

Implementing a Grants.Gov System-to-System Interface using the Microsoft .NET Framework

Summary

The National Endowment for the Humanities has developed a Grants.Gov system-to-system interface that may serve as a low-cost model for other agencies using the Microsoft .NET Framework (.NET). NEH is publishing this white paper in the hopes that it will assist other agencies in their efforts to participate in Grants.Gov. If you have any questions or comments about the materials in this paper, please don't hesitate to contact us. Also, to view the latest version of this document (and to see sample code), please see our website at: <http://www.neh.gov/whowere/cio.htm>

Contacts

Questions or comments may be directed to:

Beth Stewart, Information Technology Specialist
National Endowment for the Humanities
bstewart@neh.gov

Brett Bobley, Chief Information Officer
National Endowment for the Humanities
bbobley@neh.gov

website: <http://www.neh.gov/whowere/cio.htm>

Background Information

The National Endowment for the Humanities (NEH) is a small grant-making agency that receives approximately 5,000 grant applications annually from both individual and institutional applicants. With a small number of technical and grant program staff members and a limited Information Technology budget, NEH decided to develop in-house a Grants.Gov system-to-system interface using existing servers and applications known by the agency's IT staff members. A system-to-system interface promised receipt of grant applications submitted via Grants.Gov with minimal oversight and involvement, an important requirement since the agency could not add staff members for the handling of these applications.

With these goals in mind, the system-to-system interface and downloaded applications would need to be developed and accessed with the following existing tools and utilities:

- Microsoft Visual Studio .NET 2003 with the Visual Basic .NET language
- Microsoft SQL Server 2000
- Microsoft Internet Information Services (IIS)
- Microsoft .NET Framework 1.1
- Microsoft Internet Explorer
- Adobe Acrobat Reader

Though NEH IT staff members recognized the relative challenge of developing a .NET system-to-system interface instead of a Java-based interface, the project was pursued because it could be developed with existing tools and managed by system administrators trained in Microsoft technologies.

During development, the following additional low-cost APIs were obtained:

- aspNetMime by Advanced Intellect, described below;
- DynamicPDF Merger for .NET by ceTe software, used to merge application files into a single, printer-friendly application file in PDF format; and
- SharpZipLib by ic#code, an open source ZIP library for .NET.

The NEH system-to-system interface entered its production phase with two pilot grant programs in October 2004. After the conclusion of successful testing, in 2005 all NEH institutional grant programs have been advertised on Grants.Gov. To achieve our goal of minimal staff involvement, the interface automatically downloads applications daily that have a status of "Validated." As applications are downloaded, they become available immediately to staff members who review applications for eligibility and are added to the NEH Grants Management System database.

