PFXplus COM Documentation

Describes the implementation of COM in PFXplus.

Documentation

This is preliminary documentation and is subject to change in future releases.

The features described here are included in the PFXplus Compiler and PFXplus Library runtime. Some features require the include file `COMOBJ.PFI`.

Overview

PFXplus now supports the Microsoft Component Object Model (COM) as an Automation Client. This means that you can conveniently write and run PFXplus programs which communicate with and control other programs which support Automation. This includes all of Microsoft Office (Word, Excel, Access, PowerPoint, etc), Mail and Exchange, Internet Explorer and countless other programs.

The four steps in COM Automation programming are as follows.

1. Load a type library (compile time)
2. Declare an object using that type library (compile time)
3. Create a new object, or get a reference to an existing one (run time)
4. Manipulate the properties and methods of that object (run time)

Here is a sample program which does just that. The program loads a type library using entries in the registry. Then it declares an application object.

Next it attempts to find an existing running copy of the Word application, but if that fails it starts a new one. It makes sure the application is visible, creates a new document and inserts some text into the document.

```
#use macro
LOAD_TYPELIB Word from "{00020905-0000-0000-c000-000000000046}" 8.0 $0409
COM_OBJECT Wordapp ISA Word\Application
trap COM_GETACTIVE Wordapp
[err] COM_CREATE Wordapp
Wordapp\Visible=true
Wordapp\Documents\Add
Wordapp\ActiveDocument\Paragraphs(1)\Range\InsertBefore ;
   "This is an Automation test."
```

Here is a different approach. This one opens a type library directly and declares a document object. Then it uses GETOBJECT to create the document object directly, and start Word by its association with the DOC extension. Finally it extracts and prints out the name of the document.

```
LOAD_TYPELIB Word from "F:\Program Files\Microsoft
Office\Office\MSWord8.OLB"
COM_OBJECT Worddoc ISA Word\DOCUMENT
SET Worddoc=COM_GETOBJECT("c:\pfx\testing.doc")
showln Worddoc\Name
```

Reading IDL

The OLE/COM object viewer provides disassembled IDL for any given type library. This is the major source of information about how to call these objects in PFXplus.

Often the simplest way is to start with Visual Basic code that does the desired job, and translate it line by line into PFXplus by using the PFXplus commands listed below and converting periods (".") into backslashes ("\").

If that gives compile errors, consult the generated IDL and the notes following for how to write in PFXplus.

Default Property

Default properties are not supported. Access all properties by name.

An object that has a default property will identify this in the IDL by giving it a `DISPID` of zero. VB allows the object to be used without qualification in contexts where a value is expected.

PFXplus requires that each property be referred to by name. Either use the name of the property that has a `DISPID` of zero, or look for a property called `Value`.

```
// this will not compile, because EXL is an object, not a value
Showln EXL
// both of these work fine, because they are in the IDL.
Showln EXL\Value
Showln EXL\_Default
```

Default Indexers

Default indexers are not supported. Access all indexed properties by name.

An object that has a default indexer will identify this in the IDL by having a property with a `DISPID` of `0xfffffffc`. Often the property name will be `_NewEnum`. VB allows the object to be used with subscripts in contexts where an array is expected.

PFXplus requires that indexed properties be referred to by name. Using the name of the property that has a `DISPID` of `0xfffffffc` will not work; instead look for a property called `Item`. For example:

```
COM_OBJECT Wsheet isa Excel\Worksheet
WSheet\Cells\Item(2, 1) = "1st Quarter"
```

Optional Parameters

Optional parameters are supported. Parameters of type `VARIANT` marked as `[optional]` may be omitted, with two restrictions.

1. Optional parameters of types other than `VARIANT` are not compatible with Automation and are treated as required.
2. If an optional parameter is omitted, all subsequent parameters must be optional and must also be omitted. Currently this prevents indexed put properties from having optional parameters, since the value being set is the last parameter.

For example:

```
EXL\SendKeys "xxx"
EXL\SendKeys "xxx" TRUE
```

Named Parameters

Named parameters are not supported. All parameters must be specified positionally.

Typeless Objects

Late binding on typeless objects is not supported. The PFXplus compiler needs to know the specific COM type of the object in order to generate correct code.

IDispatch* represents a new object, with its own type description. Unfortunately, the type library gives no hint as to what that might be, so it is left up to the programmer to make the correct decision. It is necessary to store the value into a suitable type of COM_OBJECT in order to access the methods and properties of the object. For example:

```
// this will not compile, because Selection returns an IDispatch*
EXL\Selection\Columns\AutoFit
// this works fine, because compiler treats it as a Range
COM_OBJECT sel isa Excel\Range
set sel = EXL\Selection
sel\Columns\AutoFit
```

User-defined Types

User-defined types (UDT) are supported as opaque data objects. If the type fits into 32 bits (4 bytes), it can be passed by value as an INTEGER. A UDT of any size can be passed by reference, as an INTEGER containing a pointer.

