XML in Distributed Applications

Objectives

- Understand how to build distributed applications using XML as the format to communicate between tiers.
- Explore the various options for communicating between tiers, to see how XML solves certain problems, but raises issues of its own.
- Learn about the lack of symmetry required by data transfer in distributed applications.
- Examine a sample application that uses different data formats to move data between application tiers.
Building Distributed Applications with XML

Back in the good old days of desktop and even client/server applications, the architecture of an app was straightforward. You didn’t have to worry much about the interface between components, because there were no components that needed an interface. Every app had its own internal communications, hidden away from view.

The simple nature of this environment is what led to its downfall. It was inflexible, limited the user’s options, and grabbed and hung on to network resources with tenacity. It is a model that works for very small workgroups—the kind that is ideal for Microsoft Access—but doesn’t work well when you have to scale up to thousands of users.

The emergence of the Web as a favored application platform hastened the trend to use massively distributed applications. Using text-based and loosely coupled protocols has forced the development of a variety of technologies for breaking an application into components.

Breaking up an application and locating pieces on machines all around the network and world dramatically increases the complexity of inter-application communication. How should data move around the application? What formats should it take on? What data from external applications, over which you have no control, does your application have to handle?

There are plenty of answers to these questions. XML is not always the answer, but many applications can benefit from its use.

Communicating Between Tiers

There are many ways to pass data and invoke methods between components in a distributed application. They can be standard and platform-independent used throughout an industry, or ad hoc or one of a kind format written for a single application for a single company. Some of the more common data formats include:

- **Strings**: Plain text is still the format of champions for its simplicity, but it requires that the component at both ends understand the exact format, usually requiring custom delimiters.

- **Disconnected ADO Recordsets**: Disconnected recordsets solved some of the scalability and other problems of connected recordsets, but they still entail lots of overhead and tie you to Windows.
• **Variant Arrays**: Arrays are easy and flexible, but don’t support specific data types and can be hard to randomly access individual pieces of data. They’re not supported by all development tools.

• **Queued or COM+ Components**: A Windows technology (with very limited exceptions) that automates the marshalling—packaging—of data, but with lots of overhead and complexity.

XML is not the nirvana for communicating between tiers that its press releases might lead you to believe, but it overcomes enough of the limitations of other methods to be an excellent, general approach.

## Why Use XML?

The same reasons that make XML useful in any kind of application make it ideal for general usage as the format between components. Because it is text, it is easily persisted to and from disk files, transferred to a text stream, used with Microsoft Message Queue for latent communication over the wire, and other uses. This makes it incredibly flexible, supporting a wide variety of platforms and applications, including ASP and HTML pages on the Web.

One of the main advantages of XML in the middle tier is that it contains its own metadata—information about the structure of the data. Other formats, such as ADO and COM objects do as well, but variant arrays and strings don’t have this information built in.

Consider the following data stream:

```
Daru Sharp New Jersey Virginia Opinion Leader
```

What does the string “Daru” represent? Is it a person’s first name? Is it a new product from Sharp Electronics? And what’s up with the two state names? (Are they state names?) Are they plant locations?

Here is a Variant array called InputData that packages up the same data for communication over the wire:

```
InputData(0): Daru
InputData(1): Sharp
InputData(2): New Jersey
InputData(3): Virginia
InputData(4): Opinion Leader
```
This adds some structure to the data, making it a bit more likely that New Jersey and Virginia are state names, but it still does not give you much more of a clue.

As in its use in other components, it is much easier to know in XML what data the string “Daru” represents in the data stream, as well as the state names. Here is the XML version of the stream:

```
<?xml version="1.0"?>
<Customers>
    <Customer>
        <FirstName>Daru</FirstName>
        <LastName>Sharp</LastName>
        <HomeState>New Jersey</HomeState>
        <LastStateVisited>Virginia</LastStateVisited>
        <Occupation>Opinion Leader</Occupation>
    </Customer>
</Customers>
```

Now it is pretty easy for either a computer or person to figure out exactly what each string is. Note, however, that the XML version takes up more bytes for the data, although there may be other overhead involved in the Variant array, depending on the particular implementation.

### Disconnectedness

In distributed applications, components and their data must be disconnected from other components and data sources. This means that once a middle tier component receives a chunk of data, the connection to the database or other component can be severed with no effect on the use of the data. This has some significant effects on the architecture of the application:

- **Don’t Use Pointers:** Pass variables, such as strings and arrays, ByVal (by value) in Visual Basic and VBScript, or by copying the actual data in memory without passing a pointer in C++ or C#.
- **Don’t Pass Object References:** It is tempting to define the parameters of middle-tier components as a specific object type, but this can cause headaches and dump scalability in the toilet. This means that XML should be passed as strings rather than as DOM objects.
- **Don’t Require a Persistent Database Connection:** Leaving an open database connection wastes a valuable resource—a client connection—which is not cheap. Retrieve data, send it on its way, and immediately close the database connection.
Forms of XML

There are basically two ways to work with XML between tiers:

- **Plain XML Text**: This is the pure form of XML. Because it is text, it is cross-platform, relatively lightweight, and usable in most applications. This is the XML that its designers meant it to be.

- **XML DOM Objects**: Pre-parsed XML saves the sending component from converting the DOM to XML and the receiving component from parsing the XML text. Each component usually needs to use the same parser to use this method, and the network protocol needs to support the binary format.

