THINKDATA INC.

Clarion Third Party Add-Ons

xmlFUSE User Guide

CLARION THIRD PARTY ADD-ONS

xmlFUSE 2.1 User Guide

2508 Pacific Avenue • Suite 1
Venice Beach, California 90291
Phone 310.823.2571
Fax 310.943.1858
info@thinkdata.com

Table of Contents

Introduction

Overview

The xmlFUSE product provides a native COM (Component Object Model) automation interface to the Microsoft XML (Extensible Markup Language) 4.0 SDK and the Microsoft SOAP (Simple Object Access Protocol) 3.0 SDK using Clarion 5.5 and Clarion 6. It relies on the Plugware COM classes which encapsulate the major portions of the COM Application Programming Interface (API) specification as designed by Microsoft. The Plugware COM classes usher in a new era of stability and performance when writing COM automation interfaces in Clarion. Full source code to these classes has been provided along with detailed examples demonstrating how to use the xmlFUSE product. This gives the developer/end-user an opportunity to understand the underlying principles of COM while providing an open and stable platform from which to develop additional automation interfaces.

What is XML?

XML is shorthand for Extensible Markup Language, a specification developed by the World Wide Web Consortium (W3C). It allows designers to create their own customized tags which enable the definition, transmission, validation, and interpretation of data between applications and between organizations. If you are new to XML and would like to view a short tutorial and sample documents please visit http://www.w3schools.com/xml/.

What is SOAP?

SOAP, or Simple Object Access Protocol, provides a way for applications to communicate with each other over the Internet independent of platform. SOAP is based on XML and relies on HTTP (HyperText Transfer Protocol) to communicate over the web. HTTP is supported by all Internet browsers and servers; therefore the SOAP communication is safe from firewall interruption. In short, SOAP provides a way to communicate between applications running on different operating systems, using different technologies and programming languages. If you are new to SOAP and would like to view a short tutorial, please visit http://www.w3schools.com/soap/.

What are Web Services?

Web services are applications whose logic and functions are accessible using standard Internet protocols and data formats such as Extensible Markup Language (XML) over Hypertext Transfer Protocol (HTTP), and SOAP (Simple Object Access Protocol). Like component-based development, Web services represent black-box functionality that can be reused without worrying about how the service is implemented.
A Web service interface is defined strictly in terms of the messages that the service accepts and generates. Applications using a Web Service can be implemented on any platform in any programming language, as long as they can create and consume messages defined for the service interface. A Web service can also aggregate other services to provide a higher-level set of features.

Why developers should be interested in Web services?

- Interoperability - Any Web service can interact with any other Web service and can be written in any language.
- Ubiquity - Web services communicate using HTTP and XML. Any connected device that supports these technologies can both host and access Web services.
- Low Barrier to Entry - The concepts behind Web services are easy to understand, and developers can quickly create and deploy them using many tool-kits available on the web.
- Industry Support - Major content providers and vendors are supporting the Web services movement.

(Source: Amazon.com Web Services API and Integration Guide)

Why Does a Developer Need xmlFUSE?

xmlFUSE provides a complete native Clarion wrapper around the MS XML 4.0 SDK. All developers dealing with XML in Clarion can benefit from using xmlFUSE. This one product provides the facility for creating and parsing XML documents using DOM (Document Object Model), SAX (Simple API for XML), XSD (XML Schema Definition Language), and SOM (Schema Object Model). You can also use XSL (Extensible Stylesheet Language) to format and transform your XML documents and the MS XML HTTP library included in the MS XML 4.0 SDK will allow you to send and retrieve documents over the Internet. xmlFUSE contains numerous examples for using the XML and XML HTTP portions of the product.

In addition to all of the XML functionality, xmlFUSE provides a complete native Clarion wrapper around the MS SOAP 3.0 SDK. This synergy between XML and SOAP in the same product allows a developer to interface with XML Web Services and process their requests and responses without purchasing additional tools. XML Web Services are growing in popularity and sites such as XMethods

(http://www.xmethods.com) provide listings of dozens of services you can integrate into your Clarion application. This cutting edge technology will deliver increased value to your customers by integrating many sources of Internet application content into your Clarion application.

Finally, the xmlFUSE product ships with complete source code to all of the class wrappers, Plugware COM layer, and example applications. This makes xmlFUSE an excellent learning tool for getting up to speed with XML and SOAP. XML is the standard technology used for describing data between applications on any operating system, using any type of database, regardless of platform or programming language. It has been widely accepted in business applications today for linking and synchronizing disparate data sources. SOAP is the standard technology for exchanging information between applications over the Internet. These two technologies are very important to all Clarion developers who want to maintain competitive advantage in the world of Windows and web application development.

Prerequisites

This manual is targeted at intermediate Clarion 5.5 and Clarion 6 programmers with some Win32 programming experience and a basic understanding of COM. It is not intended to be a primer on COM – for that we recommend the following reading list.

Inside COM by Dale Rogerson.
: This book discloses the secrets of COM programming for the advanced engineer. It includes many sample programs on CD-ROM. (Microsoft Press, ISBN 1-57231-349-8)

Essential COM by Don Box
: This text covers the motivation for the design of COM and its distributed aspects. It shows how COM works and contains coverage of the core concepts of distributed COM including detailed descriptions of COM theory and remote servers. It also offers a thorough explanation of COM's basic vocabulary. (Addison Wesley ISBN 0-201-63446-5)

xmlFUSE covers the MS XML 4.0 SDK and the MS SOAP 3.0 SDK. For in depth discussion of XML, DOM, SAX, XSLT, and the MS XML 4.0 SDK we recommend the following reading list.

XML Step by Step, Second Edition by Michael J. Young.
: This book guides you through the process of creating XML documents and displaying them on the Web. It covers XML, XSLT and the MSXML 4.0 SDK. (Microsoft Press, ISBN 0-7356-1465-2)

The Plugware COM classes and OutlookFUSE product are written exclusively for Clarion 5.5H and Clarion 6. Earlier versions of Clarion are not currently supported because of improvements in the compiler to support interfaces and passing GROUP structures by value.

Installation

To install xmlFUSE, simply execute the supplied xmlfuse.exe file and follow the instructions in the installation program. xmlFUSE consists of Clarion source files, templates, documentation and example applications. The default installation directory (\C55) is located based on your Clarion 5.5 installation. If you choose \C55 as the installation directory the installer will place files in the following order:

\C55\Libsrc – The following source files for the Plugware COM classes and the xmlFUSE wrapper will default to this directory:

pwapi.inc
pwapifnc.inc
pwcomdef.inc
pwcom.inc
pwcom.clw
pwcom.exp
pwheap.inc
pwheap.clw
msoap.inc
msoap.clw
msoap1.inc
msoap1.clw
msoapdef.inc
msoapiid.inc
msoapint.inc
msxml2.inc
msxml2.clw
msxml21.inc
msxml21.clw
msxml2def.inc
msxml2iid.inc
msxml2int.inc

\C55\Docs\Lib – xmlFUSE library files for Win32 calls including:

ole32.lib
oleaut32.lib
olepro32.lib

\C55\Docs\xmlFUSE – xmlFUSE documentation including:

xmlFUSE.pdf

\C55\Template – xmlFUSE global extension templates including:

xmlfuse.gif
xmlfuseabc.tpl
xmlfuseleg.tpl

\C55\Examples\xmlFUSE – Source files for the example applications including:

xmlfuse.dct
xmlfuse.app
invoice.xml
invoice.xsl
readme.txt
thinkdata.ico

Plugware COM Overview

The Plugware COM classes, written by Plugware Solutions.com Ltd., provide the Clarion developer with a set of classes to encapsulate the COM API natively in Clarion. Now any developer can write pure Clarion source code to interface with COM objects using native early or late binding without worrying about stability, performance, or flexibility. So without further ado let us explore the Plugware COM classes!

Plugware COM ships as seven Clarion source files and one export file for multi-DLL and hand-coded applications. These files are:

pwapi.inc – Contains a large number of the Win32 API data types and constants. It is included by the pwcomdef.inc file.

pwapifnc.inc – Contains a large number of the Win32 API function prototypes for Clarion. It is included by the pwcomdef.inc file.

pwcomdef.inc – Contains the common interfaces (IUnknown, IDispatch, ITypeInfo), common data types for implementing COM automation, and the function prototypes for Win32 API calls used by the COM classes. It is included by the pwcom.inc file.

pwcom.inc – Contains the class definitions for the early and late binding implementations of generic COM objects including COM string classes, variant factory object, safe array support and dispatch interface wrapper.

pwcom.clw – Contains the source implementation of the classes defined in pwcom.inc.

pwheap.inc – Contains the class definition for the PWHeap class which is used by the string classes to allocate memory properly. It is included by the pwcom.clw file.

pwheap.clw – Contains the source implementation of the class defined in pwheap.inc.

pwcom.exp – Contains the exports for the Plugware COM classes and methods used in multi-DLL and hand-coded Clarion application development.

String Classes

There are four classes provided to deal with strings in COM. Figure 1.1 shows the inheritance relationships of these classes.

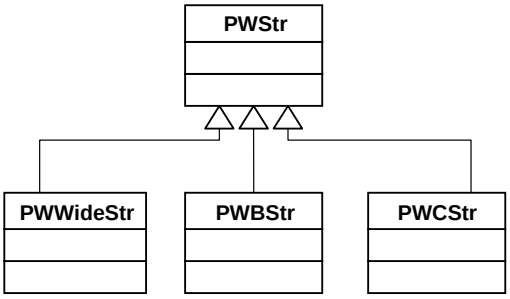

FIGURE 1.1 The relationship between the parent PWStr class and the derived classes PWWideStr, PWBStr (BSTR or binary string support in COM) and PWCStr

The PWStr is the parent class of the three other classes. It contains a virtual destructor which calls the PWStr.Release method to handle freeing the memory for the string. The PWWideStr class is used internally by the PWBStr class to allocate memory and map a character string to a wide-character (Unicode) string. The two classes of interest to the developer are the PWBStr and PWCStr classes.

The PWBStr class was written to support the COM BSTR data type. A BSTR is a pointer to a wide character string and is sometimes referred to as a Basic string or binary string. The PWBStr can take a Clarion cstring or string in its initialization method and the PWBStr.GetStr() method will be called when the developer wishes to pass a BSTR to a COM object.

The PWCStr class is designed to retrieve a Clarion CSTRING from a PWBStr object. It will be used frequently to retrieve string output parameters from calls to COM objects. It operates in the reverse of the PWWideStr by calling the WideCharToMultiByte API function to map a wide-character (Unicode) string to a character string.

