

From a command prompt, "cscript //e:"Object Rexx" Safe.rex GREAT!" produces one line of output, "Conventional Rexx File". Note the lack of the word GREAT!. WSH does not pass the command line args to Object Rexx. The WScript~Arguments method/object must be used, as in the following code:

```
/*    Note that the WScript object is not declared. It just appears
 * courtesy of CScript and WScript
 */
Say  "The arguments as WSH sees them."
If WScript~Arguments~length > 0 Then Do I = 0 To (WScript~Arguments~length - 1)
  Say i WScript~Arguments(i)
  End
Else Say "No arguments were sent."
```

From a command prompt, "wscript //e:"Object Rexx" Safe.rex GREAT!", will seem as if it produced no output at all. As when WSH.wsf is run by WScript with a known engine (see the relevant paragraph earlier), Object Rexx is still executing the SAY instruction, WScript does not detect the output to STDOUT, and no DOS window is created.

9.2.4. Invocation as a COM Object

This is the most intricate of the script files to execute. Multiple steps are involved, and there is no command that directly invokes the script. C/WScript cannot be used to directly invoke a `.wsc` file. It must be processed by other means first. Once created, the file must be registered.

Once registered, this can be invoked by any program that can call COM objects. It does not have to be another script; that program could be Visual Basic or C++. If the COM object is to be invoked by Visual Basic, it is a good idea to generate a Typelib. This helps Visual Basic to form its parameter list.

9.2.4.1. Registering the COM Object

Use either of two methods to register a `.wsc` file. The first is to right-click it in Windows Explorer, and choose `Register` from the list of commands that appears. The second is from the command line. For example, to register `WSH.wsc`, at a command prompt, enter the command, `"regsvr32 /c WSH.wsf"`.

The GUID in the `clsid=` attribute must be unique for the machine the COM object is being registered on. In other words, no other COM object may use the GUID. Once it is registered, the script cannot be moved. The path to a COM object is stored in the Registry as a complete path. If the script is moved, then Windows will not know how to find it.

9.2.4.2. Generating a Typelib

Use either of two methods to generate the Typelib. One is using code in the Register method of the `<registration>` section. See the sample `.wsc` code above for an example of this. The other is to choose `Generate Type Library` from the list of commands that appear when the file name is right-clicked in Windows Explorer.

9.2.4.3. Invoking

The easiest method of invoking the script, once it is a COM object, is to use an OLE-enabled application, such as Object Rexx. The following Object Rexx code shows how to define the object in Object Rexx, and invoke its methods.

```
<?xml version="1.0"?>
<?job error="true" debug="true" ?>

<package id="wstest">

<!--
/*****************************************************************/
/* DISCLAIMER OF WARRANTIES.  The following [enclosed]           */
/* code is sample code created by Rexx Language Association. This */
/* sample code is not part of any standard or RexxLA             */
/* product and is provided to you solely for the                 */
/* purpose of assisting you in the development of your           */
/* applications.  The code is provided "AS IS", without          */
/* warranty of any kind.  RexxLA shall not be liable for         */
/* any damages arising out of your use of the sample             */
/* code, even if they have been advised of the                   */
```

```
/* possibility of such damages.                                      */
/*********************************************************************/
 !-->

<!---
  ---     This example shows how easy it is to
  ---  invoke a COM object that is a script by means of
  ---  Object Rexx.
  -->
<job id="RunByDefault">
  <script language="Object Rexx"><![CDATA[

Say "Creating the ObjectRexx.Com object. "
Sample = .OLEObject~new("SimpleObjectRexx.Com")
Say "Just before the default method "
ReturnValue = Sample~DefaultMethod("A parm");
ReturnValue = Sample~NamedRoutine("A parm");
  ]]></script>
  </job>
</package>
```

Object Rexx is not the only way to invoke the script. Any application that can call COM objects can invoke it. For further information, see the relevant documentation.

