



Legacy Application Integration Inside the Firewall

Microsoft Corporation

Published: November 2002

Abstract

This white paper describes the many ways that you can use Microsoft technologies, tools, and industry standards found in Microsoft® Windows® Server 2003, the Microsoft .NET Framework, and in Microsoft Visual Studio® .NET to integrate various applications and data and to increase productivity. It outlines the considerations involved in creating an integration architecture to help ensure that systems can evolve over time to address changing business needs.

This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2002 Microsoft Corporation. All rights reserved.

Microsoft, SQL Server, Visual Basic, Visual Studio, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

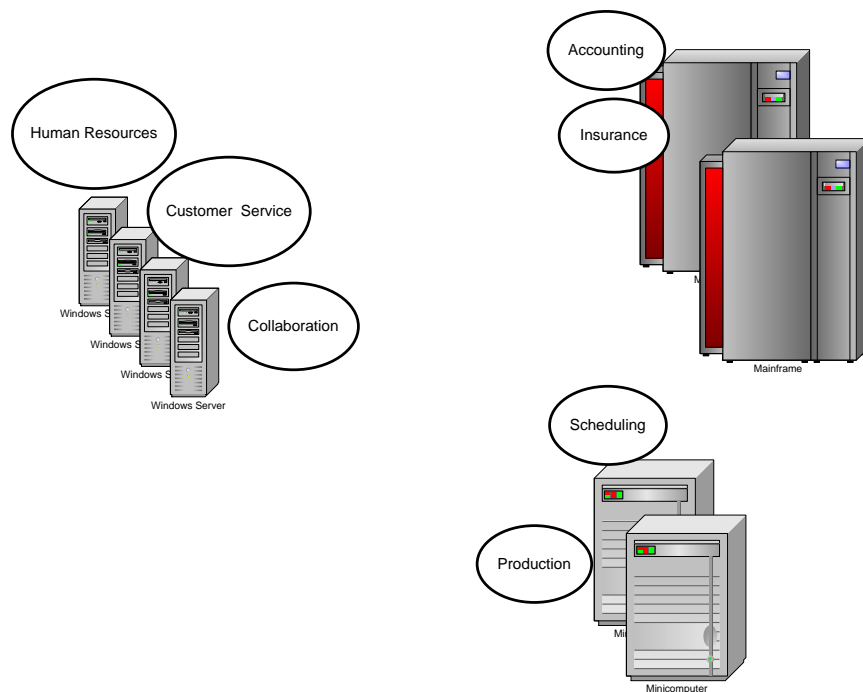
The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Contents

Introduction	4
Technologies for Integration	5
Table 1: Key integration technologies	6
Data Integration	7
Data Access and .NET.....	9
Data on Mainframes.....	12
Connecting to the Host System.....	12
Accessing Relational Mainframe Data	13
Accessing Non-Relational Mainframe Data	14
Host System Integration with SQL Server.....	14
Terminal Emulation	15
Application Integration	16
Transaction Processing	18
Integration with Mainframe CICS and IMS Applications.....	18
Integration With Mainframes Through Message Queuing	19
Integrating Applications through XML Web Services	21
Coordinating Web Service Development with Enterprise UDDI Services	25
Table 2: Scenarios for UDDI Services	25
Installing UDDI Services.....	26
UDDI Data	26
Table 3: UDDI Provider/Business entry information.....	27
Adding Records to UDDI Services	28
UDDI Services Runtime Advantages	28
Process Integration	30
BizTalk Orchestration Services.....	30
BizTalk Messaging Services.....	31
Putting it Together.....	32
Conclusion	34

Introduction

Many organizations have applications in place and they need to integrate them with new applications or simply link existing applications. Users may have direct access to these systems but if they do not integrate well together, users may be forced to enter the same data multiple times or they may not be able to obtain useful information from multiple systems. For instance, if you have several applications on mainframes, several more on UNIX systems, and many more on Windows® systems, how do your users access common data on those systems? In the manufacturing world, the term “islands of automation” was used to describe this scenario where different systems were efficient at what they did but the systems did not communicate together. This type of system is illustrated in the following diagram where each application runs on a particular operating system and does not integrate well or at all with other applications.



Integration projects can be divided into three levels:

- Data
- Applications
- Processes

Each level is not distinct but they usually build on each other. At the foundation of integration is access to your data, wherever it resides. Before you start looking at new and exciting technologies like Web services and what they can do for you, consider the simple but complex task of accessing that data. If you can't get at your data, you have no hope of integration at all. Microsoft's Universal Data Access goal is to provide easy access to data, wherever it resides.

Just getting access to the data, however, does not provide information directly to users or provide them with tools to manipulate that data. Users may take tools like Microsoft Office and try to pull together data from disparate systems into information they can use. This approach usually leaves it to the user to perform the integration of data once they have access to it.

Most applications have a certain amount of logic in them that manipulates data in one or more ways. Without the ability to integrate your applications, you will be stuck duplicating much of this logic. Data and application integration provide you with access to data/information that your workers can use. For instance, you could create a common interface for users that pulls together sales and product history. This approach is similar to the one Microsoft used with Alchemy. The company used XML Web Services to pull together data from several different systems, providing users with one interface to all the systems. The users never know where the data is coming from or going.

To take full advantage of your integrated applications, you should orchestrate them in such a way that they match your business processes. This is the ultimate end of Enterprise Architecture Integration, (EAI) integrating applications and data within the enterprise into automated business processes. The idea of creating integration architecture also allows you to design the integration layers to provide a flexible system that can be adapted to business changes as they occur.