A helper function called _cstr has been added to Plugware COM to support the automatic conversion of BSTR return values to a Clarion CSTRING. It is prototyped as follows:

_cstr(long bstrVal, short fFreeBStr = true),*cstring

The _cstr function is passed the BSTR parameter and returns the reference to a newly created Clarion CSTRING.

Helper Classes

There are five classes provided for making the process of interfacing to the Win32 COM API easier:

PWCOMIniter

The PWCOMIniter should be the first object instantiated on each thread using the Plugware COM classes. It calls the Win32 API function CoInitialize() to initialize COM for that thread. When its destructor is called the CoUnInitialize() API function is called. The PWCOMIniter must be instantiated before any of the other Plugware COM classes to ensure that its destructor method is called after all other classes have destructed.

PWCOMError

The PWCOMError class is responsible for translating COM errors returned as HRESULTs into something readable to the developer or end user. The method of interest to the developer inside the PWCOMError class is:

PWCOMError.GetError procedure(*cstring szErrorMsg, *long dwBufferLen, HRESULT hr),long – This method fills the passed szErrorMsg with the results from the Windows API error message for the passed HRESULT. This method uses the FormatMessage API function internally.

PWDateTime

The PWDateTime class converts Clarion dates and times into values compatible with COM objects. The methods of interest to the developer inside the PWDispatch class are:

PWDateTime.Init procedure(*tVariant vtTime) – This method takes a variant of type VT_DATE and initializes the PWDateTime structure with its contents.

PWDateTime.Init procedure(*real systime) – This method takes a Clarion real containing a COleDateTime date/time value and initializes the PWDateTime structure with its contents.

PWDateTime.Init procedure(*SYSTEMTIME systime) – This method takes a Windows API SYSTEMTIME structure and initializes the PWDateTime with its contents.

PWDateTime.Init procedure(date cwDate, time cwTime) – This method initializes the PWDateTime structure with a Clarion date and time. Use this method in conjunction with PWDateTime.GetAsCOleDateTime to pass dates to COM objects.

PWDateTime.GetAsCOleDateTime procedure(),real – This method returns the contents of the PWDateTime structure as a real which is equivalent to the COleDateTime class used in the Microsoft Foundation Classes (MFC). Most COM objects will accept dates in this format.

PWDateTime.GetAsClarionDate procedure(),date – This method returns the contents of the PWDateTime structure as a Clarion DATE. Use this method in conjunction with PWDateTime.GetAsClarionTime to get the complete date/time component.

PWDateTime.GetAsClarionTime procedure(),time – This method returns the contents of the PWDateTime structure as a Clarion TIME. Use this method in conjunction with PWDateTime.GetAsClarionDate to get the complete date/time component.

PWDateTime.Now procedure() – This method sets the PWDateTime structure to the current time using the GetLocalTime API function.

PWInvokeHelper

The PWInvokeHelper class is used in late binding automation to handle parameter passing to COM object methods via the PWDispatch.Invoke method. Plugware COM provides a number of different versions of PWDispatch.Invoke which will be discussed in the Late Binding Automation section of this documentation. These Invoke methods will cover calling methods with up to 10 parameters; therefore, one will generally not need to manipulate a PWInvokeHelper object directly. The easiest way to understand the functionality of PWInvokeHelper is to follow the code in the PWDispatch.Invoke methods in pwcom.clw.

PWSafeArray

The PWSafeArray encapsulates the functionality of the SAFEARRAY data type. A SAFEARRAY is an array which includes boundary information. This provides the developer with the size and dimensions of the array, thus eliminating the possibility of out of bounds addressing errors. The PWSafeArray class effectively wraps the SafeArray COM API functions defined in OLEAUT32.DLL. The methods of interest to the developer inside the PWCOMError class are:

PWSafeArray.Attach procedure(procedure(*tVariant vtsa, short fSelfCleaning = true),short) – This method attaches a PWSafeArray object to a variant returned from a call to a COM object method. It returns true if the attach operation succeeds and false if it fails.

PWSafeArray.Attach procedure(* SAFEARRAY sa, short fSelfCleaning) – Attaches a PWSafeArray object to a data structure of type _SAFEARRAY. It returns true if the attach operation succeeds and false if it fails.

PWSafeArray. Create procedure(VARTYPE vt, long ulCount, long lLBound)– Creates a new array of type vt (see pwcomdef.inc for a list of valid VARTYPEs) with a size and the lower bound specification. It makes calls internally to SafeArrayCreate to create the new array descriptor and allocate and initialize the data type for the array.

PWSafeArray. Copy procedure(*HRESULT hr),*PWSafeArray - Copies the existing PWSafeArray and returns a reference to the newly created PWSafeArray.

PWSafeArray.GetType procedure,long – Returns the type of SafeArray contained in the PWSafeArray object. See pwcomdef.inc for a list of the valid VARTYPEs.

PWSafeArray.GetLowerBound procedure(long uDim = 0),long – Returns the lower bound element of the SafeArray. This return value is zero based.

PWSafeArray.GetUpperBound procedure(long uDim = 0),long – Returns the upper bound element of the SafeArray. This return value is zero based.

PWSafeArray.GetDimensions procedure(*long xElems, *long yElems, *long nDims) – Returns the number of dimensions of the array and the number of elements in the x and y dimensions.

PWSafeArray.GetDimension procedure(long nDim, *long nElems) – Returns the zero based number of elements in the dimension specified by the first parameter. Pass 0 as the first parameter to get the number of elements in the 1st dimension of the array.

PWSafeArray.AccessData procedure – Retrieves a pointer to the array data and increments the lock count of the array. You must call PWSafeArray.UnaccessData once you are finished manipulating the data after calling this method.

PWSafeArray.UnaccessData procedure – Calls SafeArrayUnaccessData internally to decrement the lock count of the array. You should call this method once you are finished manipulating the data after calling PWSafeArray.AccessData.

Early Binding Automation

The PWCOMObject class defined in pwcom.clw is responsible for handling the low level instantiation of COM objects and the proper release of those interfaces once the developer is finished using them. The PWDispatch class used in Late Binding Automation is actually derived from the PWCOMObject and will be covered in the next section.

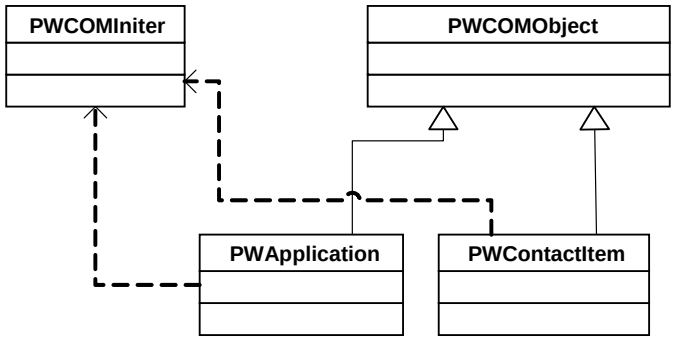

FIGURE 1.2 The relationship between the PWCOMObject and two of the Outlook COM objects, PWApplication and PWContactItem. The dotted arrows indicate that the child objects are reliant on a PWCOMIniter being instantiated for the current thread.

The methods of interest to the developer inside the PWCOMObject class are:

PWCOMObject.AttachExisting procedure(RCLSID rclsid, REFIID riid) - Attaches the object to an existing instance of the CLSID passed in, if it exists, and attaches itself to the IID passed as the second parameter.

PWCOMObject.CreateInstance procedure(REFCLSID rclsid, REFIID riid, long dwClsContext = CLSCTX_ALL) – Uses the CoCreateInstance API call to create an instance of the CLSID passed in. It attaches itself to the IID passed as the second parameter. The third parameter is the context in which the code that manages the newly created object will run in. It is taken from the CLSCTX itemized equates defined in pwcomdef.inc.

PWCOMObject.GetInterface procedure(long pInterface, REFIID riid, *long pvObject, short fRelease) – This method returns an interface from a pointer to an IDispatch or IUnknown interface. The first parameter contains the long pointer to the IDispatch interface passed in. The second parameter contains the IID of the interface we wish to get back from the internal call to QueryInterface. The third parameter will contain a pointer to the output interface. The fourth parameter is a true/false flag where a true will cause the internally referenced IUnknown to release. This method will be used most frequently in the early binding automation when a procedure call returns a pointer to an IDispatch.

PWCOMObject.QueryInterface procedure(REFIID riid, *long pvObject) – This method calls QueryInterface on the object wrapped in the current PWCOMObject. This might be used when attaching PWCOMObject to IUnknown interfaces to determine what kind of interface is contained in the object.

PWCOMObject.AddRef procedure – This is equivalent to calling IUnknown.AddRef and is used internally by the PWCOMObject class.

PWCOMObject.Release procedure – This method releases the interface contained in the PWCOMObject and is equivalent to calling IUnknown.Release. It maintains an internal reference count to ensure that the object is released at the proper time.

PWCOMObject.Attach procedure(long pUnk, short fNoAddRef = false) – This method attaches the object to a passed pointer to an IUnknown interface. The first parameter contains the long pointer to the IUnknown interface we wish to attach to. The second parameter determines whether we should call IUnkown.AddRef on this interface.

Late Binding Automation

The PWDispatch class defined in pwcom.clw is the derived late bound version of the PWCOMObject class. It encapsulates much of the functionality of working with IDispatch interfaces. PWDispatch has been designed to abstract complexity away from the development of COM automation solutions by providing a simple interface with functions to perform type conversions and variant parameter passing. Figure 1.3 shows the relationship between PWDispatch and the classes it relies upon to get its job done.

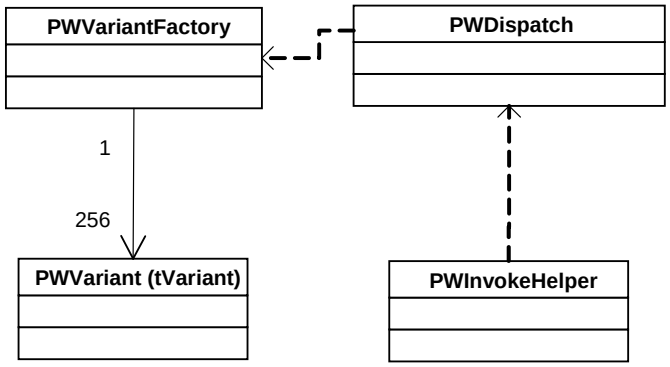

FIGURE 1.3 The PWVariantFactory object is a dependency of the PWDispatch class because variants are cached in the global PWVariantFactory when handling the Invoke calls to the COM object inside PWDispatch. PWVariantFactory can have a maximum of 256 cached variants at a time. PWInvokeHelper can be used independently of the PWVariantFactory to prepare parameter lists and call Invoke on them.