Access to fields inside UDTs is not supported.

Events

Handling events requires doing two things: install an event sink and specify what to do with individual events.

Install an Event Sink

These are the steps to follow.

1. Load a type library.
2. Declare a COM_OBJECT for a coclass that has a source (outgoing) interface. This is a connectable object: COBJ.
3. Set COBJ to an object reference of that type (eg COM_CREATE or COM_GETOBJECT).
4. Declare a COM_OBJECT for the source (outgoing) interface. This is EVIF.
5. Connect to COBJ using EVIF and receive events.

For example

```
COM_CONNECT EVIF TO COBJ                                    Connect
SET EVIF TO 0                                               Disconnect
```

Designate an Event Handler

These are the steps to follow.

1. Write a procedure (or method) with a signature that matches an event.
2. Specify the procedure as the handler for an event. The event name can be derived from the procedure name, or it can be supplied as a string.

Note that events typically do not have a return value, but they often do have reference parameters, to return values back to the connectable object.

For example:

```
PROCEDURE PROC1 INDICATOR DONE BYREF
END_PROCEDURE

COM_HANDLER PROC1 FOR EVIF
COM_HANDLER PROC2 FOR EVIF "Alias"
```

Command Reference

The supported commands are as follows.

```
LOAD_TYPELIB comlib FROM [guid version locale|path]
```

```
COM_OBJECT comobj ISA comlib
COM_CREATE comobj [flags]
COM_GETACTIVE comobj [flags]
SET comobj=comexpr
...COM_GETOBJECT(arg1, arg2)...
...COM_GETUNKNOWN(integer)...
Assignment and Expressions
COM_CONNECT eventobj TO comobj
COM_HANDLER proc FOR eventobj
```

LOAD_TYPELIB

Load a type library to gain access to the definitions in it.

```
LOAD_TYPELIB comlib FROM progid [ version locale ]     (1)
LOAD_TYPELIB comlib FROM guid [ version locale ]       (2)
LOAD_TYPELIB comlib FROM path                          (3)
```

<comlib> is an identifier.

<progid> is a ProgID string which identifies an object defined in the type library, in the form `application.object` or `application.object.version`.

<guid> is the globally unique identifier for the type library, as a 38 character string (including braces and hyphens) in the format shown.

```
{99999999-9999-9999-9999-999999999999}                 Example
{00020905-0000-0000-c000-000000000046}                 Word
```

<version> (optional) is the major and minor version number required. For example, MS Word in Office 97 is version 8.0. The default is 1.0 and usually works.

<locale> (optional) is the identifier for the language required, usually expressed in hexadecimal. For example, $0c09 is Australian English, $409 is US English, $9 is English, $0 is neutral.

<path> is a pathname.

In format 1 (recommended) the type library is opened using the registry. Key values can be found under `HKEY_CLASSES_ROOT` (`HKCR`). For example, the MS Word application object can be found in the registry using `RegEdit` at either of these keys, which then contain a reference to the type library.

```
HKCR\Word.Application
HKCR\Word.Application.8
```

In format 2 the type library is opened using the registry. Key values can be found under `HKEY_CLASSES_ROOT\TypeLib`. For example, MS Word for English can be found in the registry using `RegEdit` at this key.

```
HKCR\TypeLib\{00020905-0000-0000-c000-000000000046}\8.0\9\Win32
```

In format 3, the type library is opened directly using the path provided. The path may be a stand-alone type library, an executable file containing a resource of type `ITypeLib` or a suitable moniker.

In any case the identifier <comlib> is defined to represent that type library in subsequent declarations.

Implementation

Calls `LoadRegTypeLib()` or `LoadTypeLibEx()`.

Examples

```
LOAD_TYPELIB Word from "Word.Application"
LOAD_TYPELIB Word from "{00020905-0000-0000-c000-000000000046}" 8.0 $0409
LOAD_TYPELIB Word from ;
                   "C:\Program Files\Microsoft Office\Office\MSWord8.OLB"
```

COM_OBJECT

Define a COM object for use by other statements and functions.

```
COM_OBJECT comobj ISA comlib\typename
```

<comobj> is an identifier.

<comlib> is a type library previously defined using LOAD_TYPELIB.

<typename> is the name of a type of object declared in the type library.

<comobj> is defined to be an object of type <typename>.

Implementation

Calls multiple functions associated with ITypelib, ITypeInfo and ITypeComp.