Microsoft provides tools to help you at each of these levels of integration. Many of them are built right into Windows Server 2003, or come as a part of Visual Studio .NET and the .NET Framework. The integration can be achieved easily by using XML Web Services and other technologies such as BizTalk Server, .NET Remoting and Host Integration Server.

Powerful integration tools also provide more direct access to systems, and help to expose those systems as components to Web services and other applications. Windows Server 2003 provides a powerful, robust infrastructure supporting both developers and IT Pros. This built-in functionality can dramatically improve your developer's productivity while reducing maintenance costs.

It's the combination of the .NET Framework and Visual Studio .NET that made the Alchemy project at Microsoft possible. The Sales and Support IT team went into that project with experience in C++, SQL, VB, and Web development, but it was their first experience with programming in the .NET Framework. They were able to rapidly come up to speed using their familiarity with existing Microsoft technologies and create an XML Web Service middle-tier combining data sources and applications. These XML Web services then became the foundation for future applications.

Technologies for Integration

For each level of integration, let's look at some of the Microsoft tools that make Windows Server 2003 such a great platform for integration projects like Alchemy and beyond. The various technologies that are used for integrating applications are part of Windows Server 2003 or are add-on products from Microsoft. The following table provides a short description of some of these products and technologies.

Table 1: Key integration technologies

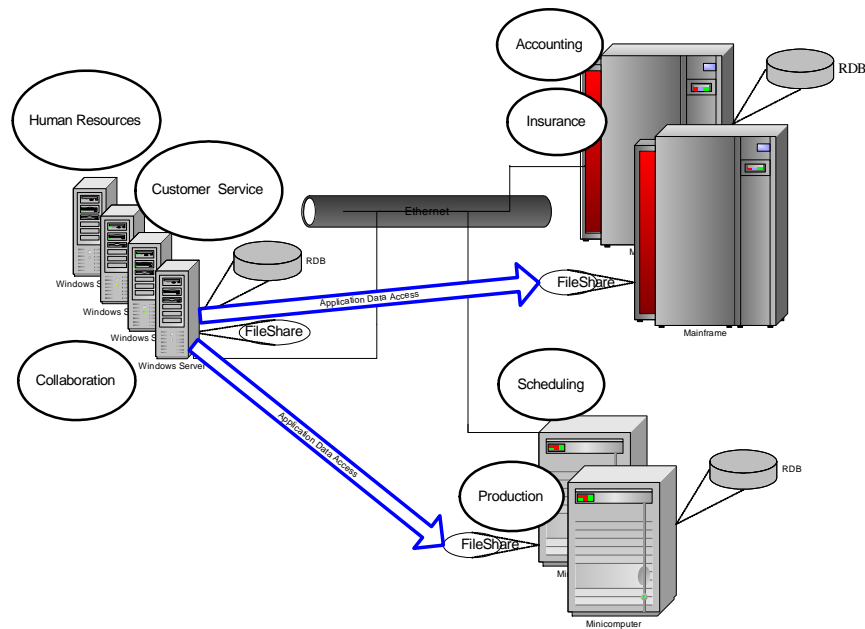
Technology / Product	Description
.NET Framework	The .NET Framework v1.1 ships as part of Windows Server 2003 and is available as a free download for other Windows systems from Windows 98 and forward.
Microsoft Message Queue	Microsoft® Message Queuing (MSMQ) provides an Internet aware transport technology that can be used between .NET applications or other systems such as IBM Mainframes.
Active Directory	The .NET Framework provides a rich set of classes that can be used to access Active Directory (AD). AD can also be used to integrate applications together by providing a common security and data store.
XML	XML is of course a standard that can be used to store, transmit, and manipulate data. XML is at the core of Windows Server 2003 and other technologies such as XML Web Services, MSMQ, SQL Server™, BizTalk Server™, and .NET Framework.
COM+	COM+ 1.5 is part of Windows Server 2003 and provides a powerful middleware transaction system. It also allows you to expose existing COM+ components as Web services to provide for seamless application integration.
Enterprise UDDI Services	Windows Server 2003 includes Enterprise UDDI Services, a dynamic and flexible infrastructure for XML Web services. This standards-based solution enables companies to run their own UDDI directory for intranet or extranet use, making it easy to discover Web services and other programmatic resources and encourage re-use.
BizTalk Server	BizTalk Server is a middleware product that provides a number of services for applications. You can use BizTalk Server integrate applications such as mainframe applications with other applications.
SQL Server	SQL Server is of course Microsoft's database server. SQL Server has undergone a number of changes over the years to make it both a highly scalable, reliable, and flexible database server.
Host Integration Server	Host Integration Server is a communications platform used to integrate applications running on Microsoft platforms with mainframes. Host Integration Server provides a scalable and proven architecture. It also provides COMTI which can be used to easily access CICS or IMS mainframe code.

There are also other companies that also provide software products that assist in integrating applications on the Windows platform. Windows Server 2003 and the other Windows operating systems are open, allowing any vendor to provide products and services.

Data Integration

Integration projects are built upon data integration. Microsoft's tools for data integration build upon their Universal Data Access strategy announced in 1996. Simply stated, Microsoft's Universal Data Access is a platform, application, and tools initiative to define and deliver standards and technologies. UDA builds upon Microsoft's success with Open Database Connectivity (ODBC) and the Common Object Model. COM provided a means for objects to communicate with each other without knowing any details about each other, or even the language in which the object was implemented. COM managed to perform this magic by using standard interfaces. ODBC is Microsoft's widely supported implementation of the Call Level Interface to SQL Databases. ODBC drivers presented a uniform means of accessing data from any SQL database.