PWVariantFactory is a class used by a global helper function called _vt. The _vt function will convert Clarion data types into variants suitable for passing to COM objects. The PWVariantFactory is globally instantiated and it is responsible for caching the variants created when _vt is called. The PWVariantFactory stores a maximum of 256 cached variants to conserve memory (this can be edited in the PWVariantFactory.Construct method in pwcom.clw) so you will see references to PWVariantFactory.Release() in the examples to clear the cache for re-use.

The PWDispatch class relies on internal PWInvokeHelper objects inside its Invoke methods to handle initializing the parameter lists before calling IDispatch.Invoke. There are four versions of the PWDispatch.Invoke class which are of particular note. They take 0, 1, 5, and 10 variant parameters respectively. This allows the COM developer to pass optional parameters to procedures which take a long list. For those rare instances in which you will need to pass more than 10 parameters to a COM procedure you will need to instantiate your own PWInvokeHelper and use the self.SetParam method similarly to the way it is used in the PWDispatch.Invoke method for 10 parameters declared in pwcom.inc.

The methods of interest to the developer inside the PWDispatch class are:

PWDispatch.CreateInstance procedure(*cstring szObject) – Take a passed cstring containing the name of the object to instantiate (i.e. 'Outlook.Application' for Microsoft Outlook) and handles the internal calls into the COM API to call CoCreateInstance on the proper CLSID.

PWDispatch.CreateInstance procedure(*cstring szProgID, long flags) – An overloaded version of the CreateInstance above which takes a second parameter containing the context in which the code that manages the newly created object will run in. It is taken from the CLSCTX itemized equates defined in pwcomdef.inc.

PWDispatch.AttachExisting procedure(RCLSID rclsid) - Attaches the object to an existing instance of the CLSID passed in, if it exists. The PWDispatch will then be attached to the IDispatch interface of the object referenced by the CLSID.

PWDispatch.AttachExisting procedure(*cstring szProgID) – An overloaded version of the AttachExisting above which takes a cstring containing the name of the existing instance of an object to attach to (i.e. 'Outlook.Application' will attach to an existing instance of Microsoft Outlook).

PWDispatch.Attach procedure(long pIDisp, short fNoAddRef) – This method attaches the object to a passed pointer to an IDispatch interface. The first parameter contains the long pointer to the IDispatch interface we wish to attach to. The second parameter determines whether we should call IDispatch.AddRef on this interface.

PWDispatch.Attach procedure(*tVariant vtDisp, short fNoAddRef) – An overloaded version of the Attach above which takes a variant parameter containing an IDispatch interface instead of a long pointer to the interface. This allows us to attach a PWDispatch to an output parameter from a call to PWDispatch.Invoke.

PWDispatch.Attach procedure(*IDispatch IDisp, short fNoAddRef) – An overloaded version of the Attach above which takes an IDispatch interface parameter rather than a long pointer or a variant.

We alluded to the Invoke methods earlier and discussed the fact that there are several different versions depending on the number of parameters the COM method may take. Optional parameters are handled easily with the variant vtMissing declared globally in the Plugware COM classes. vtMissing can be used to pass empty or optional parameters to the COM methods. The next section covers the use of the PWDispatch.Invoke method (which is a wrapper around the IDispatch.Invoke COM method) and gives examples of using the vtMissing variant to pass empty or optional parameters.

PWDispatch.Invoke procedure(*cstring szMember, long wFlags, *tVariant vtRet, short fDispatchPut) – This version of PWDispatch.Invoke takes no input parameters. The first parameter is a cstring containing the name of the member method or property to

invoke. The second parameter is a flag for IDispatch::Invoke and can be one of the following constants declared in pwcomdef.inc:

DISPATCH_METHOD – This Member is a COM object method to execute
DISPATCH_PROPERTYGET – This Member is a property we wish to retrieve
DISPATCH_PROPERTYPUT – This Member is a property we wish to put
DISPATCH_PROPERTYPUTREF – This Member is a property we wish to but by reference, not by value

The third parameter is a variant output parameter from the Invoke call. It will contain the return value from the call to IDispatch.Invoke once it has completed. The fourth parameter is a true/false flag and should be set to true only when the COM object expects a put-by-reference rather than a put-by-value. An example call is listed below:

```
szMember = 'Count'
hr = DispInboxItems.Invoke(szMember, DISPATCH_PROPERTYGET, vtIDisp)
```

PWDispatch.Invoke procedure(*cstring szMember, long wFlags, *tVariant vtParam1, *tVariant vtRet, short fDispatchPut = false) – This version of PWDispatch.Invoke is identical to the one above except it takes one input parameter called vtParam1. The other parameters are treated the same as the version which takes no input parameters. An example call is listed below:

```
szMember = 'GetDefaultFolder'
hr = DispNamespace.Invoke(szMember, DISPATCH_METHOD, _vt(lolFolderInbox), vtIDisp)
```

PWDispatch.Invoke procedure(*cstring szMember, long wFlags, long nParams, *tVariant vtParam1, *tVariant vtParam2, *tVariant vtParam3, *tVariant vtParam4, *tVariant vtParam5, *tVariant vtRet, short fDispatchPut) – This version of PWDispatch.Invoke is identical to the one above except it takes five input parameters called vtParam1, vtParam2, vtParam3, vtParam4, and vtParam5. This method can be used for procedures which take between 1 and 5 parameters because we can pass a vtMissing for parameters which we wish to omit. An example call listed below uses this version to call a method requiring only one parameter:

```
szMember = 'GetDefaultFolder'
hr = DispNamespace.Invoke(szMember, DISPATCH_METHOD, _vt(lolFolderInbox), _vt(vtMissing), _vt(vtMissing),
_vt(vtMissing), _vt(vtMissing), vtIDisp)
```

PWDispatch.Invoke procedure(*cstring szMember, long wFlags, long nParams, *tVariant vtParam1, *tVariant vtParam2, *tVariant vtParam3, *tVariant vtParam4, *tVariant vtParam5, *tVariant vtParam6, *tVariant vtParam7, *tVariant vtParam8, *tVariant vtParam9, *tVariant vtParam10, *tVariant vtRet, short fDispatchPut) – This version of PWDispatch.Invoke is identical to the one above except it takes ten input parameters called vtParam1, …, vtParam10. This method can be used for procedures which take between 1 and 10 parameters using the vtMissing variant.

Multi-DLL Considerations

Plugware COM provides support for compiling the classes into a multi-DLL project. All Plugware COM products ship with the pwcom.exp file installed into the Libsrc directory. This export file should be merged into the export file of the DLL containing your base class declarations and data. Once you have done this the Plugware COM objects can be referenced from any DLL or EXE which references this base DLL.

The linking of the classes is handled using project pragma settings similar to the way the ABC classes are exported in Clarion. A list of the pragmas for exporting the Plugware COM class definitions is described below for those who prefer to hand code their project in Clarion:

Applications Compiled with Data Local to the Module

%#pragma define(_APIDllMode_=>off)

%#pragma define(_APILinkMode=>on)

%#pragma define(_COMDllMode_=>off)

%#pragma define(_COMLinkMode_=>on)

Applications Compiled with Data External to the Module

%#pragma define(_APIDllMode_=>on)

%#pragma define(_APILinkMode=>off)

%#pragma define(_COMDllMode_=>on)

%#pragma define(_COMLinkMode_=>off)

xmlFUSE Overview

xmlFUSE consists of wrapper classes for the Microsoft XML 4.0 SDK and the Microsoft SOAP 3.0 SDK. The fourteen source files which form the xmlFUSE product and the early bound version wrappers for MS XML 4.0 and MS SOAP 3.0 are:

msxml2int.inc – Contains the Microsoft XML 4.0 COM object model interfaces defined with Clarion compatible prototypes. It is included by the msxml2.inc and msxml21.inc file.

msxml2iid.inc – Contains the definitions for the CLSID and IID values for the entire Microsoft XML 4.0 COM object model. It is included by the msxml2int.inc file.

msxml2def.inc – Contains the Microsoft XML 4.0 COM object model constants defined as Clarion equates. It is included by the msxml2int.inc file.

msxml2.inc – Contains the Plugware COM classes to implement the interfaces defined in msxml2int.inc.

msxml2.clw – Contains the source implementation of the classes defined in msxml2.inc.

msxml21.inc – Contains the second portion of the Plugware COM classes to implement the interfaces defined in msxml21.inc.

msxml21.clw – Contains the source implementation of the class defined in msxml21.inc.

msoapint.inc – Contains the Microsoft SOAP 3.0 COM object model interfaces defined with Clarion compatible prototypes. It is included by the msoap.inc and msoap1.inc file.

msoapiid.inc – Contains the definitions for the CLSID and IID values for the entire Microsoft SOAP 3.0 COM object model. It is included by the msoapint.inc file.

msoapdef.inc – Contains the Microsoft SOAP 3.0 COM object model constants defined as Clarion equates. It is included by the msoapint.inc file.

msoap.inc – Contains the Plugware COM classes to implement the interfaces defined in msoapint.inc.

msoap.clw – Contains the source implementation of the classes defined in msoap.inc.

msoap1.inc – Contains the second portion of the Plugware COM classes to implement the interfaces defined in msxml21.inc.

msoap1.clw – Contains the source implementation of the class defined in msoap1.inc.