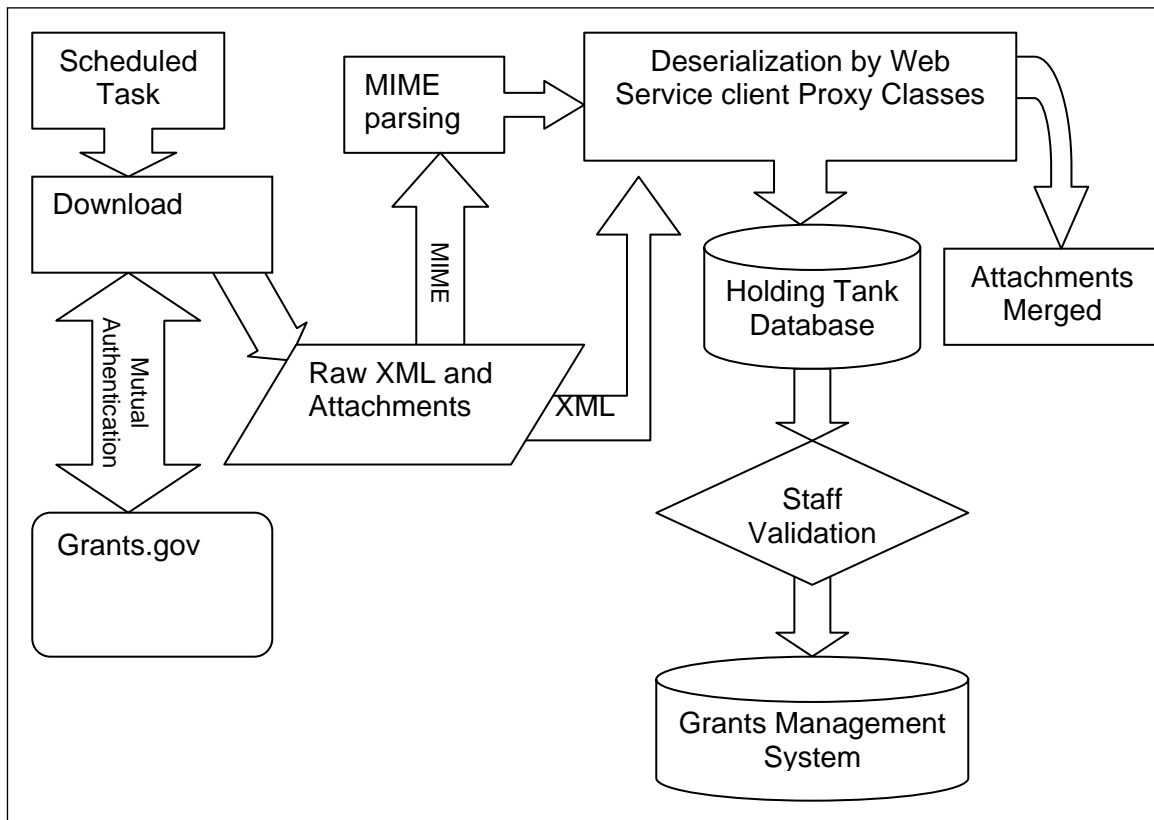
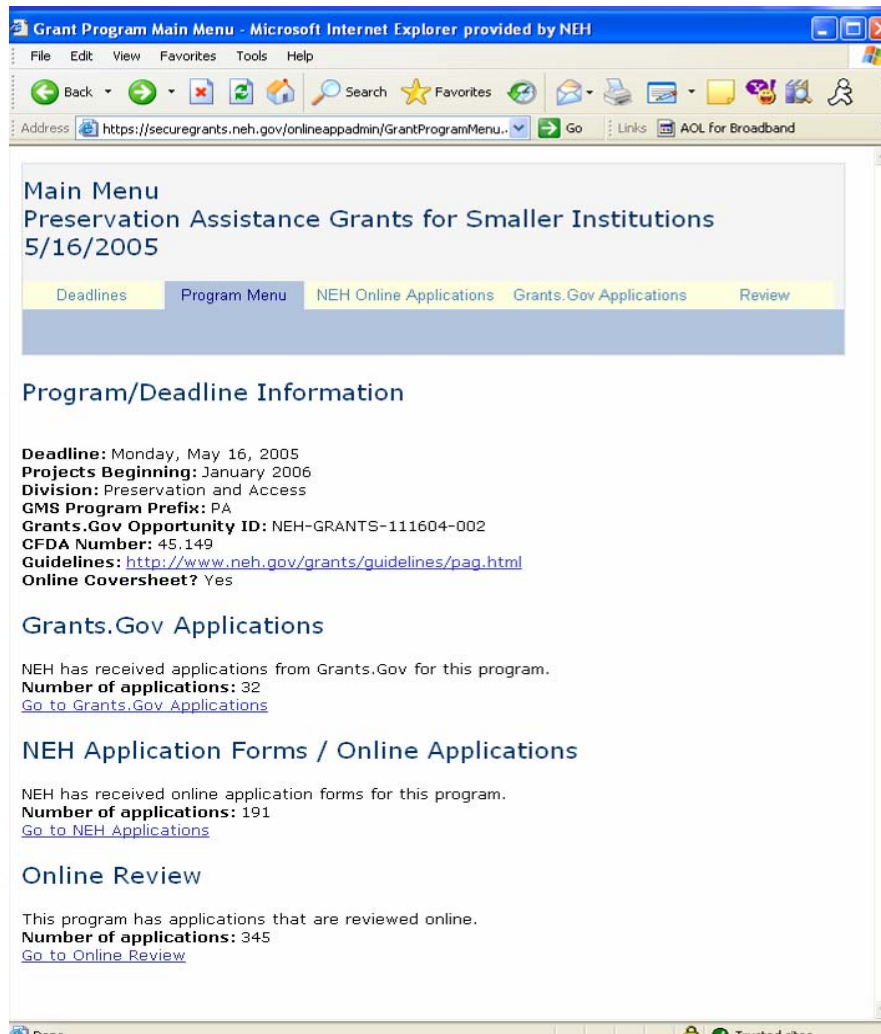