9.2.4.4. Events

When scripts are turned into COM objects they can initiate events. Several types of events are supported: the default COM events, HTML or Behavior events, and ASP events. The type of event that the COM object supports is denoted by the `type=` attribute of the `<implements>` tag. An in-depth discussion of events and how to create, code, and handle them is beyond the scope of this documentation. However, there are a few concepts that should be mentioned.

9.2.4.4.1. COM Events

In the `<public>` section, where the external attributes of the COM object are disclosed, `<event>` tags can be added. They name the events that the script could possibly activate. When the script that calls the COM object instantiates it by using the method provided by WScript, rather than the Object Rexx method, it can inform the COM object that it will handle the events that the COM object fires. Note that when a script agrees to handle the events of an object, it must handle all of the events of that object.

For example, suppose the public section looked as follows:

```
<public>
  <event name="Event1" />
  <event name="Event2" />
</public>
```

and the script that instantiated the COM objects code looked as follows:

```
RexxObject = WScript~CreateObject("ObjectRexx.Com","Event_");
```

In that case, the instantiating script would be required to define the two routines below.

```
::Routine Event_Event1 Public
::Routine Event_Event2 Public
```

It is not acceptable if only one of the events is supported. Also, note the naming convention. The second parameter of CreateObject() names the prefix of the routine name that will support the event. The remainder of the routine name is composed of the event name from the <event> tag of the <public> section. Neither the prefix nor the empty string can be elided. In other words, neither `CreateObject("object",)` nor `CreateObject("object","")` is allowed. The script host will generate an error.

9.2.4.4.2. Internet Explorer Events

When coding Internet Explorer events, the user should be aware of the following. The section of code between the quotes on an HTML tag has to be complete, with correct syntax. The `THIS` object is implicity defined for the scope of the section. If the section calls a function, and the function needs access to `THIS`, then the section must pass `THIS` as a variable to the function. `THIS` is the browser's object that represents the tag that the event was fired from. For all of the exact properties and methods associated with `THIS`, see the documentation for the corresponding tag.

To illustrate, consider the following code extract:

```
<p onmouseover="Call RxMouseOver This" id="SomeTag">
"HOT" text, get your "HOT" text right here
</p>

<script language="Object Rexx">
::Routine RxMouseOver Public
Use Arg This
Text = "This is a <"This~tagName"> tag named '"THIS~id"'"
a = RxMessageBox(Text,"RxMouseOver","OK",)
Return "OK"
</script>
```

The code for the `onmouseover= "Call RxMouseOver This"` is complete and correct. If a function call had been used instead, the code would be something similar to `"a = RxMouseOver(This)"`. Do not forget to assign the results of a function call to something. If `THIS` is not passed as an argument to RxMouseOver, it will have the default value of a string whose content is `THIS`.

To cancel Internet Explorer events, the Object Rexx Boolean value `.false` must be returned. The integer values `0` and `1` are not appropriate alternatives. For example:

```
<a onmouseover="call SomeFunction; return .false" href="someURL">
```

9.2.5. WSH Samples

There are more features to WSH than are listed here. The Samples\WSH subdirectory of your Object Rexx for Windows installation directory contains some appropriate samples and an explanation of the

relevant features. Before running any samples, make sure that the latest version of Windows Scripting Host is installed on the machine.

Several sample files are stand-alone; these are all of the file types `.htm`, `.wsf` or `.rex`. However, all of the samples covering the aspects of using Object Rexx scripts as COM objects are in pairs or, in one case, a group of three. One file is the COM object, and the other is the script that instantiates it. All of the COM objects are of the file type `.wsc`. The files that instantiate them are either `.wsf` or `.rex`. The sample that uses three files illustrates the `include=` attribute of the <script> tag. All of the `.wsc` files must be registered before they can be used (see Registering the COM Object).

To view the `.htm` samples, use Windows Explorer to view the sample directory. Right-click the desired sample file, and choose Open With->Internet Explorer from the menu that appears.

To view the `.wsf` or `.rex` samples, use either a DOS window or Windows Explorer. From Windows Explorer, double-click the desired file. It will execute automatically. From the DOS window, make the sample directory the current directory, and use either CScript or WScript to execute the sample. The file `Print.rex` is an include file. It is not intended for direct execution.