Which form of XML data you use depends on the environment of the application. If both ends are running on Windows, both can use MSXML 3.0 and communicate via the Web’s HTTP protocol. This means that you can pass objects containing parsed XML data between application components. MSXML’s ServerXMLHTTP object is designed for such a purpose. (This object is discussed in some detail in the “XML on the Server” chapter in this course.) But if any of these prerequisites don’t apply, you’re better off using XML in text form.

Issues with XML

One of the issues to deal with when using XML is that it isn’t yet a native data type in most tools or platforms. This means that any tier that does more than pass the XML data on, has to do some sort of processing. You learned about some of these issues in the “XML on the Client” chapter. Here’s a quick recap:

- **Tools**: Which tiers have the XML parser and tools? If you can require Internet Explorer 5.0 or later, the client can handle the work. If it’s a public Internet site, you’ll probably need to do it all on the server, or at least check if the browser can handle it.

- **Performance**: Which tier has the processing cycles do the work so the application performs responsively? Can you offload it to a dedicated middle-tier machine?

- **Architectural design**: Which resources outside your application or network must you interact with? Where are the contact points? If you will be working with data on other platforms, how will the application communicate with those resources?

But there is more to consider when using XML throughout a distributed application, particularly on the middle tier.
Uniformity

Repeatedly converting between XML and other formats takes processing cycles, sometimes lots of them. Reading data as a disconnected recordset in the database tier, converting it to XML in the middle tier, applying XSL transformations to apply business rules and maybe incorporating other recordsets, then converting it to HTML on the client tier can eat up a lot of processing power. In general, the fewer different formats an application uses, the better it will perform and the easier it will be to maintain. Ideally, an application would use a single data format throughout, and XML can be an excellent choice.

If you have to deal with many components using many different data formats, one option would be to concentrate the XML processing in a single location, perhaps a single component. This eliminates the need for multiple components to understand XML, at least until more tools fully support it and there are more useful extension technologies, such as XSLT.

Performance

The chapter on performance covers this topic in more detail, but one way to improve the performance of an application that uses XML is to keep the XML data as small and simple as you can. In many cases there is a worse than linear degradation of performance as the size of data grows. This isn’t an issue only with XML, but XML data frequently occupies more bytes than other formats.

Binary formats efficiently pack data and require less bandwidth to transmit across the network. XML seems to be wasteful, but can actually have minimal impact on the network. One consideration is that text with characters using only the first 128 ASCII characters—thus requiring only 7 bits—doesn’t need conversion to transmit over the Internet.

Validation

An issue related to performance is validation. Should you validate data used in a distributed application, using a DTD or XML schema?

- Validation takes extra processing cycles, which hurts parsing performance, but reduces the need for error-checking code in every component that receives XML data.
- Not validating can allow more flexible XML because it doesn’t have to adhere to a rigid format, but requires custom validation in the components that consume the data.

Technology Commitment

Picking one data format over another requires a commitment to that technology. Picking one option frees you from some technologies but binds you to others.
Would you rather use an emerging, platform-independent standard like XML, or a maturing, proprietary, Windows technology like ADO? Microsoft is constantly enhancing ADO and is the database access method of choice in Microsoft products. But it wasn’t that long ago that DAO was in the same position. What will take ADO’s place tomorrow?

Of course, XML, even as an open standard, is not immune to these kinds of changes. But there is much more of a mandate with W3C standards to maintain backwards compatibility. And because the XML committee is made up of vendors from different platforms, it is less likely to change XML in ways that optimizes its use on only a single platform.

XML is the cool technology of the first decade of the new millennium, but will it still be in ten years? Or even two? Obviously, we think that XML is the leading contender for use in distributed applications, but it is a design decision for your specific application and use.

**Stateless Components**

Stateless components are important to distributed applications. You must be sure that your use of XML does not create unintended state-full situations. With stateless components an object is instantiated into memory, a method called with whatever arguments it requires, the component does its work, and is then released from memory. It could also be managed by a pooling service such as MTS or COM+, but as far as the application is concerned it’s history.

With stateless components, you are using network resources very efficiently, since no part of the application holds on to a resource any longer than necessary, freeing the resources for use by other components.

XML in text form is inherently stateless. But it is all too easy to pass an object reference that is kept alive by a programming error that lets a remote component hold onto a reference to the object.

**Network Protocol**

Virtually any network protocol can transmit XML as text data, but not necessarily as binary data, such as a parsed DOM object. Some networks, including the Internet’s TCP/IP, generally deal only with 7-bit data. There are conversions available for 8-bit binary data, but these add overhead to your application during the translation steps at each end.

The incentive for developing the XML standard was the Web, however, so much of XML and its extension standards assume—although may not require—transmission over HTTP. So XML should work fine in any Web-related application.

Be sure that if you select a binary protocol, or text format that uses all 8 bits, it will communicate efficiently over your network.
A Word to the Wise

One final piece of advice in deciding how and where to use XML on the middle tier: Even though there are some good rules of thumb about using XML for particular uses, you have to test it with your data (representative of both size and structure), use your network, and run it with your usage patterns. This is the only way that you can be sure that any data format will meet the design goals of the application.

Also, how well do the various conversions work that you use in the app, such as ADO to XML, XML to XML, and so on?

Performance is not always a primary design goal. What do your users want? Or better, what do they need?
Moving to a distributed application design has many benefits, but one very significant complication is that it creates many data paths through the application. A very simple representation is shown in Figure 1, where the movement of data is indicated by the arrows between objects.