Figure 1. Logical Description of the NEH system-to-system interface

Integration with Existing Systems

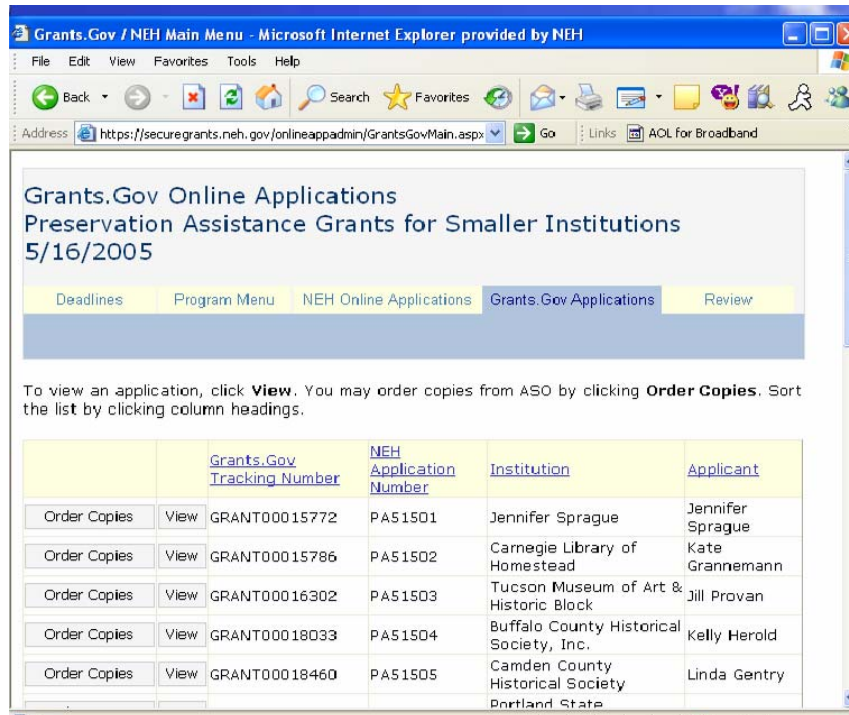
A web-based application system has been in place at NEH since 2002 for the receipt, review, and management of applications from individual applicants, whose numbers total more than half of the applications received by NEH. Because the existing online application system has been widely adopted by NEH grant programs, Grants.Gov applications were tightly integrated with the existing system to encourage adoption and promotion of Grants.Gov as a viable and staff-friendly alternative to NEH online application forms.

A single grant program accessed online offers the following information to staff members:

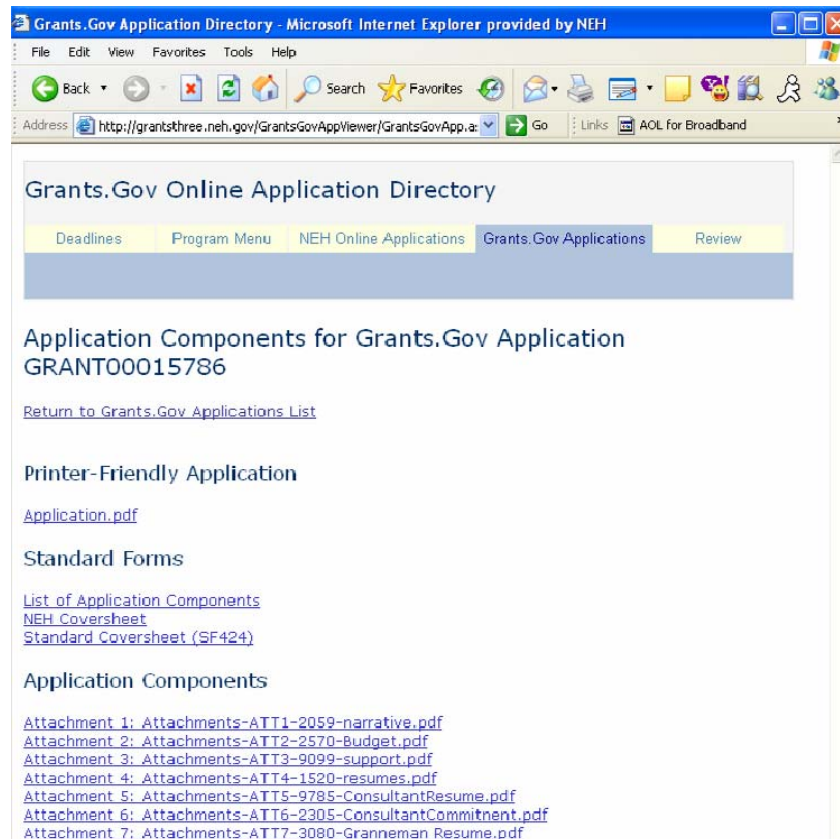


The screenshot shows a web browser window titled "Grant Program Main Menu - Microsoft Internet Explorer provided by NEH". The address bar displays "https://securegrants.neh.gov/onlineappadmin/GrantProgramMenu..". The page content includes a "Main Menu" section with the title "Preservation Assistance Grants for Smaller Institutions 5/16/2005" and a navigation bar with tabs for "Deadlines", "Program Menu", "NEH Online Applications", "Grants.Gov Applications", and "Review". Below this is a "Program/Deadline Information" section with the following details: **Deadline:** Monday, May 16, 2005; **Projects Beginning:** January 2006; **Division:** Preservation and Access; **GMS Program Prefix:** PA; **Grants.Gov Opportunity ID:** NEH-GRANTS-111604-002; **CFDA Number:** 45.149; **Guidelines:** <http://www.neh.gov/grants/guidelines/pag.html>; **Online Coversheet?** Yes. The "Grants.Gov Applications" section states "NEH has received applications from Grants.Gov for this program." and shows "Number of applications: 32" with a link "Go to Grants.Gov Applications". The "NEH Application Forms / Online Applications" section states "NEH has received online application forms for this program." and shows "Number of applications: 191" with a link "Go to NEH Applications". The "Online Review" section states "This program has applications that are reviewed online." and shows "Number of applications: 345" with a link "Go to Online Review".

By following the "Go to Grants.Gov Applications" link, the list of applications that were downloaded via the system-to-system interface is displayed:



A single application consists of a series of separate files and a single file that merges the application components into a printer-friendly PDF format. This single file simplifies printing and offers a way to send Grants.Gov applications to reviewers.



Web-based access to Grants.Gov applications has minimized the involvement of IT staff members in providing copies of applications, as grant programs are able to order copies and even save locally a grant application.

Technical Challenges

Despite the simple access to Grants.Gov applications enjoyed by NEH grant programs, development of a .NET system-to-system interface is far from simple for software developers. Debates over Web Services protocols that split the J2EE and .NET communities resulted in incompatible protocols for binary data that must be resolved to author a successful interface. Below, the technical details of the solution adopted by NEH are described.