Samples whose names begin with "w" use only Window pop-up boxes for output. Samples without the leading "w" are best viewed from the DOS window. They produce output that will not display in a Windows-only environment. Samples whose name begins with "call" are used to instantiate the COM objects once they are installed. If they are not installed, the error message `"Error 98.909: Class "....." not found"` will be issued.

9.3. Interpretation of and Deviation from the WSH Specification

This section deals with a number of issues to do with interpreting the WSH specification and with deviations from it.

9.3.1. Windows Scripting Host (WSH) Advanced Overview

Accommodating to WSH has necessitated some deviations from the Object Rexx standard. To best understand what these deviations are, you need to be aware of the components of WSH. In addition to the products that are hosts, there are special COM objects and different mechanisms for initiating the engine.

9.3.1.1. Hosts Provided by Microsoft

Microsoft provides three fully-implemented scripting hosts. They are Microsoft Internet Explorer, CScript, and WScript. As an expansion on the concept of using a scripting language to drive external products, CScript and WScript were developed to control the Windows operating system. The two modules are so similar that they are sometimes referred to as C/WScript. CScript is intended to be used from the command line, and WScript is best used in the Windows environment. Both provide their services to the script through the WScript object. Using the default method for output `WScript~Echo()`, CScript sends the output to a console screen in the same manner as the Object Rexx command `Say`,

whereas `WScript~Echo()` in a script controlled by WScript will create a pop-up box in which the user must click the OK button to make it disappear.

These are not the only Microsoft products that have WSH capabilities. The core of C/Wscript is `scrobj.dll`. Several Microsoft products implement various parts of the scripting host architecture by using `scrobj.dll`.

9.3.1.2. Additional COM Objects

Since JScript and VBScript were developed primarily to manipulate the Web browser DOM (Domain Object Model), they lack many of the features associated with a language that drives an operating system. They have no native facilities for I/O (Input and Output), or for controlling the file system. These powers are granted through several additional COM objects.

Most of the literature on WSH describes these objects. Most of the features in these additional COM objects are native to Object Rexx; for further information, see The OLEObject Class. Further documentation on the additional COM objects is readily available from other sources.

Object Rexx, since it is OLE-enabled, has access to all of these objects. OLE (Object Linking and Embedding) is an advanced protocol based on COM. Be aware that the automatic object WScript is only available when Object Rexx is activated by C/Wscript. Access cannot be obtained if Object Rexx is initiated by Internet Explorer, or when it is initiated in the classical method `"Rexx someFile.rex"`, either from the command line or from a command issued by the file explorer as an association with a file type. This is not a limitation of Object Rexx. It is a consequence of the manner in which this object is loaded.

The WScript object is not registered in the Windows Registry. It exists only when C/WScript dynamically creates it and then passes the pointer to Object Rexx. All scripting languages, including JScript and VBScript, have this limitation.

9.3.1.3. Where to Find Additional Documentation

The best source of up-to-date information on WSH is the World Wide Web. The keyword to use when searching the help facilities provided by Microsoft is "scripting". If you are using a search engine (available when you click "Search" on your browser's menu bar), insert "activescript" as the keyword.

In addition, there are several books on the subject. When browsing online bookstores, use the keyword, "activescript". The MSDN (Microsoft Developers Network) is a good reference source for the syntax of the XML used to define the WSH files.

Note that the correct file type to use for the XML file that C/WScript processes is `.wsf`. Existing documentation often states misleadingly that the file type to use is `.ws`. C/WScript requires the full file name, including file type, and it processes the file correctly only when the file type is `.wsf`. This seems to be hard coded into C/WScript, and no workaround is available.

9.3.2. Object Rexx in the WSH Environment

Object Rexx is fully compatible with the WSH environment. Interaction with JScript and VBScript is transparent. Legacy applications developed with these languages will not have to be discarded.