Figure 1. The movement of data among the components of a distributed application.

This logical separation of application tiers with its complex paths of communication, is what enables physical separation of the components. You can conceivably locate each component on a different physical machine on the network. You can even take it further and partition each individual component using the particular tool’s features, such as partitioning a SQL Server database.
Bi-Directional Data

Keep in mind when designing a distributed application between any two components, that you don’t have to communicate data in the same format in both directions. For example, you could read data from a database using an ADO recordset using a Visual Basic data component. That component could persist the data to an ADO stream as XML and send it on to the middle-tier component that requested the data.

When the middle tier has done whatever it needs to do with the data, it could send a variant array back to the data-tier component, which converts it into an XML updategram to update the SQL Server database.

The result consists of three components with four data paths, using four different formats, two of them in XML, as shown in Figure 2. And you can use even more formats to communicate between the middle-tier component and the rest of the application.

![Bi-directional data formats in a distributed application.](image)
Transformations on the Middle Tier: Implementing Business Rules

One of the most useful techniques on the middle tier is to use XSL transformations to implement business rules. XSLT is covered in detail in other chapters in this course, so we won’t repeat that information here.

Imagine this scenario. You are developing a Human Resources application, which by its nature contains extensive, personal, highly sensitive information about the individuals who work at the company. It also interacts with various enterprise systems, including payroll, accounts payable, and the football pool software.

This application will have many different types of users, with many different requirements depending on their job. Take two categories of workers as an example:

- **Payroll Clerk**: This person is responsible for mailing employee paychecks to the many remote work locations each month. Besides stuffing the envelopes, he has to print summary reports, update the database with address changes, and constantly analyze mail destinations to take advantage of lower postage costs. He needs access to a small subset of data, such as employee names and addresses, plus a few other innocuous fields in the database.

- **Chief Financial Officer**: The CFO is responsible for high-level decisions about how to move cash through the organization, and frequently needs both summary and detailed views of employee data, ranging from salary information, records of health claims against the company, and pool betting data. She needs access to a wide variety of custom report formats as well.

It is conceivable that these two employees will have the need for the same report, differing only by level of detail included. This is a perfect situation to use XSLT templates to change what data is delivered to each user.

Consider the diagram in Figure 3. Each of these users runs an application that requests some sort of XML data. The requests are processed in the same way; the only difference is that the CFO is a member of the ExecutiveAdmin group in Windows 2000, while the clerk is a member of the WorkerDrone group. The application treats them the same, and the same XML data is generated.
Figure 3. Modifying the view of XML data based on the category of user.

Once the XML data is available, the security credentials of the two users kicks in, and the clerk’s data is transformed using one XSLT template, while the CFO’s with another. Because these templates filter the data for the clerk and sort the information for the CFO, they get very different formatted reports, but each now has the information to do their job.
A Distributed Example

The sample SharkData application uses a SQL Server database as a source of XML data. A middle-tier component, SharkData.dll receives requests from the user interface ASP pages, creates an ADO recordset to retrieve data, then converts it to various forms. It shows how to use different data formats to pass data between tiers. Figure 4 shows the starting user interface.

Figure 4. The user interface for the distributed application sample.

The various ASP pages that use the Visual Basic middle-tier component call various methods of the Customer object. Here is a representative example:
Dim objCust
Dim varReturn
Dim varCustomerId
Dim varXML
Dim intI, strDoc
Dim xmlDoc, xmlRoot, xmlNode

' Don't cache this page
Response.Expires = -1

varCustomerId = Request("CustomerId")

Set objCust = Server.CreateObject("SharkData.Customer")
' Retrieve all the records from tbCustomer
varReturn = objCust.GetRecordXML(Null, Null, Null, _
    varXML)

Each of the ASP pages goes on to use the return value of these methods to
generate a table with a list of customers for the user to select for update, shown
for CustomersXML.asp in Figure 5.
Each of the functions in the Customer object that return customer records in different formats accepts three input parameters and one output parameter. The input parameters can be used in various combinations:

- If the CustomerID is passed in, the database returns the single record for that customer.
- If the first and last name is passed in, but a null is passed for the CustomerID, the database searches for the record based on the name, and returns what it finds.
- If all three parameters are null, the function returns a list of all customers.

The function uses a Select Case statement to select between the three options. It then builds ADO parameters, if applicable, and calls the appropriate SQL Server stored procedure.
Variant Arrays

The GetRecord function retrieves the list of customers from the database and converts them to a variant array, which is then passed to the ASP page.

```vbp
Public Function GetRecord(ByVal varCustomerId, ByVal varFirstName, ByVal varLastName, ByRef avarRecords) As Boolean

On Error GoTo ErrorHandler

Dim cnnShark As ADODB.Connection
Dim cmdGet As ADODB.Command
Dim rstGet As ADODB.Recordset
Dim strSQL As String
Dim prm As ADODB.Parameter
Dim lngRecords As Long
Dim blnReturn As Boolean

' Assume failure
blnReturn = False

Set cnnShark = New ADODB.Connection
cnnShark.Open conSharkCon
cnnShark.CommandTimeout = 90

Set cmdGet = New ADODB.Command
Set cmdGet.ActiveConnection = cnnShark
cmdGet.CommandType = adCmdStoredProc
```