SOAP with Attachments vs. WS-Attachments

The SOAP-based specifications of the Web Services world, which depend on the use of XML, do not provide independently a method for sending binary data. Several solutions to sending binary data have been proposed, including SOAP Messages with Attachments (SwA), which has been built into J2EE and by consequence is used by Grants.Gov, and WS-Attachments using DIME, which is supported by .NET. SwA uses multipart MIME to send a SOAP envelope, which must be the root MIME part, in addition to other MIME parts used for transmitting binary data. The following is an example of a SOAP message that uses SwA from the Grants.Gov Agency Integration Toolkit:

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
start="PrimaryMIMEPart"
Content-Description: an optional message description

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <PrimaryMIMEPart>

<?xml version='1.0'>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-
envelope/" >
  <SOAP-ENV:Body>
    ..
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--MIME_boundary
Content-Type: application/pdf
Content-Transfer-Encoding: base64
Content-ID: <Content Id>

... Actual PDF binary data ...

--MIME_boundary
```

.NET provides no support for SwA, and instead recommends use of WS-Attachments and DIME at this time. In addition, the January 2005 W3C Recommendation of the SOAP Message Transmission Optimization Mechanism (MTOM) foretells J2EE and .NET implementations of the new MTOM specification in the future and not Microsoft support for SwA. With no promise of .NET support for SwA, .NET implementations of a Grants.Gov web service client must find a way to handle messages that use SwA without foregoing entirely .NET support of Web Services.

Accessing the MIME-formatted SOAP message however is problematic because the default behavior of the .NET Web Services infrastructure assumes that every SOAP message should be deserialized into objects. These objects are defined by the proxy classes generated when binding to a WSDL document either by adding a web reference to a .NET project within the Visual Studio IDE or by using the wsdl.exe utility. When a .NET web service client receives a MIME-formatted SOAP message, an exception is thrown because the content-type is not the expected text/xml and deserialization does not occur.

SOAP Extensions

Two key challenges arise when consuming MIME-formatted SOAP messages. First, the binary data encapsulated by some of the MIME parts must be extracted and stored according to an agency's requirements. Second, the SOAP message itself must be reformatted so that the .NET Web Services infrastructure is able to deserialize the XML. Both tasks may be accomplished through the use of SOAP Extensions, which have unique access to the streams containing inbound and outbound SOAP messages. A SOAP extension can replace the stream containing the message from a Web service with another to facilitate deserialization, and it can access the stream for logging or, in the case of MIME, to access the parts of the MIME message. A SOAP extension is contained in a class that inherits from the `System.Web.Services.Protocols.SoapExtension` abstract class and overrides a few of its members.

There are four points at which a SOAP extension may execute code, and the format of the SOAP message (XML or objects) varies depending on the selected point. To manage MIME-formatted SOAP messages, the key stage is **BeforeDeserialize**, which occurs when the SOAP message has been received in XML (or MIME) but has not been deserialized. The `ChainStream` and `ProcessMessage` methods are used together to capture the stream at the appropriate time, as a specific example below demonstrates. Once the stream has been captured, a MIME parser may be used to identify the parts of the SOAP message, save binary files, and create a new XML-formatted stream for deserialization.

A SOAP Extension for the `GetApplicationZip()` Method

By way of example, we will examine a SOAP extension class that captures and manipulates the stream sent by Grants.Gov in response to a `GetApplicationZip()` method call. The class is called `GetApplicationZipExt` and does the following:

1. stores the stream received from Grants.Gov,
2. identifies the content type of the SOAP message in the stream to determine whether it is MIME or text/xml,

3. uses a third-party MIME parser to extract and save binary data to the file system, and
4. constructs a new SOAP message from the first MIME part, which is deserialized into object instances of the Web service proxy classes.