9.3.2.1. Object Rexx Features Available

All of the features normally associated with Object Rexx are available when Object Rexx is loaded by WSH. In addition, OLE support is loaded automatically. Scripts do not need to include `'::requires "ORexxOLE.CLS"'`. However, when Object Rexx is invoked by Internet Explorer, it honors the "sandbox" settings that the user has set in the browser's security panel. Access to I/O, the file system, external commands, and COM objects may not be granted.

9.3.2.2. Changes in Object Rexx due to WSH

To comply with the WSH definition, some of the scoping rules and default behavior of Object Rexx have been modified. The default behavior has been altered to allow some objects to be implicitly defined. The normal scoping rules now allow "global" objects to appear at any procedure depth, without requiring the use of EXPOSE, or the passing of the object as a parameter. Second-level objects can now be accessed without specifying the first level. These changes only apply to objects that WSH provides to Object Rexx. All other objects and variables behave in the standard ways.

Normally, access to objects requires explicit declaration through one of the OLE methods, as in:

```
"Window = .OLEObject~new("window")"
```

Some, like WScript, can only be passed in; others - `window`, for example - have a history of being implicitly available. Full documentation is not yet available as to what objects have these features, and therefore only a few will be mentioned.

As previously mentioned, the WScript object is implicitly available when Object Rexx is started by C/WScript. The "window" object is implicitly available when Object Rexx is initiated by Internet Explorer. For events associated with an HTML tag - `ONMOUSEOVER`, for example - the scriptlet in the HTML tag has `THIS` implicitly defined. Unlike `"WINDOW"`, `THIS` is not global. Typically, this scriptlet calls a procedure, and `THIS` must be passed to the procedure if the procedure needs to reference `THIS`.

Normally, you reference an object by naming the top level object, followed by the objects at second and subsequent levels, separated by the tilde symbol (~). However, in order to emulate the current behavior of Internet Explorer, the engine must resolve object names starting at the second level to the appropriate top level that owns them. The shorthand `"Document~WriteLn()"` or `"Alert()"` is just as acceptable as `"Window~Document~WriteLn()"` or `"Window~Alert()"`. It is preferable, as good coding practice, to explicitly state this relationship. Stating `"Doc = Window~Document"` removes all doubt as to which global object WriteLn() is associated with when the statement `"Doc~WriteLn()"` is encountered.

> **Note:** This applies only to global objects supplied by WSH. Objects created in or supplied by Object Rexx must be named in the normal fashion.

9.3.2.3. Parameters

A called routine may receive more parameters than expected. This is not necessarily an error on the caller's part; WSH adds extra parameters on occasion. When WSH does this, Object Rexx adds the parameters at the end. There is an exception to this. The documentation is ambiguous in certain sections about defining properties for scripts that are used as COM objects. If the XML that defines the script

states that a name should be a property, but Object Rexx finds it defined as a function, then Object Rexx will prepend the parameter list with `GET` or `PUT`, depending on the direction of the property access. For more information, see the sample file `Call_ExtraParms.wsf` in the Samples\WSH subdirectory of your Object Rexx for Windows installation directory.

9.3.3. Properties

WSH defines properties as variable values that a COM script exposes to outside routines, or strings and numbers extracted from a Typelib. Properties are to be treated as global variables within the accessing script. Properties can be implemented as variables or as functions.

Object Rexx supports declaring and defining properties in the intent of the specification (see the section on `.wsc` files). That means that the variables at the highest scope, the closest to what could be considered as global, may have their values exposed as properties for other programs to use.

For another program to reference these properties, it must instantiate the COM object, and the object name must precede the property name. For example:

```
Object = .OLEObject~New("SimpleObjectRexx.Com")
/*  The next line is a property GET */
Say "The value for ExternalPropertyName is:" Object~ExternalPropertyName
Object~ExternalPropertyName = "New Value"  -- This is a PUT
```

If you experiment, you will find that there is also a shorthand method, as follows:

```
Object = .OLEObject~New("SimpleObjectRexx.Com")
/*  The next line is a property GET */
Say "The ExternalPropertyName value is:" Object~ExternalPropertyName()
Say Object~ExternalPropertyName("New Value")
```

In the case of the second reference, the method is both a PROPERTYGET and a PROPERTYPUT. It gets the old value, replacing the current one with the parameter inside the parenthesis. If more than one parameter is passed, the additional parameters are ignored.