Examples

```
COM_OBJECT Wordapp  ISA Word\Application
COM_OBJECT Wordapp2 ISA Word\Application
COM_OBJECT Worddocs ISA Word\Documents
```

COM_CREATE

Create a single uninitialised object of the given class.

```
COM_CREATE comobj [flags]
```

<comobj> is a defined COM object.

<flags> is a value which specifies the context in which the object will run. Allowed values are defined as CLSCTX_xxx in COMOBJ.PFI. The default is CLSCTX_SERVER.

When used with an Application object, a new copy of the application is started. Further calls would be used to create or load a document and manipulate it.

Implementation

Calls CoCreateInstanceEx().

Example

```
#use COMOBJ
COM_CREATE Wordapp
COM_CREATE Wordapp CLSCTX_INPROC_SERVER
```

COM_GETACTIVE

Retrieve a reference to a COM object from the Running Object Table.

```
COM_GETACTIVE comobj [flags]
```

<comobj> is a defined COM object.

<flags> is defined for future use, and must be 0.

<comobj> is set as a reference to an object of its defined type, obtained from the Running Object Table. If there is no such object, an error is triggered. It is often useful to trap this error and then call COM_CREATE to create a new object, as in the example.

Implementation

Calls GetActiveObject().

Example

```
#use macro
trap COM_GETACTIVE Wordapp
[err] COM_CREATE Wordapp
```

SET

Create or destroy an additional reference to a COM object.

```
SET comobj=0                                          (1)
```

```
SET comobj=comexpr                                    (2)
SET property=comexpr                                  (3)
```

<comobj> is a defined COM object.

<comexpr> is an expression which returns a reference to a COM object.

<property> is an expression which returns a reference to a property of a COM object, with the `[propputref]` attribute.

In format (1), any previous reference by <comobj> to an object is released. If there are no other references to the object, the object will be destroyed. <comobj> then does not refer to any object.

In format (2), any existing reference is released as for format (1).

If <comexpr> is a reference to an object, <comobj> becomes a second reference to the same object.

If <comexpr> is a function, and the function can be successfully evaluated and returns an object, <comobj> becomes a reference to that object. Otherwise <comobj> does not refer to any object.

No error is triggered if the type of COM object returned by <comexpr> is not the same as expected for <comobj>. However, any subsequent attempt to use <comobj> will almost certainly fail with an error such as a type mismatch.

Format (3) functions exactly as an assignment described below, but should be used for cases where the property to be set is defined with the `[propputref]` attribute.

Implementation

Calls `AddRef()` and `Release()`.

Example

```
SET Worddoc=COM_GETOBJECT("c:\pfxplus\testing.doc")
SET Worddoc=0
SET MyObj\DocRef=Worddoc
```

COM_GETOBJECT

Obtain a reference to an object, creating it if necessary.

```
...COM_GETOBJECT(filename)...                         (1)
...COM_GETOBJECT(filename, progid)...                 (2)
...COM_GETOBJECT("",progid)...                        (3)
...COM_GETOBJECT("?",progid)...                       (4)
```

<filename> is a pathname.

<progid> is a `ProgID` string defined in the registry, in the form `application.object` or `application.object.version`.

In format (1) the application associated with <filename> is started (if necessary) and the object defined in <filename> is activated and returned. <filename> can be a simple disk file or a moniker.

In format (2) the application associated with <progid> is started (if necessary) and the file specified by <filename> is loaded. <filename> may contain multiple types of object, but the object of type specified by <progid> is the one activated and returned.

In format (3) a new object instance of the specified type is created and returned.

In format (4) a reference to an existing object of the specified type is obtained and returned. If no such object exists, an error is triggered.

Implementation

Format (1) uses `CreateBindCtx()`, `MkParseDisplayName()` and `BindMoniker()`.

Format (2) uses `CLSIDFromProgID()`, `CoCreateInstance()`, and `IPersistFile Load()`. Format (3) and (4) use `CLSIDFromProgID()` and `CoCreateInstance()` or `GetActiveObject()`.

Example

```
SET Worddoc=COM_GETOBJECT("c:\pfxplus\testing.doc")
SET Worddoc=COM_GETOBJECT("c:\pfxplus\testing.doc","Word.Document")
SET Wordapp=COM_GETOBJECT("","Word.Application")
SET Wordapp=COM_GETOBJECT("?","Word.Application")
```

COM_GETUNKNOWN

Obtain a reference to an object previously created.

```
...COM_GETUNKNOWN(value)...
```

<value> is an integer value representing a COM object, typically obtained by a Windows API call.

The return value is a reference to the object denoted by <value> as a COM expression, suitable for SET or the like.

Implementation

Calls `QueryInterface()`.