```

Public Class GetApplicationZipExt
    Inherits SoapExtension

    Private networkStream As Stream
    Private newStream As Stream
    Private isResponse As Boolean = False
    Private applD As String

    Public Overrides Function ChainStream(ByVal stream As Stream) As Stream
        If (isResponse) Then
            networkStream = stream
            newStream = New MemoryStream
            Return newStream
        Else
            Return stream
        End If
    End Function

    Public Overrides Sub ProcessMessage(ByVal message As _
        System.Web.Services.Protocols.SoapMessage)

        Select Case message.Stage

            ' Capture the tracking number sent as the parameter
            Case SoapMessageStage.BeforeSerialize
                applD = message.GetInParameterValue(0).Grants_govTrackingNumber

            Case SoapMessageStage.AfterSerialize
                isResponse = True

            Case SoapMessageStage.BeforeDeserialize
                If message.ContentType.StartsWith("text/xml") Then
                    ParseXml(message)
                ElseIf message.ContentType.StartsWith("text/html") Then
                    ParseHtml(message)
                Else
                    ParseMime(message)

                    ' Change the content type so that .net will accept the new response
                    message.ContentType = "text/xml"
                End If

            Case SoapMessageStage.AfterDeserialize

            Case Else
                Throw New Exception("Invalid Stage")
            End Select
        End Sub
    End Class

```

In the above code sample, the `ProcessMessage` method sets the `isResponse` boolean as `True` when the `AfterSerialize` stage is reached. Then, when the `ChainStream` method is called again (before the `BeforeDeserialize` stage is reached), the stream—which now contains the response from Grants.Gov—is stored as `networkStream`. Next, `ProcessMessage` is called again, and because we have reached the `BeforeDeserialize` stage the message will be categorized depending on its content type. A content type of "text/xml" will indicate that the SOAP message contains a SOAP Fault, "text/html" indicates a failure to access the Grants.Gov server, and "multipart/related" requires special MIME parsing. Within each parse method, the stream captured within

ChainStream is used, and a new XML-formatted SOAP message is written to the newStream.

Within ParseMime, the networkStream stream is parsed by the third-party MIME parsing software aspNetMime (<http://www.advancedintellect.com/product.aspx?mime>) and binary data is saved to the file system.

```
Dim soapStream As New MemoryStream
Dim utf8 As New UTF8Encoding(False, False)

Dim contentType As Byte() = utf8.GetBytes("Content-type: " + _
    message.ContentType + Chr(13))
soapStream.Write(contentType, 0, contentType.Length)

Dim buffer As Byte() = New Byte(1024) {}
Dim count As Integer = networkStream.Read(buffer, 0, buffer.Length)

While (count > 0)
    soapStream.Write(buffer, 0, count)
    count = networkStream.Read(buffer, 0, buffer.Length)
End While

soapStream.Position = 0

Dim mime As MimeMessage = _
    MimeMessage.ParseStream(soapStream, Encoding.UTF8)
Dim part As MimePart

Dim embeddedParts As MimePartCollection = mime.EmbeddedParts
For Each part In embeddedParts
    mime.GetEmbeddedPart(part.EmbeddedName())

    ' Use the embedded name as the file name
    ' Strip any illegal characters
    Dim eFileName As String
    If (part.EmbeddedName().StartsWith("cid")) Then
        eFileName = New String(part.EmbeddedName().Substring(4))
    Else
        eFileName = New String(part.EmbeddedName())
    End If

    part.SaveAs(eFileName)
Next
```

In the above sample, the contents of networkStream are copied into soapStream so that the stream conforms to a format expected by aspNetMime. Then, the embedded parts of the message, which contain the binary files, are saved. Meanwhile, the first part of the MimeMessage is copied into newStream, as it contains the SOAP message expected by the proxy classes:

```
Dim mpc As MimePartCollection = mime.RetrieveAllParts()
Dim xmlRdr As StreamReader = New StreamReader(mpc(0).DataStream)
Dim strLast As String = "", strCurr As String = ""
Dim byCurr As Byte()

While (strLast <> "</SOAP-ENV:Envelope>")
    Do While (Convert.ToChar(xmlRdr.Peek()) <> ">")
        strCurr += Convert.ToChar(xmlRdr.Read())
    Loop

    ' Read the ">"
    strCurr += Convert.ToChar(xmlRdr.Read())

    ' Copy the xml that can be serialized into newStream
    byCurr = utf8.GetBytes(strCurr)
    newStream.Write(byCurr, 0, byCurr.Length)
```