SharkData.vbp

```xml
<xml>
<database>
<connectionString>conSharkCon</connectionString>
<commandTimeout>90</commandTimeout>
<storedProcName>SharkData</storedProcName>
</database>
</xml>
```
If Not IsNull(varCustomerId) Then
   cmdGet.CommandText = "procGetCustomer"
   
   Set prm = cmdGet.CreateParameter("CustomerId", _,
    adInteger, adParamInput, , Z2Null(varCustomerId))
   cmdGet.Parameters.Append prm
ElseIf Not IsNull(varLastName) _
   And IsNull(varFirstName) Then
   cmdGet.CommandText = "procGetCustomerByName"
   
   Set prm = cmdGet.CreateParameter("LastName", _,
    adVarChar, adParamInput, 50, Z2Null(varLastName))
   cmdGet.Parameters.Append prm

   Set prm = cmdGet.CreateParameter("FirstName", _,
    adVarChar, adParamInput, 20, _
    Z2Null(varFirstName))
   cmdGet.Parameters.Append prm
Else
   cmdGet.CommandText = "procGetCustomer"
End If

Set rstGet = cmdGet.Execute(lngRecords)

If lngRecords <> 0 Then
   avarRecords = rstGet.GetRows()
   blnReturn = True
End If
Disconnected Recordsets

In this case, the Customer object passes an ADO disconnected recordset to the ASP page from the GetRecordRST method.

SharkData.vbp

Public Function GetRecordRst(ByVal varCustomerId, ByVal varFirstName, ByVal varLastName, ByRef varRst) As Boolean

On Error GoTo ErrorHandler

If Not rstGet Is Nothing Then
    rstGet.Close
    Set rstGet = Nothing
End If
If Not cmdGet Is Nothing Then
    Set cmdGet = Nothing
End If
If Not cnnShark Is Nothing Then
    cnnShark.Close
    Set cnnShark = Nothing
End If

ExitHere:
    GetRecord = blnReturn
    Exit Function

ErrorHandler:
    Resume ExitHere

End Function
Dim cnnShark As ADODB.Connection
Dim cmdGet As ADODB.Command
Dim rstGet As ADODB.Recordset
Dim strSQL As String
Dim prm As ADODB.Parameter
Dim lngRecords As Long
Dim blnReturn As Boolean

blnReturn = False

Set cnnShark = New ADODB.Connection
cnnShark.Open conSharkCon
cnnShark.CommandTimeout = 90

Set cmdGet = New ADODB.Command
Set cmdGet.ActiveConnection = cnnShark
cmdGet.CommandType = adCmdStoredProc

If Not IsNull(varCustomerId) Then
    cmdGet.CommandText = "procGetCustomer"
    Set prm = cmdGet.CreateParameter("CustomerId", _
adInteger, adParamInput, , Z2Null(varCustomerId))
    cmdGet.Parameters.Append prm
ElseIf Not IsNull(varLastName) And IsNull(varFirstName) Then
    ' Get record by name
    cmdGet.CommandText = "procGetCustomerByName"
XML in Distributed Applications

Set prm = cmdGet.CreateParameter("LastName", _
adVarChar, adParamInput, 50, Z2Null(varLastName))
cmdGet.Parameters.Append prm

Set prm = cmdGet.CreateParameter("FirstName", _
adVarChar, adParamInput, 20, _
  Z2Null(varFirstName))
cmdGet.Parameters.Append prm
Else
  cmdGet.CommandText = "procGetCustomer"
End If

Set rstGet = New ADODB.Recordset
rstGet.CursorLocation = adUseClient
Set rstGet = cmdGet.Execute(lngRecords)
Set cmdGet.ActiveConnection = Nothing

Set varRst = rstGet
blnReturn = True

ExitHere:
  GetRecordRst = blnReturn
  Exit Function

ErrorHandler:
  blnReturn = False
  Resume ExitHere
End Function

XML

The GetRecordXML method uses the same techniques to extract data from the database, but converts it to XML to pass up to the ASP page.
Public Function GetRecordXML(ByVal varCustomerId, _
    ByVal varFirstName, ByVal varLastName, _
    ByRef varXML) As Boolean

On Error GoTo ErrorHandler

Dim cnnShark As ADODB.Connection
Dim cmdGet As ADODB.Command
Dim rstGet As ADODB.Recordset
Dim strSQL As String
Dim prm As ADODB.Parameter
Dim lngRecords As Long
Dim blnReturn As Boolean

blnReturn = False

Set cnnShark = New ADODB.Connection
    cnnShark.Open conSharkCon
    cnnShark.CommandTimeout = 90

Set cmdGet = New ADODB.Command
Set cmdGet.ActiveConnection = cnnShark
    cmdGet.CommandType = adCmdStoredProc

If Not IsNull(varCustomerId) Then
    cmdGet.CommandText = "procGetCustomer"
    Set prm = cmdGet.CreateParameter("CustomerId", _
        adInteger, _
        adParamInput, , Z2Null(varCustomerId))
    cmdGet.Parameters.Append prm
ElseIf Not IsNull(varLastName) And IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"
ElseIf Not IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"

ElseIf Not IsNull(varLastName) And IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"
ElseIf Not IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"

ElseIf Not IsNull(varLastName) And IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"
ElseIf Not IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"

ElseIf Not IsNull(varLastName) And IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"
ElseIf Not IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"

ElseIf Not IsNull(varLastName) And IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"
ElseIf Not IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"

ElseIf Not IsNull(varLastName) And IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"
ElseIf Not IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"