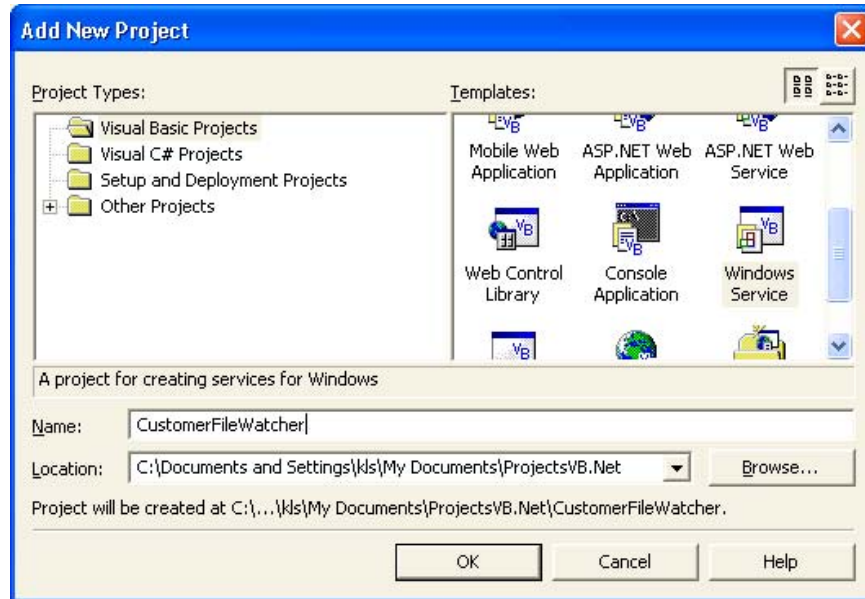
The world does not immediately move forward to embrace each type of new technology. Instead, different ways are needed to move data between systems as integration takes place over time. The next figure demonstrates a typical scenario where files are used to transfer data from one system to another. The network protocols are somewhat immaterial in this situation because any typical file sharing protocol can be used (such as FTP). The important part of this process is that files containing the shared data are placed into particular folders and that other systems have access to those folders to extract the data.



This type of scenario is used in many organizations today. It works but it is usually very fragile, depending upon specific timed operations and flag files to keep things in sync.

Windows Server 2003 provides a solution for this problem by providing features in the operating system that fire notifications when a change occurs (such as when a new file is created in a particular folder). This feature is supported on NT 4, Windows 2000, and Windows Server 2003. Once a file system event fires and is picked up by the FileSystemWatcher, the application can

handle the file by processing it in a number of ways. One easy way to handle this is to create a Windows Service that uses the `FileSystemWatcher` class which is part of the `System.IO` namespace. You can create this Windows Service in Visual Studio .NET by adding a new project and selecting Windows Service as shown in the following figure.



Then you can simply drag a `FileSystemWatcher` component from the Toolbox and drop it on the service, then set the properties to point the `FileSystemWatcher` at a particular file type and folder. Then you wire up the `Create` event to handle the file like this:

```
Private Sub fsWatcher_Created(ByVal sender As Object, _
    ByVal e As System.IO.FileSystemEventArgs) _
    Handles fsWatcher.Created
    Dim ds As New DataSet()

    ds.ReadXml(e.FullPath)

    ' ... your code
End Sub
```

This code fires when a new file arrives or is created in the target directory, then loads the file into a `DataSet`. Then the application can manipulate the `DataSet` in many ways, such as sending the `DataSet` to another system via email or MSMQ or simply loading the data into a database.

For more information, see the following topics

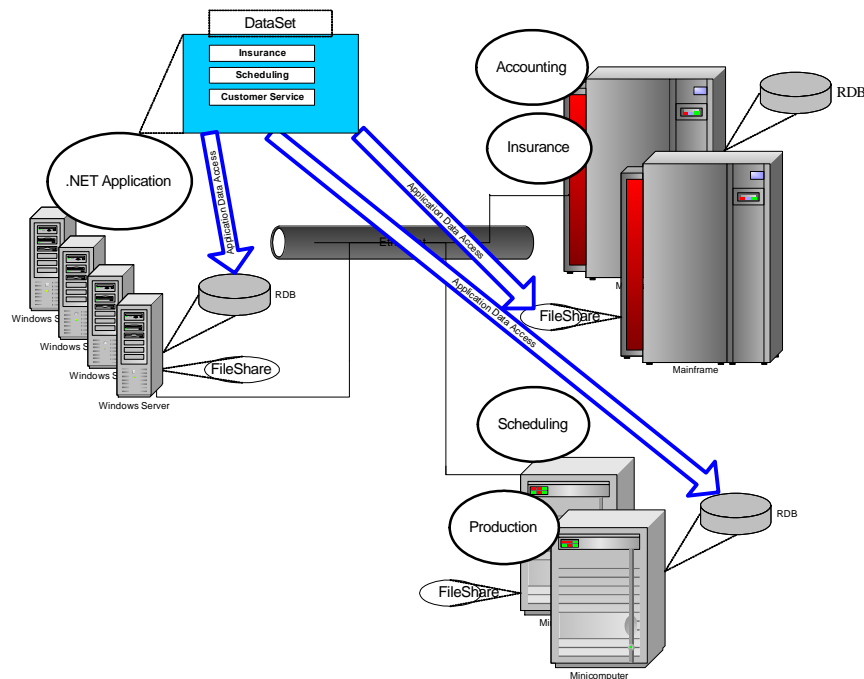
- [Walkthrough: Reacting to File System Events](#)
- [Windows Services: New Base Classes in .NET Make Writing a Windows Service Easy](#)
- [Windows Service Applications](#)

Data Access and .NET

ADO.NET is the successor to ADO. ADO.NET takes full advantage of the .NET Framework, and in keeping with the focus on standards, provides better support for both XML and a disconnected data access model. It's a data-centric model, as opposed to a database-centric model. At the heart of ADO.NET is the DataSet class which takes the place of ADO's Recordset.