```

        strLast = Trim(strCurr)
        strCurr = ""
    End While

    ' Reset newStream's position so it is ready for serialization
    newStream.Seek(0, SeekOrigin.Begin)

```

Complete source code for the `GetApplicationZipExt` is available, which includes necessary error handling.

Activating a SOAP Extension

For a SOAP extension to be activated for a particular Web Service method, it is necessary to also author a custom attribute for the class. Our custom attribute class, `GetApplicationZipExtAttribute`, must inherit from the `SoapExtensionAttribute` abstract class and must override its `Priority` and `ExtensionType` properties; the latter is especially important in that it returns the `System.Type` object that defines the SOAP extension class. The `Priority` is simply the order in which the associated `SoapExtension` would gain access to the message in relation to other defined SOAP extensions, as multiple SOAP extensions may access a message.

```

<AttributeUsage(AttributeTargets.Method)> _
Public Class GetApplicationZipExtAttribute
    Inherits SoapExtensionAttribute

    Public Overrides ReadOnly Property ExtensionType() As Type
        Get
            Return GetType(GetApplicationZipExt)
        End Get
    End Property

    Public Overrides Property Priority() As Integer
        Get
            Return 1
        End Get
        Set(ByVal Value As Integer)
        End Set
    End Property
End Class

```

The attribute is associated with the appropriate Web Service method by modifying the method in the proxy class file (`Reference.vb` or `Reference.cs`):

```

<System.Web.Services.Protocols.SoapDocumentMethodAttribute _
("https://ws.grants.gov:446/AgencyIntegration/GetApplicationZip", _
Use:=System.Web.Services.Description.SoapBindingUse.Literal, _
ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Bare), _
GetApplicationZipExtAttribute)> _
Public Function GetApplicationZip
    (<System.Xml.Serialization.XmlElementAttribute _
([Namespace]:="http://apply.grants.gov/WebServices/
AgencyIntegrationServices-V1.0")> _
ByVal GetApplicationZipRequest As GetApplicationZipRequest) As _
<System.Xml.Serialization.XmlElementAttribute _
("GetApplicationZipResponse", _
[Namespace]:="http://apply.grants.gov/WebServices/
AgencyIntegrationServices-V1.0")> GetApplicationZipResponse

    Dim results() As Object = _
        Me.Invoke("GetApplicationZip", New Object() _
        {GetApplicationZipRequest})
    Return CType(results(0), GetApplicationZipResponse)

```

End Function

Once the attribute is added to the GetApplicationZip method within the proxy class file, the SOAP extension will be used when this method is called.

Alternatives

Other implementations of SwA for .NET include:

- **Pocket SOAP**, an open source SOAP client COM component (<http://www.pocketsoap.com/pocketsoap/>). A sample of a Web Service written in ASP.NET using SwA is available at <http://www.pocketsoap.com/weblog/2004/09/1472.html>. A sample Web Service client is not provided.
- **Smart421 Solutions** offers a .NET SOAP Extension that supports SwA (http://www.smart421.com/solutions/smart421_solutions/soap.asp).
- **AlotSoft.com** offers a .NET SOAP Extension that supports SwA (http://www.alotsoft.com/alotsoftweb/soap_attachment.jsp).

Conclusion

Despite the problems that arise while attempting to handle MIME with the .NET infrastructure, the sample implementation outlined here offers one low-cost option that has proven successful for the National Endowment for the Humanities.

References

Balena, Francesco. *Programming Microsoft Visual Basic .NET*. Microsoft Press, 2002.

Bosworth, Adam, et. al. "XML, SOAP, and Binary Data."
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/infoset_whitepaper.asp.

Ewald, Tim. "Accessing Raw SOAP Messages in ASP.NET Web Services."
<http://msdn.microsoft.com/msdnmag/issues/03/03/WebServices/default.aspx>.

Powell, Matt. "Web Services, Opaque Data, and the Attachments Problem."
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/opaquedata.asp>.