ElseIf Not IsNull(varLastName) And IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"
ElseIf Not IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"

ElseIf Not IsNull(varLastName) And IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"
ElseIf Not IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"

ElseIf Not IsNull(varLastName) And IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"
ElseIf Not IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"

ElseIf Not IsNull(varLastName) And IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"
ElseIf Not IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"

ElseIf Not IsNull(varLastName) And IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"
ElseIf Not IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"
Set prm = cmdGet.CreateParameter("LastName", _
  adVarChar, _
  adParamInput, 50, Z2Null(varLastName))
cmdGet.Parameters.Append prm

Set prm = _
  cmdGet.CreateParameter("FirstName", adVarChar, _
  adParamInput, 20, Z2Null(varFirstName))
cmdGet.Parameters.Append prm
Else
    cmdGet.CommandText = "procGetCustomer"
End If

Set rstGet = cmdGet.Execute(lngRecords)

If lngRecords <> 0 Then
    varXML = RstToXML(rstGet, "Customers")
    blnReturn = True
End If

If Not rstGet Is Nothing Then
    rstGet.Close
    Set rstGet = Nothing
End If
If Not cmdGet Is Nothing Then
    Set cmdGet = Nothing
End If
If Not cnnShark Is Nothing Then
    cnnShark.Close
    Set cnnShark = Nothing
End If

GetRecordXML = blnReturn

ExitHere:
    Exit Function
ErrorHandler:
   Resume ExitHere

End Function

There are two ASP pages that use XML data. One, CustomersXML.asp, receives the XML data and uses brute force to convert it to HTML. The other, CustomersXSL.asp, takes the XML and applies an XSL template to transform the XML to HTML. The latter approach requires less script code, which is always a good thing.
Summary

- One of the major architectural issues associated with distributed application development is that communication between tiers gets much more complex, and that must be handled for an app to succeed.

- Strings, disconnected ADO recordsets, variant arrays, COM components, and custom solutions are all viable options for passing data between components, but each has its own limitations.

- XML overcomes many of the limitations of other data formats for inter-component communication, making it an ideal general solution. XML is not perfect for every app, but close enough in most cases.

- Various issues may need to be dealt with when using XML with some distributed applications: uniformity, performance, validation, technology commitment, stateless components, and the network protocol.

- Data doesn’t have to move in different directions between the same two components. You can use one format when sending data to a component and a different format when receiving data back.
Questions

1. If you are considering the use of text as the communication format between components, why might you consider using XML instead? Why wouldn’t you?

2. Is a stateless component connected or disconnected?

3. Describe a situation when you should probably use validation when parsing XML data received from an application.

4. True/False/It Depends: XML is a good choice for all the components in any application that must interact with applications on other platforms. Why?

5. Describe a use for XSLT on the middle tier.
Answers

1. If you are considering the use of text as the communication format between components, why might you consider using XML instead? Why wouldn’t you?
   XML adds metadata to the text so that both people and computers can more easily use the data. But it enlarges the size of the data, and all components involved in the communication must understand XML.

2. Is a stateless component connected or disconnected?
   Yes. It depends on what it is doing. While it is instantiated in memory, it may have a persistent connection to a network resource, say to process a million records in a database. But once its work is finished, it releases that connection and leaves memory (or rejoins the object pool in MTS or COM+).

3. Describe a situation when you should probably use validation when parsing XML data received from an application.
   When the data is not reliably structured correctly, and you don’t want to load your code with error handling.

4. True/False/It Depends: XML is a good choice for all the components in any application that must interact with applications on other platforms. Why?
   Generally true, but it really depends. XML is available on almost any platform, so inter-application communication is potentially easier. But it really depends on the needs of the application, and performance and network protocols can be a major factor in the decision.

5. Describe a use for XSLT on the middle tier.
   Filtering data based on the category of user.
Lab 14: XML in Distributed Applications

TIP: Because this lab includes a great deal of typed code, we’ve tried to make it simpler for you. You’ll find all the code in XMLDistributed.TXT, in the same directory as the sample project. To avoid typing the code, you can cut/paste it from the text file instead.
Lab 14 Overview

In this lab you’ll learn how to build and fit together the various pieces of a distributed application using XML. There are a lot of pieces to this lab, as there are in such applications. You’ll see that these tend to be complex, but with the many advantages of distributed applications.

To complete this lab, you’ll need to work through four exercises:

- Create an IIS Virtual Directory
- Create a Visual Basic Middle Tier Component and Test Harness
- Build an ASP Page to Display XML
- Build an ASP Page Using XML and XSLT

Each exercise includes an “Objective” section that describes the purpose of the exercise. You are encouraged to try to complete the exercise from the information given in the Objective section. If you require more information to complete the exercise, the Objective section is followed by detailed step-by-step instructions.

NOTE  If you don’t have Visual Basic available, you can still do the other parts of this lab. In the Completed subdirectory, you can register the SharkDataXML.dll using the regsvr32.exe utility. It is easiest if you copy the utility to the Completed directory. Then open a Command Prompt window and run this line of code:

```
regsvr32.exe SharkDataXML.dll
```
Create an IIS Virtual Directory

Objective

In this exercise, you’ll learn how to create a virtual directory in Internet Information Server 5.0 to run ASP pages. Because ASP runs code on the server, you can’t just load and view an ASP page in a browser like you can a regular HTML page.

Create a virtual directory called XMLDist that points to the real directory where you installed the files for this lab.