This manual assumes that you have access to the Microsoft XML 4.0 Parser SDK and the Microsoft SOAP 3.0 Toolkit documentation contained in the installations from Microsoft. If you have not installed the toolkits, please visit the following links:

MSXML 4.0 Service Pack 2 (Microsoft XML Core Services)

http://www.microsoft.com/downloads/details.aspx?familyid=3144B72B-B4F2-46DA-B4B6-C5D7485F2B42&displaylang=en

Microsoft SOAP Toolkit 3.0 Download

http://www.microsoft.com/downloads/details.aspx?FamilyId=C943C0DD-CEEC-4088-9753-86F052EC8450&displaylang=en

List of Interfaces

The following table lists the interfaces contained in the xmlFUSE product along with their equivalent Microsoft XML or SOAP interface name and their file location. Due to the size of the wrapper and Clarion's limitations with respect to the number of class definitions in each source file, xmlFUSE has been split up into four source files with each one having a maximum of 40 class definitions corresponding to an interface in either MS XML or MS SOAP. Please refer to the Microsoft XML 4.0 Parser SDK and the Microsoft SOAP Toolkit User Guide for more details on what each interface accomplishes when automating MS XML and MS SOAP.

xmlFUSE Interface	Microsoft Interface Name	Source File	Interface Type
PWIAttachment	IAttachment	msoap.inc	**MS SOAP 3.0 Interface**
PWIByteArrayAttachment	IByteArrayAttachment	msoap.inc	**MS SOAP 3.0 Interface**
PWIComposerDestination	IComposerDestination	msoap.inc	**MS SOAP 3.0 Interface**
PWIDataEncoder	IDataEncoder	msoap.inc	**MS SOAP 3.0 Interface**
PWIDataEncoderFactory	IDataEncoderFactory	msoap.inc	**MS SOAP 3.0 Interface**
PWIDimeComposer	IDimeComposer	msoap.inc	**MS SOAP 3.0 Interface**
PWIDimeParser	IDimeParser	msoap.inc	**MS SOAP 3.0 Interface**
PWIDSOControl	IDSOControl	msxml2.inc	**MS XML 4.0 Interface**
PWIEnumSoapMappers	IEnumSoapMappers	msoap.inc	**MS SOAP 3.0 Interface**
PWIEnumWSDLOperations	IEnumWSDLOperations	msoap.inc	**MS SOAP 3.0 Interface**
PWIEnumWSDLPorts	IEnumWSDLPorts	msoap.inc	**MS SOAP 3.0 Interface**
PWIEnumWSDLService	IEnumWSDLService	msoap.inc	**MS SOAP 3.0 Interface**
PWIErrorInfo	IErrorInfo	msoap.inc	**MS SOAP 3.0 Interface**
PWIFileAttachment	IFileAttachment	msoap.inc	**MS SOAP 3.0 Interface**
PWIGCTMObjectFactory	IGCTMObjectFactory	msoap.inc	**MS SOAP 3.0 Interface**
PWIGetComposerDestination	IGetComposerDestination	msoap.inc	**MS SOAP 3.0 Interface**
PWIGetParserSource	IGetParserSource	msoap.inc	**MS SOAP 3.0 Interface**
PWIHeaderHandler	IHeaderHandler	msoap.inc	**MS SOAP 3.0 Interface**

PWIMessageComposer	IMessageComposer	msoap.inc	**MS SOAP 3.0 Interface**
PWIMessageParser	IMessageParser	msoap.inc	**MS SOAP 3.0 Interface**
PWIMXAttributes	IMXAttributes	msxml2.inc	**MS XML 4.0 Interface**
PWIMXNamespaceManager	IMXNamespaceManager	msxml2.inc	**MS XML 4.0 Interface**
PWIMXNamespacePrefixes	IMXNamespacePrefixes	msxml2.inc	**MS XML 4.0 Interface**
PWIMXReaderControl	IMXReaderControl	msxml2.inc	**MS XML 4.0 Interface**
PWIMXSchemaDeclHandler	IMXSchemaDeclHandler	msxml2.inc	**MS XML 4.0 Interface**
PWIMXWriter	IMXWriter	msxml2.inc	**MS XML 4.0 Interface**
PWIParserSource	IParserSource	msoap.inc	**MS SOAP 3.0 Interface**
PWIReceivedAttachment	IReceivedAttachment	msoap.inc	**MS SOAP 3.0 Interface**
PWIReceivedAttachments	IReceivedAttachments	msoap.inc	**MS SOAP 3.0 Interface**
PWISAXAttributes	ISAXAttributes	msxml2.inc	**MS XML 4.0 Interface**
PWISAXContentHandler	ISAXContentHandler	msxml2.inc	**MS XML 4.0 Interface**
PWISAXDeclHandler	ISAXDeclHandler	msxml2.inc	**MS XML 4.0 Interface**
PWISAXDTDHandler	ISAXDTDHandler	msxml2.inc	**MS XML 4.0 Interface**
PWISAXEntityResolver	ISAXEntityResolver	msxml2.inc	**MS XML 4.0 Interface**
PWISAXErrorHandler	ISAXErrorHandler	msxml2.inc	**MS XML 4.0 Interface**
PWISAXLexicalHandler	ISAXLexicalHandler	msxml2.inc	**MS XML 4.0 Interface**
PWISAXLocator	ISAXLocator	msxml2.inc	**MS XML 4.0 Interface**
PWISAXXMLFilter	ISAXXMLFilter	msxml2.inc	**MS XML 4.0 Interface**
PWISAXXMLReader	ISAXXMLReader	msxml2.inc	**MS XML 4.0 Interface**
PWISchema	ISchema	msxml2.inc	**MS XML 4.0 Interface**
PWISchemaAny	ISchemaAny	msxml2.inc	**MS XML 4.0 Interface**
PWISchemaAttribute	ISchemaAttribute	msxml2.inc	**MS XML 4.0 Interface**
PWISchemaAttributeGroup	ISchemaAttributeGroup	msxml2.inc	**MS XML 4.0 Interface**
PWISchemaComplexType	ISchemaComplexType	msxml2.inc	**MS XML 4.0 Interface**
PWISchemaElement	ISchemaElement	msxml2.inc	**MS XML 4.0 Interface**
PWISchemaIdentityConstraint	ISchemaIdentityConstraint	msxml2.inc	**MS XML 4.0 Interface**
PWISchemaItem	ISchemaItem	msxml2.inc	**MS XML 4.0 Interface**
PWISchemaItemCollection	ISchemaItemCollection	msxml2.inc	**MS XML 4.0 Interface**
PWISchemaModelGroup	ISchemaModelGroup	msxml2.inc	**MS XML 4.0 Interface**
PWISchemaNotation	ISchemaNotation	msxml2.inc	**MS XML 4.0 Interface**
PWISchemaParticle	ISchemaParticle	msxml2.inc	**MS XML 4.0 Interface**
PWISchemaStringCollection	ISchemaStringCollection	msxml2.inc	**MS XML 4.0 Interface**
PWISchemaType	ISchemaType	msxml2.inc	**MS XML 4.0 Interface**
PWISentAttachments	ISentAttachments	msoap.inc	**MS SOAP 3.0 Interface**
PWISequentialStream	ISequentialStream	msoap.inc	**MS SOAP 3.0 Interface**
PWIServerXMLHTTPRequest	IServerXMLHTTPRequest	msxml2.inc	**MS XML 4.0 Interface**
PWIServerXMLHTTPRequest2	IServerXMLHTTPRequest2	msxml2.inc	**MS XML 4.0 Interface**
PWISimpleComposer	ISimpleComposer	msoap.inc	**MS SOAP 3.0 Interface**
PWISimpleParser	ISimpleParser	msoap.inc	**MS SOAP 3.0 Interface**
PWISoapClient	ISoapClient	msoap.inc	**MS SOAP 3.0 Interface**
PWISoapConnector	ISoapConnector	msoap.inc	**MS SOAP 3.0 Interface**
PWISoapConnectorFactory	ISoapConnectorFactory	msoap.inc	**MS SOAP 3.0 Interface**
PWISoapError	ISoapError	msoap.inc	**MS SOAP 3.0 Interface**
PWISoapErrorInfo	ISoapErrorInfo	msoap.inc	**MS SOAP 3.0 Interface**

PWISoapMapper	ISoapMapper	msoap.inc	**MS SOAP 3.0 Interface**
PWISoapReader	ISoapReader	msoap.inc	**MS SOAP 3.0 Interface**
PWISoapSerializer	ISoapSerializer	msoap.inc	**MS SOAP 3.0 Interface**
PWISoapServer	ISoapServer	msoap.inc	**MS SOAP 3.0 Interface**
PWISoapTypeMapper	ISoapTypeMapper	msoap.inc	**MS SOAP 3.0 Interface**
PWISoapTypeMapperFactory	ISoapTypeMapperFactory	msoap.inc	**MS SOAP 3.0 Interface**
PWIStream	IStream	msoap.inc	**MS SOAP 3.0 Interface**
PWIStreamAttachment	IStreamAttachment	msoap.inc	**MS SOAP 3.0 Interface**
PWIStringAttachment	IStringAttachment	msoap.inc	**MS SOAP 3.0 Interface**
PWIVBMXNamespaceManager	IVBMXNamespaceManager	msxml2.inc	**MS XML 4.0 Interface**
PWIVBSAXAttributes	IVBSAXAttributes	msxml2.inc	**MS XML 4.0 Interface**
PWIVBSAXContentHandler	IVBSAXContentHandler	msxml2.inc	**MS XML 4.0 Interface**
PWIVBSAXDeclHandler	IVBSAXDeclHandler	msxml2.inc	**MS XML 4.0 Interface**
PWIVBSAXDTDHandler	IVBSAXDTDHandler	msxml2.inc	**MS XML 4.0 Interface**
PWIVBSAXEntityResolver	IVBSAXEntityResolver	msxml2.inc	**MS XML 4.0 Interface**
PWIVBSAXErrorHandler	IVBSAXErrorHandler	msxml21.inc	**MS XML 4.0 Interface**
PWIVBSAXLexicalHandler	IVBSAXLexicalHandler	msxml21.inc	**MS XML 4.0 Interface**
PWIVBSAXLocator	IVBSAXLocator	msxml21.inc	**MS XML 4.0 Interface**
PWIVBSAXXMLFilter	IVBSAXXMLFilter	msxml21.inc	**MS XML 4.0 Interface**
PWIVBSAXXMLReader	IVBSAXXMLReader	msxml21.inc	**MS XML 4.0 Interface**
PWIWSDLBinding	IWSDLBinding	msoap1.inc	**MS SOAP 3.0 Interface**
PWIWSDLMessage	IWSDLMessage	msoap1.inc	**MS SOAP 3.0 Interface**
PWIWSDLOperation	IWSDLOperation	msoap1.inc	**MS SOAP 3.0 Interface**
PWIWSDLPort	IWSDLPort	msoap1.inc	**MS SOAP 3.0 Interface**
PWIWSDLReader	IWSDLReader	msoap1.inc	**MS SOAP 3.0 Interface**
PWIWSDLService	IWSDLService	msoap1.inc	**MS SOAP 3.0 Interface**
PWIXMLAttribute	IXMLAttribute	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDocument	IXMLDocument	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDocument2	IXMLDocument2	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMAttribute	IXMLDOMAttribute	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMCDATASection	IXMLDOMCDATASection	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMCharacterData	IXMLDOMCharacterData	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMComment	IXMLDOMComment	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMDocument	IXMLDOMDocument	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMDocument2	IXMLDOMDocument2	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMDocumentFragment	IXMLDOMDocumentFragment	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMDocumentType	IXMLDOMDocumentType	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMElement	IXMLDOMElement	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMEntity	IXMLDOMEntity	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMEntityReference	IXMLDOMEntityReference	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMImplementation	IXMLDOMImplementation	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMNamedNodeMap	IXMLDOMNamedNodeMap	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMNode	IXMLDOMNode	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMNodeList	IXMLDOMNodeList	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMNotation	IXMLDOMNotation	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMParseError	IXMLDOMParseError	msxml21.inc	**MS XML 4.0 Interface**