The DataSet is an in-memory cache of data retrieved from a data source. The DataSet can contain the results of multiple queries or data sources, each represented as a separate table. This allows a developer to build an application that pulls data from sources such as files (either XML as shown in the last example or comma separated value or any format) and relational databases and link this disparate data together in one high performance application. The DataRelation object allows the developer to create relationships between the different tables. You can also easily update information contained in tables, eventually feeding those changes to an type of database or file.

The next figure demonstrates conceptually how the DataSet works. You can see the DataSet in the top left section of the figure. This DataSet contains three different tables, each the result of a query or process that pulled the data from three different systems. Once the DataSet has been loaded, the .NET Application can use that data locally as there is no connection back to the source of the data. The data can even be persisted to disk or streamed over a queue or other streaming protocol.



As mentioned earlier, the DataSet is an in-memory cache. The DataSet never has a connection directly to a data source such as a database. Coordinating all the details of dealing with a data source is the managed provider. Managed provider classes are provider specific implementations

of a standard interface for dealing with a data source. They establish connections, issue commands against the data source, populate DataSets, and commit changes back to the data source. Microsoft has created the managed provider architecture as an open architecture that any vendor or third party can support with their data sources or applications.

The disconnected nature of DataSets makes it easier to write scalable applications by loosely coupling your applications from their back end data sources. When you populate a DataSet from a database, the connections are open only long enough to perform the database operation. The typical way to load a DataSet from a database is to use a DataAdapter. The DataAdapter communicates with the underlying data source, requesting data to populate the dataset or updating the database. The Data adapter can be extended to perform data validation checks, or add any extra processing you might need. Microsoft provides three managed providers, one for SQL Server, one for Oracle, and the other for OLE DB, providing you access to any data source with an OLE DB provider.

Let's look at an example. For instance to load a DataSet with the results of two different queries, requires only a small amount of code. To start out, you can create a small block of code that builds a DataSet with two tables like this:

```
Dim oDataAdapter As Sql Client. Sql DataAdapter
Dim sSQL As String
```

```
Try
```

```
    sSQL = "Select * from customers"
    oDataAdapter = New Sql Client. Sql DataAdapter(sSQL, Connecti onStri ng)
    oDataAdapter. Fi ll (ds, "Customers")
    oDataAdapter = Nothi ng
```

```
Catch e As Excepti on
```

```
    Throw New Excepti on("An excepti on occurred loadi ng Customers table", e)
```

```
Fi nal ly
```

```
End Try
```

```
Try
```

```
    sSQL = "Select * from orders"
    oDataAdapter = New Sql Client. Sql DataAdapter(sSQL, Connecti onStri ng)
    oDataAdapter. Fi ll (ds, "Orders")
    oDataAdapter = Nothi ng
```

```
Catch e As Excepti on
```

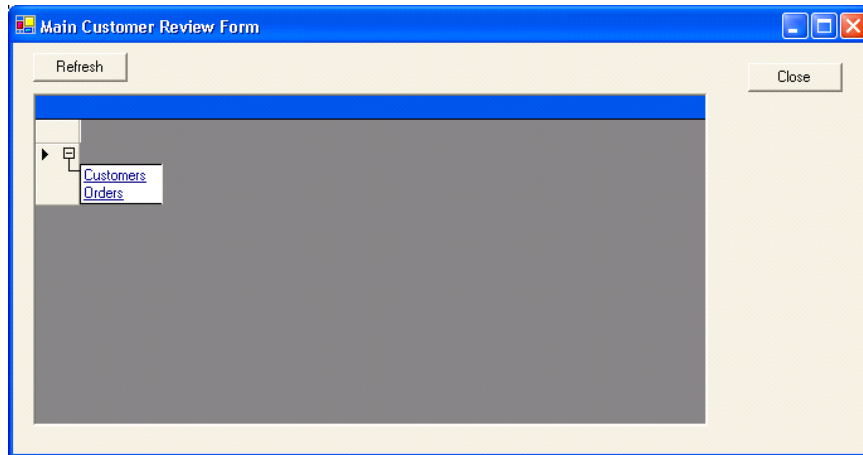
```
    Throw New Excepti on("An excepti on occurred loadi ng Orders table", e)
```

```
Fi nal ly
```

```
End Try
```

```
DataGri d1. DataSource = ds
```

This few lines of code can be used to create a Windows application that presents the data in a UI like this:



The user can work with this data by selecting either Customer or Orders, sort the data as it's displayed, and otherwise manipulate it.

Like the rest of the .NET platform, ADO.NET has strong support for XML. Working with data in a relational model is fast and efficient, but to gain the advantage of XML Web Services, you need to be able to exchange XML data with components and clients. DataSets are automatically serialized into XML by the .NET Framework when returned from a XML Web Service and in other operations. The DataSet also contains methods to import and export XML and XML Schemas for your data. A DataSet's tables, columns, relationships and constraints can all be defined in an XML Schema, a standards-based form for defining the structure of XML data. You can infer XML Schemas from existing datasets. You can also create datasets given an existing XML Schema. Visual Studio comes with an XML Designer tool to make it easier to create and edit XML Schema files interactively for your ADO.NET datasets.