> **Note:** This does not always work, and is supported only by Object Rexx. The cases in which it does not work are where the properties are defined as functions and not as simple variables. These calls are, in fact, methods and not property references. When Object Rexx receives method calls for properties, it converts them to the appropriate action. In the case of properties defined as functions, WSH translates the property action into a function action. However, when the action is initiated as a function and not as a property, WSH does not always make the appropriate or correct translation.

Object Rexx does not support the concept of global variables. For a COM script to reference its own properties, and to react to outside scripts changing them, then the properties have to be global. To meet the requirement that properties are global in scope within the defining script, the Built-In Function (BIF) Value() has been expanded to accept "WSHPROPERTY" as a selector when referencing properties. As with variables accessed with the "ENVIRONMENT" selector, these variables persist only during the life of the COM object that supplies the properties. The next time that the COM is run, the values will be at initial coded state.

The WSH supports various syntax combinations in the case of implementing a property as a function. In all combinations, the function is named in the <property> section or tag. It assumes that, when no function is named, the property is a variable; however, it does not enforce this assumption. It is possible to name a property and define it as a function. Object Rexx defines this to mean that the function must be invoked whenever a property access is attempted. Object Rexx notifies the function of the intended access direction by inserting GET or PUT as the first argument, and shifting all original arguments accordingly; that is, the original first argument is the second, the second is the third, and so on. For a demonstration of this behavior, see the Call_PropertyORexx.wsf sample in the Samples\WSH subdirectory of your Object Rexx for Windows installation directory.

The WSH also establishes that Type Library variables may be made accessible to the script. This violates the default value and scope mechanisms of Object Rexx. To meet the requirement that properties are global in scope within the defining script, the Built-In Function (BIF) Value() has been expanded to accept "WSHTYPELIB" as a selector when referencing elements in a Type Library. As with variables accessed with the "ENVIRONMENT" selector, these variables (because they are external to Object Rexx) are global and persist only during the life of the COM object that supplies the properties. In addition, they are read only. They are immutable; they cannot be changed.

9.3.4. The Object Rexx "Sandbox"

Object Rexx contains a feature known as the Security Manager. When this is enabled it can restrict and audit the other native abilities of Object Rexx. When used with WSH, Object Rexx honors the IObjectSafety interface and its methods GetInterfaceSafetyOptions() and SetInterfaceSafetyOptions() by translating their calls into Security Manager settings. This means that when Object Rexx is in the Internet Explorer's sandbox, it will restrict itself to the user's settings. The most secure situation is one where Object Rexx does not interact with the user's desktop (no reads or writes to the hard disk, no external commands, and so on).

9.3.4.1. Implications of Browser Applications That Run Outside the "Sandbox"

The most useful aspect of this feature is that the user may select the most secure settings for the Internet, but allow desktop interaction for pages delivered by the local intranet server. In keeping with the current trend in IT, Object Rexx allows users to leverage their investment in desktop software. This facility is intended for clients who use the intranet to lighten the client, or put a Web interface on legacy applications. A lighter client desktop means less software on the user desktop to maintain.

9.3.5. Features Duplicated in Object Rexx and WSH

Several features are available from both WSH and Object Rexx. However, the overlap is not exact, and knowing the differences can aid the user in deciding which is more appropriate to use.

9.3.5.1. Declaring Objects with Object Rexx or WScript

When instantiating COM or OLE objects as Rexx objects, either the native Rexx .OLEObject~new() method, or the WScript~CreateObject() method can be used. The WSH method has the advantage of allowing the script to support the events that the object might fire. This is part of its definition, and no scripting language will have access to this ability in its native object enabler. The disadvantage is that it is a COM object performing a function that can be done internally.