PWIXMLDOMProcessingInstruction	IXMLDOMProcessingInstruction	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMSchemaCollection	IXMLDOMSchemaCollection	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMSchemaCollection2	IXMLDOMSchemaCollection2	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMSelection	IXMLDOMSelection	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLDOMText	IXMLDOMText	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLElement	IXMLElement	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLElement2	IXMLElement2	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLElementCollection	IXMLElementCollection	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLError	IXMLError	msxml21.inc	**MS XML 4.0 Interface**
PWIXMLHTTPRequest	IXMLHTTPRequest	msxml2.inc	**MS XML 4.0 Interface**
PWIXSLProcessor	IXSLProcessor	msxml21.inc	**MS XML 4.0 Interface**
PWIXSLTemplate	IXSLTemplate	msxml21.inc	**MS XML 4.0 Interface**
PWIXTLRuntime	IXTLRuntime	msxml21.inc	**MS XML 4.0 Interface**
PWXMLDOMDocumentEvents	XMLDOMDocumentEvents	msxml21.inc	**MS XML 4.0 Interface**

xmlFUSE Procedures

xmlFUSE ships with a number of procedures in the example application which are intended to be reused. They aid in serializing Clarion data structures to XML and converting XML back into GROUPs, QUEUEs or FILEs. These procedures are also good learning tools for learning to write your own specialized XML code in Clarion. The procedures and their purpose are as follows:

XMLAppendChildNode (*PWIXMLDOMNode xmlNodeSource, *PWIXMLDOMNode xmlNodeDest),*PWIXMLDOMNode – This procedure wraps the PWIXMLDOMNode.appendchild method call defined in msxml21.inc. It is passed a source node (xmlNodeSource) and a destination node (xmlNodeDest). It appends the source node to the destination node and returns a reference to the new child node successfully appended to the list. It returns NULL if no object is created. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***CreateXMLUsingDOM***

XMLAppendChildTextNode (*cstring szText, *PWIXMLDOMDocument2 xmlDoc, *PWIXMLDOMNode xmlNode),*PWIXMLDOMText – This procedure wraps the PWIXMLDOMDocument.createTextNode and PWIXMLDOMNode.appendchild method calls defined in mxsml21.inc. It is passed a Clarion CSTRING (szText), a created XML Document (xmlDoc), and a destination node (xmlNode). It appends a child text node to the destination node and returns a reference to the newly created text node. **Location:** *xmlfuse.app, procedure* ***ProcessConvertContactsToXML****, method* ***TakeWindowEvent***

XMLConvertDOMtoFILE (*PWIXMLDOMNode xmlParentNode, *file fConvert, long lMapping) - This procedure is passed a parent XML DOM node (xmlNode), a Clarion FILE (fConvert), and the mapping (lMapping) for either mapping by field order (XMLFUSE:MapByOrder) or mapping by field name

(XMLFUSE:MapByName). It will take the children of the parent node and append their text contents into the FILE structure either by field order or by field name. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***ConvertClarionFILEtoDOM***

XMLConvertDOMtoGROUP (*PWIXMLDOMNode xmlNode, *group grpConvert, long lMapping) – This procedure is passed a parent XML DOM node (xmlNode), a Clarion GROUP (grpConvert), and the mapping (lMapping) for either mapping by field order (XMLFUSE:MapByOrder) or mapping by field name (XMLFUSE:MapByName). It will take the children of the parent node and put their text contents into the GROUP structure either by field order or by field name. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***ConvertClarionFILEtoDOM***

XMLConvertDOMtoQUEUE (*PWIXMLDOMNode xmlParentNode, *queue qConvert, long lMapping) - This procedure is passed a parent XML DOM node (xmlNode), a Clarion QUEUE (qConvert), and the mapping (lMapping) for either mapping by field order (XMLFUSE:MapByOrder) or mapping by field name (XMLFUSE:MapByName). It will take the children of the parent node and append their text contents into the QUEUE structure either by field order or by field name. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***ConvertClarionQUEUEtoDOM***

XMLConvertFILEToDOM (*file fConvert, *PWIXMLDOMDocument2 xmlDoc, long lMapping),*PWIXMLDOMNode - This procedure is passed a Clarion FILE (fConvert), a created XML Document (xmlDoc), and the mapping (lMapping) for either mapping by field order (XMLFUSE:MapByOrder) or mapping by field name (XMLFUSE:MapByName). It returns a reference to the parent node of the created XML DOM. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***ConvertClarionFILEtoDOM***

XMLConvertGROUPToDOM (*group grpConvert, *PWIXMLDOMDocument2 xmlDoc, long lMapping),*PWIXMLDOMNode – This procedure is passed a Clarion GROUP (grpConvert), a created XML Document (xmlDoc), and the mapping (lMapping) for either mapping by field order (XMLFUSE:MapByOrder) or mapping by field name (XMLFUSE:MapByName). It returns a reference to the parent node of the created XML DOM. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***ConvertClarionGROUPtoDOM***

XMLConvertGROUPToDOMNodeList (*cstring szdocElementName, *group grpConvert, *PWIXMLDOMDocument2 xmlDoc),*PWIXMLDOMNodeList – This procedure is used to pass complex data types to SOAP methods which require them. It is passed a Clarion CSTRING (szdocElementName), a Clarion GROUP (grpConvert) and a created XML Document (xmlDoc). It returns a

PWIXMLDOMNodeList containing the field pairs for the name and the data. **Location:** *xmlfuse.app, procedure* ***MSSOAPExamples****, routine* ***AmazonSOAPExample***

XMLConvertQUEUEToDOM (*queue qConvert, *PWIXMLDOMDocument2 xmlDoc, long lMapping),*PWIXMLDOMNode – This procedure is passed a Clarion QUEUE (qConvert), a created XML Document (xmlDoc), and the mapping (lMapping) for either mapping by field order (XMLFUSE:MapByOrder) or mapping by field name (XMLFUSE:MapByName). It returns a reference to the parent node of the created XML DOM. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***ConvertClarionQUEUEtoDOM***

XMLCreateAttribute (*cstring szAttribute, *PWIXMLDOMDocument2 xmlDoc),*PWIXMLDOMAttribute – This procedure is a wrapper around the PWIXMLDOMDocument.createAttribute method call defined in msxml21.inc. It is passed a Clarion CSTRING (szAttribute) and a created XML Document (xmlDoc). It returns a newly created PWIXMLDOMAttribute object. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***CreateXMLUsingDOM***

XMLCreateCDATASection (*cstring szData, *PWIXMLDOMDocument2 xmlDoc),*PWIXMLDOMCDATASection – This procedure is a wrapper around the PWIXMLDOMDocument.createCDATASection method call defined in msxml21.inc. It is passed a Clarion CSTRING (szData) and a created XML Document (xmlDoc). It returns a newly created PWIXMLDOMCDATASection object. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***CreateXMLUsingDOM***

XMLCreateComment (*cstring szComment, *PWIXMLDOMDocument2 xmlDoc),*PWIXMLDOMComment – This procedure is a wrapper around the PWIXMLDOMDocument.createComment method call defined in msxml21.inc. It is passed a Clarion CSTRING (szComment) and a created XML Document (xmlDoc). It returns a newly created PWIXMLDOMComment object. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***CreateXMLUsingDOM***

XMLCreateDocumentFragment (*PWIXMLDOMDocument2 xmlDoc),*PWIXMLDOMDocumentFragment – This procedure is a wrapper around the PWIXMLDOMDocument.createDocumentFragment method call defined in msxml21.inc. It is passed a created XML Document (xmlDoc). It returns a newly created PWIXMLDOMDocumentFragment object. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***CreateXMLUsingDOM***

XMLCreateElement (*cstring szElement, *PWIXMLDOMDocument2 xmlDoc),*PWIXMLDOMElement – This procedure is a wrapper around the PWIXMLDOMDocument.createElement method call defined in msxml21.inc. It is passed a Clarion CSTRING (szElement) and a created XML Document (xmlDoc). It returns a newly created PWIXMLDOMElement object. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***CreateXMLUsingDOM***

XMLCreateProcessingInstruction (*cstring szTarget, *cstring szData, *PWIXMLDOMDocument2 xmlDoc),*PWIXMLDOMProcessingInstruction – This procedure is a wrapper around the xmlDoc.createProcessingInstruction method call defined in msxml21.inc. It is passed a Clarion CSTRING (szTarget), a second Clarion CSTRING (szData) and a created XML Document (xmlDoc). It returns a newly created PWIXMLDOMProcessingInstruction object. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***CreateXMLUsingDOM***

XMLCreateTextNode (*cstring szTagName, *PWIXMLDOMDocument2 xmlDoc),*PWIXMLDOMText – This procedure is a wrapper around the PWIXMLDOMDocument.createTextNode method call defined in msxml21.inc. It is passed a clarion CSTRING (szTagName) and a created XML Document (xmlDoc). It returns a newly created PWIXMLDOMText object. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***CreateXMLUsingDOM***

XMLGetFirstChild (*PWIXMLDOMNode xmlNode),*PWIXMLDOMNode – This procedure is a wrapper around the PWIXMLDOMNode.get_firstChild method call defined in msxml21.inc. It is passed a parent DOM node (xmlNode). It returns the first child node contained in the passed parent node. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***ConvertDOMtoClarionGROUP***

XMLGetXMLString (*PWIXMLDOMNode xmlNode),*cstring – This procedure takes a DOM node as an input parameter (xmlNode) and returns a Clarion CSTRING containing the XML for that node and its children. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***ConvertDOMtoClarionGROUP***

XMLLoadDocument (*PWIXMLDOMDocument2 Obj, *cstring szUrl, *short _fResult),HRESULT – This procedure is a wrapper around the PWIXMLDOMDocument.Load method call defined in msxml21.inc. It loads an XML document (szUrl) into the passed PWIXMLDOMDocument2 (Obj) and returns the result of the load operation (_fResult). **Location:** *xmlfuse.app, procedure* ***XMLTransformUsingXSL***