Defining your XML schemas is an important part of building applications that share data. SOAP and TCP/IP standardize the exchange of data, but not the form of the data itself. A data provider and a data consumer still need to agree on the format that data will take. XML yes, but what tags will the XML use, and what do those tags mean in relation to your existing data. XML Designer helps make this task easier.

For more information about ADO.NET, see the following links:

- <http://www.gotdotnet.com/default.aspx>
- www.ASP.NET
- [ADO.NET Support Center](http://www.microsoft.com/ado.net/support)
- [Microsoft ADO.NET Step by Step](http://www.microsoft.com/ado.net/stepbystep)
- [OOP: Building Reusable Components with Microsoft Visual Basic .NET](http://www.microsoft.com/ado.net/oop)

Data on Mainframes

Mainframes have been around since the early years of the computing and they store a tremendous amount of data. While these systems are sometimes called legacy systems, they are no less critical to the operations of many businesses.

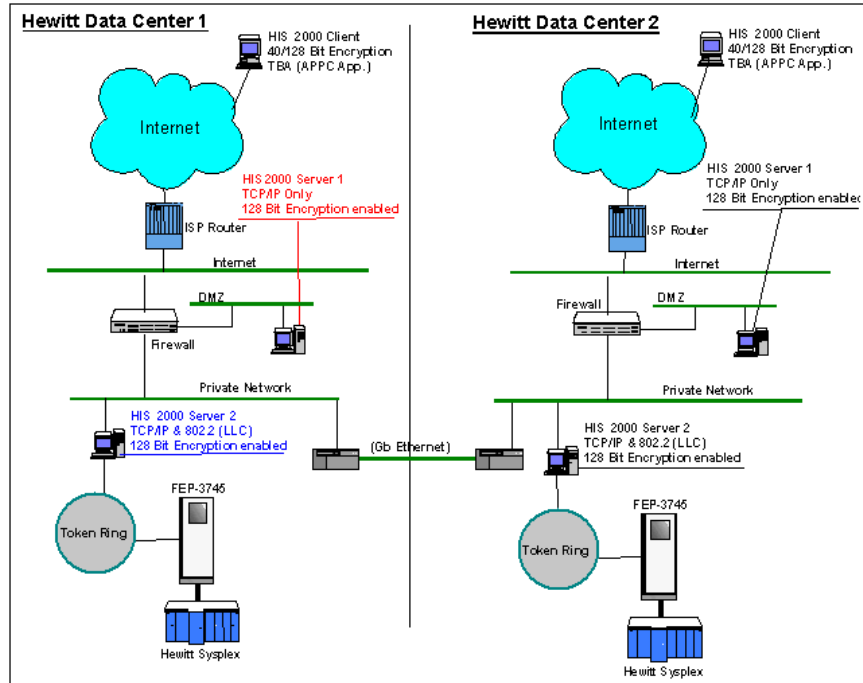
In the late 1990's the Gartner group estimated that 80 percent of business data was still in legacy systems, primarily VSAM. A more recent estimate is 70 percent. There is just too much investment in mainframe applications, billions of lines of code involved in handling your airline tickets, making sure your plane lands safely, managing your sales and customer information, tracking your mortgage payments, and most likely, keeping your business running.

Windows Server 2003 provides you with many tools to easily create, deploy, and manage integration with mainframes. Add on products, Microsoft SQL Server 2000, and Microsoft's Host Integration Server 2000 (HIS) provides you the tools you need to access your mainframe data while receiving all the advantages of Windows' development tools. To assist with your Data integration projects, HIS provides the connectivity you need, as well as the means to get at both relational and non-relational data on the host system.

Connecting to the Host System

The preferred network protocol for Windows Server 2003 is TCP/IP, though older systems in a Windows network may also be using NETBIOS. Windows Server 2003 supports both protocols but many host systems support neither. To connect to your Systems Network Architecture (SNA) systems, HIS provides an SNA Gateway, the successor to SNA Server 4.0. Through the gateway, HIS allows users running Windows, Macintosh, UNIX, the MS-DOS® operating system, and IBM OS/2 to share resources on mainframes and AS/400 systems without requiring system administrators to install SNA protocols on the PCs or install costly software on the host. HIS supports several protocols for communicating between the server using Link Services, provided by Microsoft, IBM, and other third parties and installed on the server. The most popular protocols include Channel attachment, LAN connection via the DLC 802.2 protocol, Synchronous Data Link Control, and X.25. If you are still using coaxial cables, HIS supports DFT link protocol and the Twinax coaxial cable connection method used to connect IBM 5250 terminals directly to an AS/400.

The following figure is from the Hewitt Associates [case study](#). This figure demonstrates a real world view of HIS in action. This figure demonstrates how HIS sits on the network and even allows client access through the Internet.



For example, Turner Broadcasting's Andrew Drooker, Vice President of Research and Development says: "Host Integration Server allows our people all over the world to connect to the AS/400 through a Web interface and to use our existing Windows NT resources, such as printers."

For more information, see the following topics

- [Turner Broadcasting Case Study](#)
- [Host Integration overview](#)

Accessing Relational Mainframe Data

Some of the operational data stored on OS/390, AS/400, and RS/6000 computers can be accessed via the relational database management system (RDBMS) that hosts it. One of these host systems is IBM's DB2. IBM's architecture provides remote access to multiple DB2 systems through Distributed Relational Databases Architecture (DRDA), which provides for simple read-only access to database tables as well as updates spanning multiple DB2 instances using a two-phase commit (2PC) protocol.