This lab assumes that you are using Windows 2000 Professional or Server. The steps in Windows NT are similar, but the lab won’t run properly in Windows 95, 98, or ME.

Things to Consider

- A virtual directory in IIS can point to any directory or Web site you want. In this case, it should point to the location where you installed the files for this lab.
- If you haven’t already installed the lab’s files, now’s a good time to do so. You won’t be able to run the app from the CD.

Step-by-Step Instructions

1. Start the Internet Services Manager. In Windows 2000 Server, select Start|Program Files, Administrative Tools|Internet Services Manager. In Windows 2000 Professional, open the Windows Control Panel, open the Administrative Tools applet, and then start the Internet Services Manager applet.

2. Expand the treeview to the left of the Internet Services Manager until you see the Default Web Site (unless you’ve previously named it something else). It should look something like Figure 6.
3. Right-click on the Default Web Site, and select **New|Virtual Directory** from the popup menu.

4. Click **Next** past the welcome screen to the Virtual Directory Alias screen. Enter **XMLDist** as the alias name. Click **Next** to continue.

5. On the **Web Site Content Directory** screen, browse to the directory where you installed the files for this lab. Click **Next** to continue.

6. Leave all of the default selections on the **Access Permissions** screen, and click **Next** to continue.

7. On the final screen, click **Finish** to complete the wizard and set up the virtual Web directory.

8. Test the virtual directory. Start Internet Explorer, and enter the URL below. This loads the default.htm page (see Figure 7) located in the real directory you entered in the wizard above.

```
http://localhost/XMLdist/
```
Create an IIS Virtual Directory

Figure 7. The default page of the new XMLDist virtual IIS directory.

Shark Doll Company
Customer Management System

Retrieve customers using:

- Variant Arrays
-Disconnected ADO Recordset
-Straight XML
- XML Transformed with XSLT

Note: The various techniques listed above are implemented only to retrieve the list of customers. To keep the sample simple, customer data is updated and added using straight recordsets.
Create a Visual Basic Middle Tier Component and Test Harness

Objective

In this exercise, you’ll create a component that accesses an Access database and returns a set of data as XML data to present on an ASP Web page. You’ll start with a shell application that has some support for returning data as variant arrays and disconnected recordsets. Then you’ll add new features to return XML, and test it in the browser.

Testing and debugging such components in an ASP page is not easy, so you’ll also create a test harness to verify that the component is working correctly before you deploy it to a Web site.

Step-by-Step Instructions

1. Load the SharkDataXML.vbp into Visual Basic from the directory where you installed this lab’s files, and open Customer.cls in the code window.

2. Create a new public function GetRecordXML, and declare some local variables:

```vba
Public Function GetRecordXML(ByVal varCustomerId, ByVal varFirstName, ByVal varLastName, ByRef varXML) As Boolean
On Error GoTo ErrorHandler
Dim cnnShark As ADODB.Connection
Dim cmdGet As ADODB.Command
Dim rstGet As ADODB.Recordset
Dim strSQL As String
Dim prm As ADODB.Parameter
Dim lngRecords As Long
Dim blnReturn As Boolean
```

---

Lab 14:
XML in Distributed Applications

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.
3. The function returns True if the function succeeds, so begin by assuming failure. Then instantiate ADO connection and command objects.

   blnReturn = False
   Set cnnShark = New ADODB.Connection
   cnnShark.Open sSharkCon
   cnnShark.CommandTimeout = 90

   Set cmdGet = New ADODB.Command
   Set cmdGet.ActiveConnection = cnnShark
   cmdGet.CommandType = adCmdStoredProc

4. The function accepts three input parameters and one output parameter. The input parameters can be used in various combinations:

   • If the CustomerID is passed in, the database returns the single record for that customer.
   • If the first and last name is passed in, but a null is passed for the CustomerID, the database searches for the record based on the name, and returns what it finds.
   • If all three parameters are null, the function returns a list of all customers.

The function uses a Select Case statement to select between the three options. It then builds ADO parameters, if applicable, and calls the appropriate querydefs (the Access equivalent to SQL Server stored procedures).
If Not IsNull(varCustomerId) Then
    cmdGet.CommandText = "procGetCustomer"

    Set prm = cmdGet.CreateParameter("customerId", _
        adInteger, adParamInput, , Z2Null(varCustomerId))
    cmdGet.Parameters.Append prm
ElseIf Not IsNull(varLastName) _
    And IsNull(varFirstName) Then
    cmdGet.CommandText = "procGetCustomerByName"

    Set prm = cmdGet.CreateParameter("lastName", _
        adVarChar, adParamInput, 50, Z2Null(varLastName))
    cmdGet.Parameters.Append prm
    Set prm = cmdGet.CreateParameter("firstName", _
        adVarChar, adParamInput, 20, Z2Null(varFirstName))
    cmdGet.Parameters.Append prm
Else
    cmdGet.CommandText = "procGetAllCustomers"
End If

5. Now that everything is set up and ready to go, it’s time to execute the querydef. If the resulting recordset has records, its BOF and EOF properties will be False, and the records are converted to XML. Then the various objects are released from memory.
Set rstGet = cmdGet.Execute()

If Not (rstGet.BOF And rstGet.EOF) Then
    varXML = RstToXML(rstGet, "Customers")
    blnReturn = True
End If

If Not rstGet Is Nothing Then
    rstGet.Close
    Set rstGet = Nothing
End If

If Not cmdGet Is Nothing Then
    Set cmdGet = Nothing
End If

If Not cnnShark Is Nothing Then
    cnnShark.Close
    Set cnnShark = Nothing
End If

6. Time to clean everything up by passing True or False to indicate success or failure, then include some error handling.