XMLParseUsingSAX (PWISAXXMLReader Reader, *cstring szFilename),HRESULT – This procedure is a wrapper around the PWISAXXMLReader.ParseURL method call defined in msxml2.inc. It parses an XML document whose location is passed in a Clarion CSTRING (szFilename) and gives control of the parsing to the PWISAXXMLReader (Reader) object. **Location:** *xmlfuse.app, procedure* ***XMLTransformUsingXSL***

XMLRemoveChildNode (*PWIXMLDOMNode xmlChild, *PWIXMLDOMNode xmlParent),*PWIXMLDOMNode – This procedure is a wrapper around the PWIXMLDOMNode.removeChild method call defined in msxml21.inc. It takes a child DOM node (xmlChild) and the parent of that child (xmlParent) and removes the child node from the parent's tree. It returns the removed child to the caller. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***CreateXMLUsingDOM***

XMLReplaceChildNode (*PWIXMLDOMNode xmlNewChild, *PWIXMLDOMNode xmlOldChild, *PWIXMLDOMNode xmlParent),*PWIXMLDOMNode – This procedure is a wrapper around the PWIXMLDOMNode.replaceChild method call defined in msxml21.inc. It takes a replacement child DOM node (xmlNewChild), the node to be replaced (xmlOldChild) and the parent node of the child to be replaced (xmlParent). It replaces the node and returns the old child that is replaced to the caller. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***CreateXMLUsingDOM***

XMLTransformUsingXSL (*cstring szXMLFile, *cstring szXSLFile, <*cstring szOutputFile>),*cstring – This procedure takes the path or URL of an XML file (szXMLFile), the path or URL of an XSL file (szXSLFile) and an optional file parameter to save a copy of the transformed result (szOutputFile). It returns the output of the XSL transformation to the caller in a Clarion CSTRING. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***TransformXMLUsingXSL***

GetURL (string sMethod, string sURL, <string sParams>),*cstring – This procedure uses MS XML HTTP to return the contents of an Internet URL. It is passed the HTTP method parameter (sMethod), the URL of the document to retrieve (sURL) and an option parameter containing parameters to pass to the URL (sParams). It returns the contents of the URL to the caller. **Location:** *xmlfuse.app, procedure* ***MSXMLHTTPExamples****, routine* ***GetSiteContents***

PostURL (string sMethod, string sURL, string sParams),*cstring – This procedure uses MS XML HTTP to POST form fields to a web page on the Internet. It is passed the HTTP method parameter (sMethod), the URL of the document to POST to (sURL) and the parameters to POST to the web page (sParams). It returns the response from the HTTP POST request to the caller. **Location:** *xmlfuse.app, procedure* ***MSXMLHTTPExamples****, routines* ***POSTToAuthorizeNET*** *and* ***POSTToThinkDataGuestBook***

GetBytesFromSafeArray (*tVariant vtSafeArray), *string – This procedure converts a SafeArray retrieved from a COM object into a Clarion STRING and will commonly be used to retrieve images from the Internet using MS XML HTTP. **Location:** *xmlfuse.app, procedure* ***DownloadImage***

DownloadImage (string sURL),*string – This procedure wraps the GetBytesFromSafeArray procedure. It is passed the URL of the image or data to return in binary format (sURL). It returns the data as a Clarion STRING to the caller. **Location:** *xmlfuse.app, procedure* ***MSXMLHTTPExamples****, routine* ***DownloadImageFromURL***

StripHTMLTags (*cstring szHTML) – This rudimentary procedure take a Clarion CSTRING and removes all of the HTML tags. It uses regular expressions to accomplish this task. It is not meant to be an exhaustive solution to this problem – it should be treated as a starting point from which to code a more specialized version. **Location:** *xmlfuse.app, procedure* ***MSXMLHTTPExamples****, button* ***?Strip:HTML:Tags***

New Procedures in xmlFUSE version 1.1

XMLGetLastChild (*PWIXMLDOMNode xmlNode),*PWIXMLDOMNode – This procedure is a wrapper around the PWIXMLDOMNode.get_lastChild method call defined in msxml21.inc. It is passed a parent DOM node (xmlNode). It returns the last child node contained in the passed parent node. **Location:** *xmlfuse.app, procedure* ***RSSSupport****, routine* ***FetchRSSFeed***

XMLGetNextSibling (*PWIXMLDOMNode xmlNode),*PWIXMLDOMNode – This procedure is a wrapper around the PWIXMLDOMNode.get_nextSibling method call defined in msxml21.inc. It is passed a parent DOM node (xmlNode). It returns the next sibling node contained in the passed parent node. **Location:** *xmlfuse.app, procedure* ***RSSSupport****, routine* ***FetchRSSFeed***

XMLGetNodeName (*PWIXMLDOMNode xmlNode),*cstring – This procedure is a wrapper around the PWIXMLDOMNode.get_nodeName method call defined in msxml21.inc. It is passed a DOM node (xmlNode). It returns a *cstring containing the name of the node. You must dispose this *cstring when you are finished with it to prevent a memory leak. **Location:** *xmlfuse.app, procedure* ***RSSSupport****, routine* ***FetchRSSFeed***

XMLGetNodeValue (*PWIXMLDOMNode xmlNode),*cstring – This procedure is a wrapper around the PWIXMLDOMNode.get_nodeValue method call defined in msxml21.inc. It is passed a DOM node (xmlNode). It returns a *cstring containing the value of the node. You must dispose this *cstring when you are finished with it to prevent a memory leak. **Location:** *xmlfuse.app, procedure* ***RSSSupport****, routine* ***FetchRSSFeed***

XMLGetNodeText (*PWIXMLDOMNode xmlNode),*cstring – This procedure is a wrapper around the PWIXMLDOMNode.get_text method call defined in msxml21.inc. It is passed a DOM node (xmlNode). It returns a *cstring containing the text value of the node. You must dispose this *cstring when you are finished with it to prevent a memory leak. **Location:** *xmlfuse.app, procedure* ***RSSSupport****, routine* ***FetchRSSFeed***

XMLFindNodeByName (*PWIXMLDOMNode xmlParent, string sNodeName),*PWIXMLDOMNode – This procedure uses several other helper procedures to find a node beneath a parent using its name for identification. It is passed a parent DOM node (xmlNode) and the name of the node to search for

(sNodeName). If the node exists it returns the *PWIXMLDOMNode object, otherwise it returns null. **Location:** *xmlfuse.app, procedure* ***RSSSupport****, routine* ***FetchRSSFeed***

New Procedures in xmlFUSE version 2.0

DownloadURLToFile (string sURL, string sOutputFile),byte,proc – This procedure wraps the GetBytesFromSafeArray procedure. It is passed the URL of the image or data to return in binary format (sURL). It saves the resulting data in the location specified by the sOutputFile parameter. **Location:** *xmlfuse.app, procedure* ***MSXMLHTTPExamples****, routine* ***DownloadURLToFile***

XMLConvertDOMNodeListToCstring (*PWIXMLDOMNodeList xmlNodeList),*cstring – This procedure iterates through an IXMLDOMNodeList and converts the nodes into a Clarion CSTRING. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***XPathExample***

XMLSelectNodes (*PWIXMLDOMNode xmlNode, string sNodeName),*PWIXMLDOMNodeList – This procedure is a wrapper around the PWIXMLDOMNode.selectNodes method call defined in msxml21.inc. It is passed a parent DOM node (xmlNode) and a string containing the path to the node(s) to return. It returns a list of nodes matching the criteria in sNodeName. **Location:** *xmlfuse.app, procedure* ***MSXMLExamples****, routine* ***XPathExample***

XMLSelectSingleNode (*PWIXMLDOMNode xmlNode, string sNodeName),*PWIXMLDOMNode – This procedure is a wrapper around the PWIXMLDOMNode.selectSingleNode method call defined in msxml21.inc. It is passed a parent DOM node (xmlNode) and a string containing the path to the node(s) to return. It returns a single node matching the criteria in sNodeName.

Example Application

The main example application which ships with xmlFUSE is xmlfuse.app. It contains over 50 modules of example code demonstrating many different aspects of XML and SOAP functionality. The documentation does not contain the source code to the entire example due to its size – for questions about it you can visit our support forum. Instead, we will focus on a few key source code examples and will provide an index of features. Some of the source code to the xmlfuse.app is contained in the free demo which can be downloaded from the ThinkData site using the following link:

http://www.thinkdata.com/aspwpadmin/stattrack/includes/dltrack.asp?Title=XMLFUSEDEMO&File=xfdemo.zip

Here are some of the features covered in the examples for xmlFUSE:

Procedure	Functionality
MSXMLExamples	- Convert Microsoft Outlook Data to XML using DOM (Document Object Model) via **ConvertOutlookToXML** procedure routine - Serialize/Deserialize Clarion GROUP data structure to XML via **ConvertClarionGROUPtoDOM** and **ConvertDOMtoClarionGROUP** procedure routines - Serialize/Deserialize Clarion QUEUE data structure to XML via **ConvertClarionQUEUEtoDOM** and **ConvertDOMtoClarionQUEUE** procedure routines - Serialize/Deserialize Clarion FILE to XML via **ConvertClarionFILEtoDOM** and **ConvertDOMtoClarionFILE** procedure routines - Conversion of Microsoft dynamDOM project at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/htm/dom_hdi_6ab8.asp to Clarion via **CreateXMLUsingDOM** procedure routine - XSL transformation of an XML document via **TransformXMLUsingXSL** procedure routine - Parse XML document using SAX via **ParseUsingSAX** procedure routine - XPath navigation via **XPathExample**
MSXMLHTTPExamples	- Fetch web site data via **GetSiteContents** - Search Google using HTTP via **GoogleHTTPSearch** procedure routine - Bill a credit card using Authorize.NET via **POSTToAuthorizeNET** procedure routine - POST an entry to ThinkData's web based Guestbook via **POSTToThinkDataGuestbook** procedure routine - Retrieve Yahoo stock quotes via **GetSingleStockQuote** and **RefreshStockQuotes** procedure routines - Download an Internet image into a Clarion image control via **DownloadImageFromURL** routine - Download a file from the Internet using HTTP or HTTPS into a file on disk via **DownloadURLToFile**