HIS provides access to relational data through both an ODBC driver and an OLE DB data provider. ODBC relies on an underlying DRDA application requester (AR) developed by Microsoft. The DRDA AR connects the ODBC driver to DB2 on popular platforms including OS/390, OS/400, RS/6000-AIX, as well as Windows NT, Windows 2000, and Windows Server 2003 systems. OLE DB also sits on top of the DRDA AR, supporting all the same systems, but providing access to developers using ADO and ADO.NET, and making your mainframe data directly available to Microsoft Office applications like Excel.

Accessing Non-Relational Mainframe Data

HIS provides record level I/O access to non-relational data types using OLE DB data providers for AS/400 and VSAM. Host Integration Server (HIS) uses IBM's Distributed Data Management (DDM) to accomplish this. The majority of files on mainframes and AS/400s are not designed to handle SQL, but instead are indexed so COBOL and RPG applications can gain access through a logical file created by the host application. The IBM DDM RLIO protocol allows any DDM-compliant application to access these logical files allowing for fast access to record-level data in all VSAM and all AS/400 files. The OLE DB Provider for AS/400 supports record level access to keyed and non-keyed physical files with external record descriptions, as well as logical files with external record descriptions. Also, the provider can use an optional Host Column Description (HCD) file to describe the format of the target file, mapping the AS/400 data types to OLE DB data types, and allowing the developer to access AS/400 flat data files and source files. The OLE DB Provider for VSAM, which relies on the HCD files to define the metadata of the target dataset or member, provides access to most types of mainframe based VSAM files.

Sometimes you just want to directly access the data files. While HIS supports 3270 terminal emulation, with its ability to transfer files using the IND\$FILE utility, it also offers more efficient methods for transferring files. The *Host File Transfer* utility allows you to transfer files using a single ActiveX control. Host File Transfer can access the same mainframe dataset types as the OLE DB Provider for VSAM or AS/400, but is optimized for download or upload of the entire contents of the dataset or member.

HIS also supports IBM's APPC File Transfer Protocol (AFTP) allowing files to be transferred from SNA systems using commands similar to the TCP/IP based File Transfer Protocol commonly used on UNIX, VMS and other operating systems. Internally, AFTP transfers files using the LU 6.2 program-to-program protocol.

Finally, HIS has an AS/400 *Shared Folders* feature, allowing a Windows Server 2003 administrator to re-share a file on an AS/400 host as if it were a local file system directory. To the client, it appears like any other Windows shared file directory. You don't need any special software on a Windows client to use these files. If you go back to the file sharing scenario earlier in this paper, you can see how your Windows server can host an application that can either talk to the mainframe through COMTI or by using the file share method to access files that are not available in either a database or via COMTI. You can also create a Windows service that monitors a shared folder, periodically pulling files down into the local system.

Host System Integration with SQL Server

Using Distributed Query Processor (DQP), a feature of Microsoft SQL Server, users can access data that resides on multiple, distributed database systems throughout your enterprise, creating queries that join tables in SQL Server with tables in DB2. They can also create SQL Server views over DB2 tables, writing directly to SQL Server and integrating both Windows-based and mainframe data in their application.

To improve decision making, you may want to centralize data stored in a variety of formats across the enterprise. Using SQL Servers Data Transformation Services (DTS) and the OLE DB provider

for DB2, administrators can create a data warehouse integrating virtually any data source accessible via an OLE DB provider. Microsoft provides import and export wizards to help set up your simpler transformation rules. For more complex rules involving multiple sources and destinations, Microsoft provides the DTS Package Designer. The DTS Package Designer exposes all the capabilities of DTS through an easy-to-use, visual interface. Within the Package Designer, users can define precedence relationships, complex queries, flow of control, and access to multiple, heterogeneous sources.

For more information, see the following topics:

- [SQL Server](#)
- [Replication and Heterogeneous Data Sources](#)

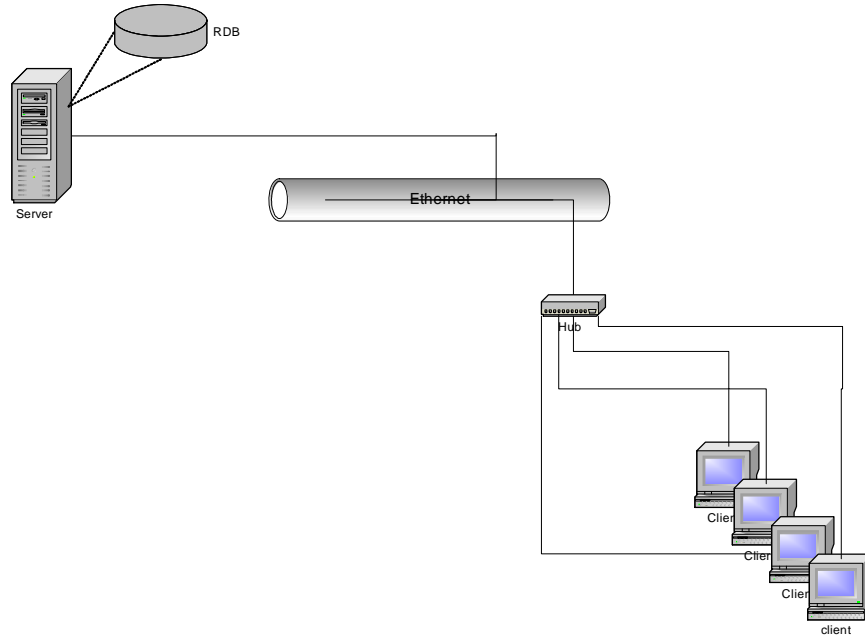
Terminal Emulation

Finally, sometimes data is best accessed through a terminal that allows you to interface directly with the mainframe application. Host Integration Server (HIS) includes support for 3270 and 5250 terminal emulation, as well as the 3270 and 5250 printing protocols. The emulator works with HIS to provide a Single Sign On feature. An additional version of 3270 and 5250 terminal emulation operates directly over TCP/IP, eliminating the need for any part of the SNA stack to be located on the workstations.