Another disadvantage of using the WSH method becomes evident if the script is executed outside of the context of WSH. The WScript object will not exist. Therefore, unless the ability to sink events is necessary, it is suggested that the native Object Rexx method be used.

9.3.5.2. Subcom versus the Host Interface

With the advent of WSH, there are two ways to use Object Rexx to drive a product. The first is through the Object Rexx Subcom interface. The second is for the product to become a Windows Scripting Host. The advantage of the WSH interface to the product is that it is a COM interface. This positions the product to take advantage of DCOM. This interface also allows the package developed by the user to pass objects to Object Rexx.

The disadvantage is the loss of richness contained in the Subcom interface, and the loss of the close integration that a `.dll` connection has over a COM connection. The Subcom interface allows the package to tailor Object Rexx in ways that are not possible through the COM interface, especially when the Object Rexx Exit Handlers are implemented.

When writing a product that will be a WSH to Object Rexx, refer to the sections "Concurrency" and "COM Interfaces" in "Windows Scripting Host Interface", in the Object Rexx for Windows: Programming Guide.

9.3.5.3. .dll vs COM

There are several issues that should be considered when a choice needs to be made between a COM or a `.dll` interface. These issues stem from the intended purposes of each interface.

The `.dll` interface was developed to extend code reuse by allowing global scope subroutines and functions to be externalized into a module that is separate from the executable. When more than one executable wanted these functions, they all shared the same code that was loaded into memory. The code that was in the `.dll` executed in the frame of the `.exe` module. It had the same address space and other environmental parameters. Multiple copies of a `*.dll` code exist on a machine at one time. The first one that was found in the search path was loaded.

COM was developed to embody a flat model world; only one copy per machine. It was developed to solve two problems with the `*.dll` interface. The first was entry point resolution, and the other was using the wrong `*.dll` because the search path was not correct. COM does this by using RPC, a mechanism that was designed to communicate between different machines. For conceptual purposes, COM modules then function in a different address space from that of the invoking `*.exe`. Therefore, there is overhead in making any data that is to be passed back and forth opaque on the sender’s side, and converting it into usable data on the receiver’s side.

Appendix A. Notices

Any reference to a non-open source product, program, or service is not intended to state or imply that only non-open source product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Rexx Language Association (RexxLA) intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-open source product, program, or service.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-open source products was obtained from the suppliers of those products, their published announcements or other publicly available sources. RexxLA has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-RexxLA packages. Questions on the capabilities of non-RexxLA packages should be addressed to the suppliers of those products.

All statements regarding RexxLA's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

A.1. Trademarks

Open Object Rexx™ and ooRexx™ are trademarks of the Rexx Language Association.

The following terms are trademarks of the IBM Corporation in the United States, other countries, or both:

1-2-3
AIX
IBM
Lotus
OS/2
S/390
VisualAge

AMD is a trademark of Advance Micro Devices, Inc.

Intel, Intel Inside (logos), MMX and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the Unites States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in

the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

A.2. Source Code For This Document

The source code for this document is available under the terms of the Common Public License v1.0 which accompanies this distribution and is available in the appendix *Common Public License Version 1.0*. The source code itself is available at `http://sourceforge.net/project/showfiles.php?group_id=119701`.

The source code for this document is maintained in DocBook SGML/XML format.

Appendix B. Common Public License Version 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS COMMON PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

B.1. Definitions

"Contribution" means:

1. in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
2. in the case of each subsequent Contributor:

    a. changes to the Program, and

    b. additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents " mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

B.2. Grant of Rights

1. Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.
2. Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such

combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

3. Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.
4. Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

B.3. Requirements

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

1. it complies with the terms and conditions of this Agreement; and
2. its license agreement:
    a. effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
    b. effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
    c. states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
    d. states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

1. it must be made available under this Agreement; and
2. a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

B.4. Commercial Distribution

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor’s responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

B.5. No Warranty

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

B.6. Disclaimer of Liability

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE

PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

B.7. General

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. In addition, if Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. IBM is the initial Agreement Steward. IBM may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W