MSSOAPExamples	- Translate text into multiple languages including English, French, Spanish, German, Italian, Portuguese, and Russian using the BabelFish SOAP web service from AltaVista via the **BabelFishSOAPExample** procedure routine (See http://babelfish.altavista.com for more information on the service) - Retrieve Currency Exchange Rates for dozens of countries using XMethods SOAP web service via **CurrencyExchangeRate** procedure routine (See http://www.xmethods.com for more information on the service) - Conversion of Microsoft's DocSample1 C++ example contained in the MS SOAP 3.0 Toolkit via the **DocSample2** procedure routine - Check spelling using the Google Dictionary SOAP web service via the **GoogleSOAPSpellingSuggestionExample** procedure routine. Search Google using the SOAP search service via the **GoogleSOAPSearchExample** procedure routine and place the results in a Clarion QUEUE using the xmlFUSE helper functions (See http://www.google.com/apis/ for more information on the service) - Search Amazon.com books using the Amazon SOAP web service via the **AmazonSOAPExample**. The results are placed in a Clarion QUEUE using the xmlFUSE helper functions. This example also demonstrates the use of SafeArray support in Plugware COM to retrieve images from Amazon.com and place them in the Clarion image control. (See http://www.amazon.com/soap) for more information on the service) - Retrieve United States city, state, zip code, area code, and time zone information using the **AddressInfo** procedure routines . The results are placed in a Clarion QUEUE using the xmlFUSE helper functions. (See http://www.webservicex.net/uszip.asmx) for more information on the service) - GeoMonster Web Services demonstrating United Kingdom, Canadian, and United States Postal Code Information as well as SMTP E-mail Mailbox verification and IP Address Location Lookup.
RSSSupport	- Demonstrates the functionality of a basic RSS (Really Simple Syndication) reader in native Clarion code. The **FetchRSSFeed** routine retrieves images from the RSS feed and can serve as a starting point for building a full featured RSS aggregator. (See http://backend.userland.com/rss for more information regarding RSS)

The following listing is a subset of the code in the example application with comments in bold to describe what the code is doing. You can view more of the source code by opening the xmlfuse.app in Clarion 5.5 or Clarion 6 and taking a look at some of the code yourself. The full version of the product contains the complete source code whereas the demo ships with a subset of the source code. The Clarion source code listing begins below:

XML Examples

ConvertClarionQUEUEtoDOM routine

```
  ! We must CreateInstance on an PWIXMLDOMDocument2 before we can call
  ! any of the xmlFUSE helper functions

  hr = xmlDoc.CreateInstance()
  if hr ~= S_OK then exit.

  ! We can call XMLConvertQUEUEToDOM to serialize the QUEUE into XML DOM
  ! The XMLFUSE:MapByOrder and XMLFUSE:MapByName are constants declared
  ! globally as follows:

  !                         itemize(0)
  ! XMLFUSE:MapByOrder        equate
  ! XMLFUSE:MapByName         equate
  !                         end

  execute choice(?bQUEUEMap)
    xmlNode &= XMLConvertQUEUEToDOM(qStrings, xmlDoc, XMLFUSE:MapByOrder)
    xmlNode &= XMLConvertQUEUEToDOM(qStrings, xmlDoc, XMLFUSE:MapByName)
  end

  ! Return the contents of the serialized XML DOM as a Clarion CSTRING
  sz &= XMLGetXMLString(xmlNode)
  szQUEUEXml = sz
  FormatXMLSpacing(szQUEUEXml)
  dispose(sz)
  dispose(xmlNode)

  ! Release the PWIXMLDOMDocument2 object so we can reuse it later
  xmlDoc.Release()

  display
```

CreateXMLUsingDOM routine

```
  ! The following code constructs the following XML using DOM
  !
  !<?xml version="1.0"?>
  !<?xml-stylesheet type='text/xml' href='dom.xsl'?>
  !<!--sample xml file created using XML DOM object.-->
  !<root created="using dom">
  !        <node1>some character data</node1>
  !        <node2><![CDATA[<some mark-up text>]]></node2>
  !        <node3>
  !                <subNode1/>
  !                <subNode2/>
  !                <subNode3/>
  !        </node3>
  !</root>

  hr = xmlDoc.CreateInstance();if hr ~= S_OK then exit.

  !<?xml version="1.0"?>
  szParam1 = 'xml'
  szParam2 = 'version="1.0"'
  xmlPi &= XMLCreateProcessingInstruction(szParam1, szParam2, xmlDoc)
  xmlNode &= XMLAppendChildNode(xmlPi, xmlDoc)
  dispose(xmlNode)
  dispose(xmlPi)

  !<?xml-stylesheet type='text/xml' href='dom.xsl'?>
  szParam1 = 'xml-stylesheet'
  szParam2 = 'type=''text/xml'' href=''dom.xsl'''
  xmlPi &= XMLCreateProcessingInstruction(szParam1, szParam2, xmlDoc)
  xmlNode &= XMLAppendChildNode(xmlPi, xmlDoc)
```

```
dispose(xmlNode)
dispose(xmlPi)

!<!--sample xml file created using XML DOM object.-->
szParam1 = 'sample xml file created using XML DOM object'
xmlComment &= XMLCreateComment(szParam1, xmlDoc)
xmlNode &= XMLAppendChildNode(xmlComment, xmlDoc)
dispose(xmlNode)
dispose(xmlComment)

! Create the root element (i.e., the documentElement).
szParam1 = 'root'
xmlRoot &= XMLCreateElement(szParam1, xmlDoc)

!<root created="using dom">
! Create a "created" attribute for the root element and
! assign the "using dom" character data as the attribute value.
szParam1 = 'created'
xmlAttribute &= XMLCreateAttribute(szParam1, xmlDoc)

szParam2 = 'using dom'
BStrParam2.Init(szParam2)
vtValue.vt = VT_BSTR
vtValue.iVal = BStrParam2.GetStr()
hr = xmlAttribute.put_value(vtValue)
BstrParam2.Release()

IDOMAttribute &= (xmlAttribute.GetIUnknown())
hr = xmlRoot.setAttributeNode(IDOMAttribute, pvObject)

! Add the root element to the DOM instance.
xmlNode &= XMLAppendChildNode(xmlRoot, xmlDoc)
dispose(xmlNode)
dispose(xmlAttribute)

! Create an element to hold text content.
szParam1 = 'node1'
xmlElement &= XMLCreateElement(szParam1, xmlDoc)

hr = xmlDoc.get_documentElement(pvObject)
if hr = S_OK and pvObject
  xmlDocElement &= new(PWIXMLDOMElement)
  hr = xmlDocElement.Attach(pvObject)
  if hr ~= S_OK then ShowXMLError(xmlDoc);return.
  !  for structured formatting:
  !  pXMLDom->documentElement->appendChild(pXMLDom->createTextNode("\n\t"));
  szParam2 = '<13><10>'
  xmlText &= XMLCreateTextNode(szParam2, xmlDoc)
  xmlNode &= XMLAppendChildNode(xmlText, xmlDocElement)
  dispose(xmlNode)
  dispose(xmlText)

  ! pe->text = "some character data";
  szParam2 = 'some character data'
  BStrParam2.Init(szParam2)
  hr = xmlElement.put_text(BStrParam2.GetStr())
  if hr ~= S_OK then ShowXMLError(xmlDoc);return.
  BStrParam2.Release()

  ! pXMLDom->documentElement->appendChild(pe);
  xmlNode &= XMLAppendChildNode(xmlElement, xmlDocElement)
  dispose(xmlNode)
end
dispose(xmlElement)
```

```
  ! Create an element to hold a CDATA section.
  szParam1 = 'node2'
  xmlElement &= XMLCreateElement(szParam1, xmlDoc)

  szParam2 = '<13><10>'
  xmlText &= XMLCreateTextNode(szParam2, xmlDoc)
  xmlNode &= XMLAppendChildNode(xmlText, xmlDocElement)
  dispose(xmlNode)
  dispose(xmlText)

  szParam2 = '<some mark-up text>'
  xmlCDATASection &= XMLCreateCDATASection(szParam2, xmlDoc)
  xmlNode &= XMLAppendChildNode(xmlCDATASection, xmlElement)
  dispose(xmlNode)
  dispose(xmlCDATASection)

  xmlNode &= XMLAppendChildNode(xmlElement, xmlDocElement)
  dispose(xmlNode)

  dispose(xmlElement)

  ! Create an element to hold three empty subelements.
  szParam1 = 'node3'
  xmlElement &= XMLCreateElement(szParam1, xmlDoc)

  ! for structured formatting
  szParam2 = '<13><10>'
  xmlText &= XMLCreateTextNode(szParam2, xmlDoc)
  xmlNode &= XMLAppendChildNode(xmlText, xmlDocElement)
  dispose(xmlNode)
  dispose(xmlText)

  xmlDocumentFragment &= XMLCreateDocumentFragment(xmlDoc)

!      pdf->appendChild(pXMLDom->createTextNode("\n\t\t"));
  szParam2 = '<13><10>'
  xmlText &= XMLCreateTextNode(szParam2, xmlDoc)
  xmlNode &= XMLAppendChildNode(xmlText, xmlDocumentFragment)
  dispose(xmlNode)
  dispose(xmlText)


!      pdf->appendChild(pXMLDom->createElement("subNode1"));
  szParam2 = 'subNode1'
  xmlSubNode &= XMLCreateElement(szParam2, xmlDoc)
  xmlNode &= XMLAppendChildNode(xmlSubNode, xmlDocumentFragment)
  dispose(xmlNode)
  dispose(xmlSubNode)

!      pdf->appendChild(pXMLDom->createTextNode("\n\t\t"));
  szParam2 = '<13><10>'
  xmlText &= XMLCreateTextNode(szParam2, xmlDoc)
  xmlNode &= XMLAppendChildNode(xmlText, xmlDocumentFragment)
  dispose(xmlNode)
  dispose(xmlText)

!      pdf->appendChild(pXMLDom->createElement("subNode2"));
  szParam2 = 'subNode2'
  xmlSubNode &= XMLCreateElement(szParam2, xmlDoc)
  xmlNode &= XMLAppendChildNode(xmlSubNode, xmlDocumentFragment)
  dispose(xmlNode)
  dispose(xmlSubNode)
```

```
!      pdf->appendChild(pXMLDom->createTextNode("\n\t\t"));
  szParam2 = '<13><10>'
  xmlText &= XMLCreateTextNode(szParam2, xmlDoc)
  xmlNode &= XMLAppendChildNode(xmlText, xmlDocumentFragment)
  dispose(xmlNode)
  dispose(xmlText)

!      pdf->appendChild(pXMLDom->createElement("subNode3"));
  szParam2 = 'subNode3'
  xmlSubNode &= XMLCreateElement(szParam2, xmlDoc)
  xmlNode &= XMLAppendChildNode(xmlSubNode, xmlDocumentFragment)
  dispose(xmlNode)
  dispose(xmlSubNode)

!      pdf->appendChild(pXMLDom->createTextNode("\n\t"));
  szParam2 = '<13><10>'
  xmlText &= XMLCreateTextNode(szParam2, xmlDoc)
  xmlNode &= XMLAppendChildNode(xmlText, xmlDocumentFragment)
  dispose(xmlNode)
  dispose(xmlText)

!      pe->appendChild(pdf);
  xmlNode &= XMLAppendChildNode(xmlDocumentFragment, xmlElement)
  dispose(xmlNode)

!      pXMLDom->documentElement->appendChild(pe);
  xmlNode &= XMLAppendChildNode(xmlElement, xmlDocElement)
  dispose(xmlNode)

!      pXMLDom->documentElement->appendChild(pXMLDom->createTextNode("\n"));
  szParam2 = '<13><10>'
  xmlText &= XMLCreateTextNode(szParam2, xmlDoc)
  xmlNode &= XMLAppendChildNode(xmlText, xmlDocElement)
  dispose(xmlNode)
  dispose(xmlText)

  ! Test of IXMLDOMNode.cloneNode
  ! If VARIANT_TRUE then all child nodes are cloned as well
  ! if VARIANT_FALSE then just this node
  hr = xmlElement.cloneNode(VARIANT_TRUE, pvObject)
  if hr = S_OK and pvObject
    xmlNode &= new(PWIXMLDOMNode)
    hr = xmlNode.Attach(pvObject);if hr ~= S_OK then
ShowXMLError(xmlDoc);return.
    xmlNode2 &= XMLAppendChildNode(xmlNode, xmlElement)
    dispose(xmlClone)
  end

  display

! Test of XMLRemoveChildNode on the cloned node we just added above
  xmlNode &= XMLRemoveChildNode(xmlNode2, xmlElement)
  dispose(xmlNode)
  dispose(xmlNode2)

! Test of XMLReplaceChildNode on the entire xmlElement node
  szParam1 = 'replacechildnodeonnode3'
  xmlNode2 &= XMLCreateElement(szParam1, xmlDoc)
  xmlNode &= XMLReplaceChildNode(xmlNode2, xmlElement, xmlDocElement)
  dispose(xmlNode)
  dispose(xmlNode2)

  dispose(xmlDocumentFragment)
  dispose(xmlElement)
  dispose(xmlDocElement)
```

```
  ! Display the generated XML
  sz &= XMLGetXMLString(xmlDoc)
  szXml = sz
  dispose(sz)

  dispose(xmlRoot)
  xmlDoc.Release()

  display
```