    GetRecordXML = blnReturn

ExitHere:
    Exit Function

ErrorHandler:
    Resume ExitHere

End Function

7. To keep things simple, this project uses a brute force method of converting the ADO recordset to XML, looping through and concatenating a string with the XML. It uses some other support functions in the project to create well-formed XML.
Function RstToXML(rst As ADODB.Recordset, _
    strDocElement As String) As String

Dim fld As ADODB.Field
Dim strOut As String

strOut = "<?xml version="1.0"?>"
strOut = strOut & _
    "<" & strDocElement & ">" & vbCrLf

Do While Not rst.EOF
    strOut = strOut & "  <Record>" & vbCrLf
    For Each fld In rst.Fields
        strOut = strOut & Space(4) & _
            CreateElement(fld.Name, fld.Value)
    Next
    strOut = strOut & "  </Record>" & vbCrLf
    rst.MoveNext
Loop
strOut = strOut & "</" & strDocElement & ">

Set fld = Nothing
RstToXML = strOut
End Function

8. Save the project, and compile it using File|Make SharkDataXML.dll. This will verify that there are no syntactical problems.

9. Add a project to the current Visual Basic workspace by selecting File|Add Project. (Don’t select New Project, because that will close the existing project.)

10. In the Add Project dialog box, select the Existing tab and open the SharkTest.vbp project. Click the Save button, and save the group project as SharkTest.vbg.

11. The SharkTest.vbp project consists of a single form shown in Figure 8.
12. Add a reference to the SharkDataXML project by selecting **Project|References**, and checking the box next to **SharkDataXML**. This makes the Customer object available to the test application.

13. The three buttons test various features of the SharkDataXML component. Add the following code to the form’s code window. The Option Explicit statement may have already been added for you by Visual Basic.
Option Explicit

Private Sub cmdDelete_Click()
Dim sdc As SharkData.Customer
Dim fok As Boolean
Dim strMsg As String

Set sdc = New SharkDataXML.Customer

fok = sdc.DeleteRecord(txtCustId.Text)

If fok Then
    strMsg = "Record deleted"
Else
    strMsg = "Didn't work."
End If

MsgBox strMsg, vbInformation
End Sub
Private Sub cmdGetAllCustomers_Click()
Dim sdc As SharkData.Customer
Dim fok As Boolean
Dim strMsg As String
Dim vaCustomers As Variant
Dim i As Integer

Set sdc = New SharkDataXML.Customer
fok = sdc.GetRecord(Null, Null, Null, vaCustomers)

For i = LBound(vaCustomers) To UBound(vaCustomers)
    strMsg = strMsg & vaCustomers(1, i) & ", " & vaCustomers(2, i) & vbCrLf
Next
MsgBox strMsg
End Sub

Private Sub cmdGetOneCustomer_Click()
Dim sdc As SharkDataXML.Customer
Dim fok As Boolean
Dim sCustomers As String

Set sdc = New SharkDataXML.Customer
fok = sdc.GetRecordXML(txtCustId.Text, Null, _
    Null, sCustomers)

If fok Then
    MsgBox sCustomers
Else
    MsgBox "Something stinks!"
End If

End Sub

14. Save your work. Then, in the Visual Basic Project Group window, right-click the SharkTest.vbp file and select Set As Start Up from the popup
menu. When you run the project, this causes the SharkTest project to begin executing, instantiating SharkDataXML as necessary.

15. Run the project, and experiment with the options. In most cases you’ll need to enter a CustomerID in the text box. Try 1 through 10 for existing customer IDs.
Build an ASP Page to Display XML

Objective

In this exercise, you’ll build an ASP page that uses the Visual Basic component you built in the last lab section.

Step-by-Step Instructions

1. Open the existing CustomersXML.asp file, using whatever tool you prefer, such as Visual InterDev if it is installed or Windows Notepad.

2. Delete the following line. This is where you’ll insert the code in the next steps.

INSERT CODE HERE

3. Insert the following code between the <% and %> code delimiters. It starts by declaring some variables, specifying that the page should not be cached so that the data is always fresh, and retrieving the customer ID the user entered.

```vbscript
Dim objCust
Dim varReturn
Dim varCustomerId
Dim varXML
Dim intI, strDoc
Dim xmlDoc, xmlRoot, xmlNode

' Don’t cache this page
Response.Expires = -1

varCustomerId = Request("CustomerId")
```

Now it’s time to use the middle tier component. The first step is to instantiate the SharkDataXML object into memory. Then it calls the
GetRecordXML method with three null arguments, and a variable passed by reference to receive the XML text.