HIS and SQL Server combined with the Windows Server 2003's MDAC and ADO.NET provides universal access to your mainframe and AS/400 data.

Application Integration

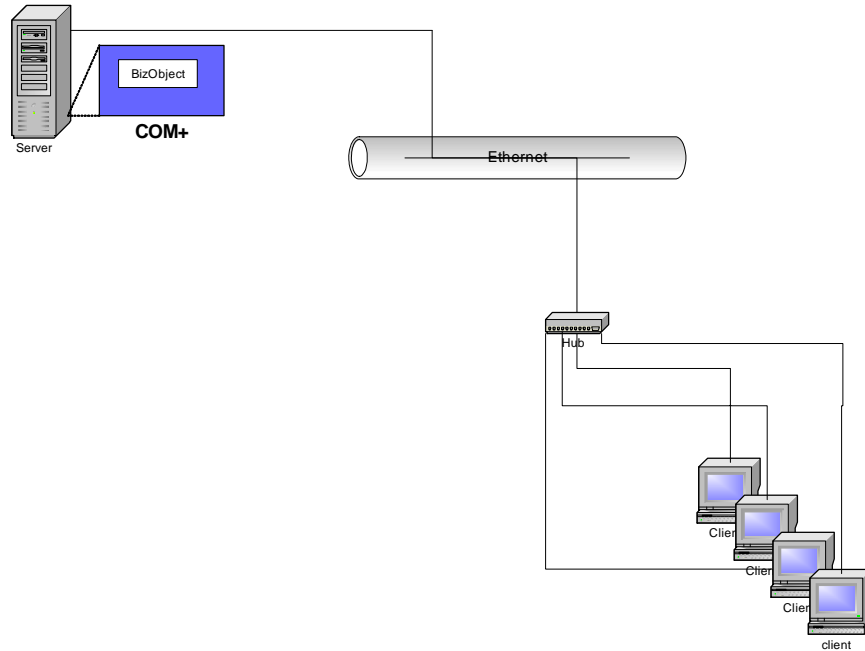
Many integration projects stop at the data access level. For instance, take a client application combining presentation and application logic that connects to a data back end. It works great as a departmental solution, and it's a natural expression of the client server model as shown in the following figure. The server is responsible for hosting the database and usually not any application components. Instead the application runs on the client.



A variation on this model connects to a data gateway or to stored procedure instead of directly connecting to a data store. The gateway can combine multiple data sources. None of these solutions scale well, as a connection from the client to the data source is maintained for each client. Two-tier solutions generally only connect to one data source or one gateway. Their application logic is also difficult to reuse or share. These applications are also difficult to deploy and maintain. If the network becomes unavailable while they are in use, they cease to function.

Multi-tiered applications typically have three or more tiers, and separate the presentation from the application logic from the data back end. Multi-tier applications are generally distributed across a network, with the client running on one system, the application logic on another, and the data server's on possibly another server.

The next figure illustrates this approach where the clients interact with business objects on a remote server. The client applications now require only a small amount of code that interacts with the business object running in COM+ on the server. The business object is responsible for performing business operations and communicating with the database or databases.



Separating the three promotes component reuse. Middle tier components can share a common infrastructure supporting security and transactions. Administrators have more control over middle tier components. They can replicate middle-tier servers and balance the network load between them. They can secure the middle tier, granting or denying access on a component by component basis. You need only one connection to the database for each component rather than one for each user. There are many compelling reasons to use a multi-tier architecture. Once you have created the middle tier, your developers can access it from their applications in many ways from Windows clients to Web applications to XML Web Services. This is why the XML Web Services model is a multi-tiered model, with XML Web Services forming a layer in the middle tier.

The following code demonstrates how to create a simple class in Visual Basic .NET.

```
Imports System.EnterpriseServices
```

```
<Transaction(TransactionOption.Required)> _
```

```
Public Class Account
```

```
    Inherits ServicedComponent
```

```
    <AutoComplete()> _
```

```
    Shared Sub CreateAccount(ByVal AccountName As String)
```

```
        '... your code
```

```
    End Sub
```

```
    <AutoComplete()> _
```

```
    Shared Function RetrieveAccountByName(ByVal AccountName As String) As DataSet
```

```
        '... your code
```

```
    End Function
```

```

<AutoComplete()> _
Shared Function RetrieveAccountByID(ByVal AccountName As String) As DataSet
    '... your code
End Function

<AutoComplete()> _
Shared Function CheckAccountStatus(ByVal AccountName As String) As Boolean
    '... your code
End Function

End Class

```

This class has attributes applied that control how the class interacts with COM+. For instance, the <AutoComplete()> attribute is applied to each procedure indicating that the COM+ should auto complete the transaction unless an error occurs. Attributing the class in this manner makes it easy for the developer to specify the COM+ behavior for the class. Placing class level attributes such as the <Transaction> attribute also lessens the load of the administrator because this type of attribute will set COM+ properties when the component is installed into COM+, relieving the administrator from this task and lessening the chance that an option would be set incorrectly.

COM+ also eases the management of components by providing Component Services Explorer. For instance, we could use this tool to place the previous assembly into a COM+ application. You can perform most of the administrative tasks either using this tool or by automating management tasks using the COM+ administrative interface. Windows Server 2003 includes COM+ 1.5, the next release of component services.