Example

```
SET Worddoc=COM_GETUNKNOWN(somevalue)
```

Assignment and Expressions

Manipulating COM objects.

```
...comattr...                                   (1)
comattr=value                                   (2)
...comattr(value, value...)...                  (3)
comattr [value]...                              (4)

comattr is comobj[\attrib]...
```

<comattr> is a reference to a specific attribute of a <comobj> formed as a path with each element separated by backslash characters.

<comobj> is a COM object of a type defined in a type library. Consult the documentation supplied with the type library for a list of available object types.

<attrib> is an attribute of <comobj> as defined in the type library. Consult the documentation for the attributes (properties and methods) of each object type.

<value> is a value of any basic type (logical, integer, date, number, real, string).

In format (1) the value of a property <comattr> is retrieved and used as the value of the expression.

In format (2) the value of <value> is assigned to the property <comattr>.

In format (3) <comattr> is called as a function, passing each <value> as its arguments. The number of arguments can vary. It must include all the required arguments, and may include the optional arguments. The returned value is used as the value of the expression.

In the special case where <comattr> refers to a collection, there should be exactly one argument of a numeric type. The expression evaluated is equivalent to <comattr>`\ITEM(`<value>`)`, and the returned value is a member of the collection.

In format (4) <comattr> is called as a procedure, passing each <value> as its arguments, using the same rules as for (3). The returned value (if any) is not used.

In cases (1) and (3) the return value may be an object. When used in a SET statement, the assignment is of a reference to the object. Otherwise, the assignment is of the default property of the object.

Examples

```
LOAD_TYPELIB Word from "{00020905-0000-0000-c000-000000000046}" 8.0 $0409
COM_OBJECT Wordapp ISA Word\Application
COM_CREATE Wordapp
showln "The name of the Word is " Wordapp\Name
showln "ActivePrinter=" Wordapp\ActivePrinter
Wordapp\Visible=true
Wordapp\Documents\Add
Wordapp\Documents\Add "FaxTemplate"
Wordapp\Documents\Add "LetterTemplate" true
showln "The name of the document is " Wordapp\Documents\Item(1)\Name
showln "The name of the document is " Wordapp\Documents(1)\Name
Wordapp\ActiveDocument\Paragraphs(1)\Range\InsertBefore ;
"This is an Automation test."
Wordapp\Documents\Open("c:\MyDocuments\Resume")
COM_OBJECT Mydoc ISA Word\DOCUMENT
SET Mydoc=Wordapp\ActiveDocument
Mydoc\Bookmarks(1)\Delete
Wordapp\Move 0 0
Wordapp\Quit true true false
```

COM_CONNECT

Connect up an event interface.

```
COM_CONNECT eventobj TO comobj                    (1)
SET eventobj TO 0                                 (2)
```

<eventobj> is a defined COM object referring to an event interface for <comobj>. It will be defined with the `[source]` attribute in the type library.

<comobj> is a defined COM object.

In format (1) <eventobj> is connected up as an event handler, and enables events to be dispatched to procedures defined using `COM_HANDLER`.

In format (2) <eventobj> is disconnected and event dispatching ceases.

Implementation

Uses `IConnectionPointContainer->FindConnectionPoint()`, `EventSink::Factory()` and `IConnectionPoint->Advise()`. The runtime provides a customised implementation of `IDispatch()`, which receives all events.

Example

```
COM_OBJECT AxFlexGrid$Object ISA AxFlexGrid$TypeLib\MSFlexGrid
COM_OBJECT AxFlexGrid$Event ISA AxFlexGrid$TypeLib\DMSFlexGridEvents
SET AxFlexGrid$Object = COM_GETUNKNOWN(CtrlIUnknown)
COM_CONNECT AxFlexGrid$Event TO AxFlexGrid$Object
COM_HANDLER DblClick for AxFlexGrid$Event
```

COM_HANDLER

Specify the procedure to handle a specific COM event.

```
COM_HANDLER proc FOR eventobj [ "alias" ]
```

<proc> is the name of a procedure to handle the specified event. It must already be defined but may be a forward reference, and the signature must match that defined for the event.

<eventobj> is an event handler object defined by `COM_HANDLER`.

< alias> is an optional constant string argument naming the event handler.

By default, <proc> becomes the handler for the event named "proc" defined on the event interface referred to by <eventobj>.

If <alias> is defined, <proc> becomes the handler for the event named "alias".

In either case procedure <proc> is called whenever the event occurs.

Implementation

Internal.

Example

```
PROCEDURE FlexGrid_OnDblClick FORWARD

COM_HANDLER FlexGrid_OnDblClick for AxFlexGrid$Event "DblClick"

PROCEDURE FlexGrid_OnDblClick
   MSGBOX "FlexGrid" "DblClicked"
END_PROCEDURE
```