SOAP Examples

```
BabelFishSOAPExample routine
  setcursor(cursor:wait)

  ! Prepare to create an instance of the MS SOAP 3.0 client using late binding
  ! Plugware COM automation This demonstrates the "high-level API" method of
  ! making SOAP requests
  progid = 'MSSOAP.SoapClient30'
  hr = soapclient.CreateInstance(progid)
  ! If we have a problem, show the error and exit the routine
  if hr ~= S_OK then ShowCOMError(hr);exit.

  ! The WSDL (Web Services Description Language) file describes what services
  ! are available to our SOAP request. You can put the URL below into your web
  ! browser to see what a WSDL file looks like.  It is just an XML formatted
  ! language used to describe the web service's capabilities.
  szWSDLFile = 'http://www.xmethods.net/sd/2001/BabelFishService.wsdl'
  szServiceName = 'BabelFishService'
  szPort = 'BabelFishPort'
  szWSMLFile = ''

  ! The following call to PWDispatch.Invoke passes 3 parameters.  This call is
  ! using the 5 parameter version of PWDispatch.Invoke so we pass vtMissing as
  ! the last two parameters.
  szMember = 'mssoapinit'
  hr = soapclient.Invoke(szMember, DISPATCH_METHOD, 3, _vt(szWSDLFile), |
                       _vt(szServiceName), _vt(szPort), _vt(vtMissing), |
                       _vt(vtMissing), vtIDisp)
  if hr ~= S_OK then ShowCOMError(hr).

  get(BabelFishQ, choice(?Language:List))

  ! We wish to translate the text in szTextToTranslate and we are using the
  ! languageID selected in the drop list.  After the call to
  ! PWDispatch.Invoke on the 'BabelFish' method we will check to see if a
  ! BSTR or binary string is returned.  If it is, we use the _cstr helper
  ! function to convert it to a Clarion CSTING and put that into the
  ! szTranslatedText text box.
  szMember = 'BabelFish'
  hr = soapclient.Invoke(szMember, DISPATCH_METHOD, 2,
_vt(BabelFishQ.LanguageID), |
                         _vt(szTextToTranslate), _vt(vtMissing),
_vt(vtMissing), |
                         _vt(vtMissing), vtValue)
  if hr ~= S_OK then ShowCOMError(hr).

  if vtValue.vt = VT_BSTR
    sz &= _cstr(vtValue.iVal)
    szTranslatedText = sz
    dispose(sz)
  end
```

```
  ! We must release the soapclient object if we intend to reuse it
  soapclient.Release()

  ! Releasing VariantFactory is important when using late binding
  ! automation.  VariantFactory caches up to 256 variants created when the
  ! _vt helper function is called
  VariantFactory.Release()

  setcursor()

  display




GoogleSOAPSpellingSuggestionExample routine
  setcursor(cursor:wait)

  ! Prepare to create an instance of the MS SOAP 3.0 client using late binding
  ! Plugware COM automation This demonstrates the "high-level API" method of
  ! making SOAP requests
  progid = 'MSSOAP.SoapClient30'
  hr = soapclient.CreateInstance(progid)
  if hr ~= S_OK then ShowCOMError(hr);exit.

  ! The WSDL (Web Services Description Language) file describes what services
  ! are available to our SOAP request. You can put the URL below into your web
  ! browser to see what a WSDL file looks like.  It is just an XML formatted
  ! language used to describe the web service's capabilities.
  szWSDLFile = 'http://api.google.com/GoogleSearch.wsdl'
  szServiceName = 'GoogleSearchService'
  szPort = 'GoogleSearchPort'
  szWSMLFile = ''

  ! The following call to PWDispatch.Invoke passes 3 parameters.  This call is
  ! using the 5 parameter version of PWDispatch.Invoke so we pass vtMissing as
  ! the last two parameters.
  szMember = 'mssoapinit'
  hr = soapclient.Invoke(szMember, DISPATCH_METHOD, 3, _vt(szWSDLFile), |
                        _vt(szServiceName), _vt(szPort), _vt(vtMissing), |
                        _vt(vtMissing), vtIDisp)
  if hr ~= S_OK then ShowCOMError(hr).

  ! The following call to PWDispatch.Invoke passes 2 parameters.  This call is
  ! using the 5 parameter version of PWDispatch.Invoke so we pass vtMissing as
  ! the last three parameters.

  szMember = 'doSpellingSuggestion'
  hr = soapclient.Invoke(szMember, DISPATCH_METHOD, 2,
_vt(szGoogleDeveloperToken), |
                         _vt(szGoogleSpellingSuggestion), _vt(vtMissing), |
                         _vt(vtMissing), _vt(vtMissing), vtValue)
  if hr ~= S_OK then ShowCOMError(hr).

  ! We wish to check the spelling of the text in szGoogleSpellingSuggestion
  ! After the call to PWDispatch.Invoke on the 'BabelFish' method we will
  ! check to see if a BSTR or binary string is returned.  If it is, we use
  ! the _cstr helper function to convert it to a Clarion CSTING and put that
  ! into the szGoogleSpellingSuggestion

  if vtValue.vt = VT_BSTR
    sz &= _cstr(vtValue.iVal)
    if sz
      case message('Google suggests replacing "'&szGoogleSpellingSuggestion&'"
with "' |
```

```
                    &sz&'"|Do you wish to use Google''s suggestion?', |
                   'Google SOAP Spell Checker', ICON:Question,
BUTTON:Yes+BUTTON:No, |
                   BUTTON:Yes)
      of button:yes
        szGoogleSpellingSuggestion = sz
      end
    else
      message('No Suggestions','Google SOAP')
    end
    dispose(sz)
  end

  ! We must release the soapclient object if we intend to reuse it
  soapclient.Release()

  ! Releasing VariantFactory is important when using late binding
  ! automation.  VariantFactory caches up to 256 variants created when the
  ! _vt helper function is called
  VariantFactory.Release()

  setcursor()

  display
```

Links

We have provided a number of Internet links which we found helpful in the development of the xmlFUSE product and in its daily use. If you find helpful resources in your development process, or have questions for us related to xmlFUSE, we encourage you to submit them to our support forum listed below.

ThinkData Support Forum

http://www.thinkdata.com/forum/

XML Links

Microsoft XML Development Center
http://msdn.microsoft.com/xml

Top XML Sample Code and Tutorial
http://www.topxml.com/xml/learnxml.asp

Aaron Skonnard's XML Resources
http://staff.develop.com/aarons/xmllinks.htm

PerfectXML Sample Code, Articles, and Tutorials
http://www.perfectxml.com/

W3Schools XML Tutorial
http://www.w3schools.com/xml/

SOAP Links

Microsoft SOAP Development Center
http://msdn.microsoft.com/soap

Download Amazon.com SOAP Toolkit
http://associates.amazon.com/exec/panama/associates/join/developer/kit.html

Download Google.com SOAP Toolkit
http://www.google.com/apis/

XMethods.com Web Services
http://www.xmethods.com/

W3Schools SOAP Tutorial
http://www.w3schools.com/soap/

SOAPClient.com Resources and services for SOAP
http://soapclient.com

RSS Links

RSS 2.0 specification
http://backend.userland.com/rss

Syndic8 RSS Feed List
http://www.syndic8.com/

InfoWorld RSS Feeds
http://www.infoworld.com/rss/rss_info.html

NewsMonster
http://newsmonster.org/

Summary

The examples we have shown only scratch the surface of the potential of the xmlFUSE product. There are literally hundreds of SOAP web services which can be consumed by your Clarion application using xmlFUSE to provide many forms of content to your users without a major investment in development. In addition, xmlFUSE is the only product of its kind for Clarion 5.5 and Clarion 6 providing native early and late binding to the MS XML 4.0 Toolkit with a stable and efficient COM layer from Plugware Solutions. Whether you need to import XML from another vendor's database, export Clarion data to XML, or develop something new and novel using an XML layer, xmlFUSE is the only solution you need.

We hope you enjoy xmlFUSE and invite you to join ThinkData's support forum to ask questions and get tips on using it to its fullest potential. Thanks for your interest in xmlFUSE and keep your eyes open for other COM automation products from our FUSE product line.