Set objCust = Server.CreateObject("SharkDataXML.Customer")
' Retrieve all the records from tbCustomer
varReturn = objCust.GetRecordXML(Null, Null, Null, varXML)
If Err.Number <> 0 Then
    Response.Write "varReturn = " & varReturn & "<br>
    Response.Write "Error number: " & err.number
    Response.Write "Error description: " & err.description
    Response.Write "<p>"
End If
If varReturn = False Then
    Response.Write "<B>Error: 
    Response.Write "We were unable to retrieve records 
    Response.Write "from the tblCustomer table!</B>"
    Set objCust = Nothing
    Response.End
Else
    ' No need to keep the object around anymore
    Set objCust = Nothing

    Set xmlDoc = Server.CreateObject("Microsoft.xmldom")
    xmlDoc.async = False
    xmlDoc.loadxml(varXML)

The rest of the code loads the XML into a DOM document object, then loops through and dynamically builds the HTML for display.
Set xmlRoot = xmlDoc.documentElement
strDoc = "<TABLE BORDER=1>" & vbCrLf & _
"<TR>" & vbCrLf & _
"<TH>Customer</TH>" & _
"<TH>City</TH>" & _
"<TH>State</TH>" & vbCrLf & _
"</TR>" & vbCrLf

For Each xmlNode In xmlRoot.childNodes
strDoc = strDoc & "<TR>" & vbCrLf & _
"<TD>" & _
"<A HREF=""UpdateCustForm.asp?customerid=" & Null2NBSP(xmlNode.selectSingleNode("CustomerID").text) & _
""">" & _
Null2NBSP(xmlNode.selectSingleNode("LastName").text) & _
" (", " + _
Null2NBSP(xmlNode.selectSingleNode("FirstName").text ) ) & _
"</A></TD>" & vbCrLf & _
"<TD>" & _
Null2NBSP(xmlNode.selectSingleNode("City").text) & _
"</TD>" & vbCrLf & _
"<TD>" & _
Null2NBSP(xmlNode.selectSingleNode("State").text) & _
"</TD>" & vbCrLf & "</TR>"
Next

strDoc = strDoc & "</TABLE>" & vbCrLf
Response.Write strDoc
End If

4. Save the file. Load Internet Explorer, navigate to the following URL, and select the Straight XML link. If everything is working correctly, you should see something like the Web page shown in [Figure 9]
Figure 9. The list of customers extracted using XML.
Build an ASP Page Using XML and XSLT

Objective

In this exercise, you’ll build an ASP page that uses XML data from the SharkDataXML middle-tier component built earlier in this lab. But this time you’ll process the resulting XML data using an XSLT transformation rather than in code as in the last lab section. The results will be the same as in the last section, but will operate much more efficiently.

Step-by-Step Instructions

1. Instead of doing all the work of transforming the XML to HTML in code, you’ll use XSLT to do the work. Start XMLSpy and create a new XSL file by selecting File|New, and selecting the .xslt Extensible Stylesheet Language document.
2. XMLSpy enters the XML declaration and the opening and closing xsl:stylesheet elements. Delete these stylesheet elements, and enter the following XSLT code. This builds a template that extracts specific data items from the source XML data, using an xsl:for-each loop to build the HTML.

```xml
<xsl:template xmlns:xsl="uri:xsl">
  <TABLE Border="1">
    <TR>
      <TH>Customer</TH>
      <TH>City</TH>
      <TH>State</TH>
    </TR>
    <xsl:for-each select="Customers/Record">
      <TR>
        <TD><A>
          <xsl:attribute name="HREF">
            UpdateCustForm.asp?customerid=
            <xsl:value-of select="CustomerID"/>
          </xsl:attribute>
          <xsl:value-of select="LastName"/>
          ,
          <xsl:value-of select="FirstName"/>
        </A></TD>
        <TD><xsl:value-of select="City"/></TD>
        <TD><xsl:value-of select="State"/></TD>
      </TR>
    </xsl:for-each>
  </TABLE>
</xsl:template>
```

3. Save the file as Customer.xsl in the same directory as the other lab files, and close XMLSpy.

4. Open the existing CustomersXSL.asp file, using whatever tool you prefer, such as Visual InterDev if it is installed or Windows Notepad.
5. Delete the following line. This is where you’ll insert the code in the next steps.

INSERT CODE HERE

6. Insert the following code between the <% and %> code delimiters. The first section of code is identical to that in the last section, through the invocation of the GetRecordXML method and handling problems.

On Error Resume Next

Dim objCust
Dim varReturn
Dim varCustomerId
Dim varXML
Dim intI

' Don't cache this page
Response.Expires = -1

varCustomerId = Request("CustomerId")

Set objCust = Server.CreateObject("SharkDataXML.Customer")
' Retrieve all the records from tblCustomer
varReturn = objCust.GetRecordXML(Null, Null, Null, varXML)

If varReturn = False Then
    Response.Write "<B>Error: "
    Response.Write "We were unable to retrieve records "
    Response.Write "from the tblCustomer table!</B>"
    Set objCust = Nothing
    Response.End
The remaining code instantiates two DOM document objects and loads the XML data and the XSLT file, then applies the style to the XML data.

Else

' No need to keep the object around anymore
Set objCust = Nothing

Set xmlDoc = Server.CreateObject("Microsoft.xmldom")
xmlDoc.async = False
xmlDoc.loadxml(varXML)

Set xmlStyle = Server.CreateObject("Microsoft.xmldom")
xmlStyle.async = False
xmlStyle.load(Server.MapPath("Customer.xsl"))

Response.Write
xmlDoc.transformNode(xmlStyle.documentElement)
End If

7. Save the file. Load Internet Explorer, navigate to the following URL, and select the Straight XML link. If everything is working correctly, you should see something like the Web page shown in Figure 10.
Build an ASP Page Using XML and XSLT

Figure 10. The completed customers list using XSLT.
Lab 14:
XML in Distributed Applications