For more information, see the following topics:

- [Building High Performance Applications](#)
- [COM+](#)

Transaction Processing

A transaction is a series of operations performed as a single unit of work. By binding a set of related operations together in a transaction, you ensure the consistency and reliability of the system despite any errors that may occur in processing any single operation. All operations in a transaction must complete successfully in order to make the transaction successful. Windows Server 2003 provides support for transactions through both COM+ and the .NET Framework. Developers familiar with COM, MTS, and COM+ 1.0 will feel immediately at home with COM+ 1.5. COM+ combines the features of a COM object broker and a transaction manager. COM+ supports distributed transactions, role-based security, and includes a built-in thread-pooling scheme.

Integration with Mainframe CICS and IMS Applications

It isn't just data that's in those systems running on mainframes. Billions of lines of code written over the last thirty years or more contain the logic that makes sure businesses run efficiently. Access to mainframe data is useful, but so much more is programmatic access to those applications, batch processes, and reports. Application integration usually begins with transaction

processing to ensure that the applications data is secure and reliable. Host Integration Services provides the tools you need to integrate your Mainframe and AS/400 transactions with your Windows Server 2003, and to make sure actions occur inside of a transaction.

Host Integration Server (HIS) uses COMTI to allow CICS, IMS application programs look like COM+ components. Windows components make method calls and pass parameters to what appears to be just another COM+ component. To use COMTI, a developer transfers the COBOL commarea description to their workstation. Using a COMTI graphical user interface, the developer imports the COBOL data description and converts the COBOL parameter definitions into their COM+ equivalents. The commarea becomes a recordset definition that can be used to pass parameters into and out of the mainframe program. The call is translated to the appropriate CICS program call and sent to the mainframe via HISand LU 6.2. The COMTI proxy component can be packaged with other COM+ components to define the complete transaction. At execution time, calls to COMTI components appear to be calls to the COMTI component.

The State of Georgia Department of Administrative Services (DOAS) used HIS and COMTI to build a rich solution for handling its child-support enforcement system because of the reliability and ease of development with COMTI and HIS. Microsoft provides a [case study](#) that describes this implementation in detail.

For more information, see the following topics

- [COMTI Performance Tuning and Deployment](#)
- [Introduction to Application Integration](#)

Integration With Mainframes Through Message Queuing

COM+ and COMTI provide methods for implementing distributed applications using synchronous transactions. With the trend toward distributed computing, applications cannot always wait for an answer. Instead of synchronous communications, you may need to design your applications to send messages and either complete or continue with some other business while the message continues on to its destination. Loosely coupling your application and data is a basic design principle of distributed applications. Yet distributed applications still need a reliable, scalable means of communicating with each other. The typical way to accomplish this is by using a Message Oriented Middleware System (MOMS). MOMS provides store and forward message queuing so applications are not affected by network fluctuations and do not have to establish connections.

Microsoft's MOMS product is the *Microsoft Message Queuing System*, (MSMQ.) MSMQ is included with Windows Server 2003, and it provides Queuing services to COM+ and .NET applications. You can mark messages passed to MSMQ for delivery as express or recoverable. As the name suggests, express is faster, but the message may be sent along without ever being committed to disk. Marking a message as recoverable makes sure that it does get committed to disk, and if for some reason MSMQ or the system on which it resides were to fail, then your message would not be lost along with the contents of the system's memory.

MSMQ Services are easily accessible by both COM and .NET applications. Within MSMQ you can create both transactional and non-transactional queues, prioritize your queues, and request notification of when a message is delivered. Data privacy and digital signature services are also part of MSMQ. MSMQ can digitally sign and encrypt messages, protecting them from view during transmission even when they are sent over the internet. Since queuing support is native to the Windows servers and clients (such as Windows XP and Windows CE) you can build powerful applications that integrate cleanly with each other, even in a distributed, disconnected environment.

To demonstrate how simple this process is using the .NET Framework, the following Visual Basic .NET code creates a procedure that sends a message via the accounts queue:

```
Sub SendQueueMessage(ByVal QueuedMessage As String)
Dim LocalAccountQueue As New System.Messaging.MessageQueue()
    LocalAccountQueue.QueueName = "queueserver\accounts"
    LocalAccountQueue.Send(QueuedMessage)
End Sub
```

The corresponding C# code would look like this:

```
private void SendQueueMessage(string QueuedMessage)
{
MessageQueue AccountQueue = new System.Messaging.MessageQueue ();
    AccountQueue.QueueName = @"YourMachine\YourQueue";
    AccountQueue.Send (QueuedMessage);
}
```

Visual Studio .NET also exposes queues through the Server Explorer. This makes it very easy for a developer to create an application that uses MSMQ and completely test the application within the environment.

Mainframes and other platforms normally perform messaging using IBM's MQSeries. IBM has made MQSeries available on other platforms, including Windows server systems. But MQSeries is not native to the Windows platform, and not integrated with COM+ and the .NET Framework. Host Integration Server includes an MSMQ-MQSeries Bridge, allowing MSMQ applications to communicate with MQSeries applications. This bridge integrates the two messaging platforms, enabling messages to be transferred in either direction across platforms. The bridge handles transforming the messages from one format to the other. For example, when translating to the MQSeries format, Recoverable messages are marked as Persistent. The client code for a .NET application will still look like the code above. The MSMQ-MQSeries Bridge links MSMQ with MQSeries, making it transparent to the .NET application where the end queue resides.

MSMQ also supports MSMQ Triggers. This allows you to associate executable code with incoming messages, triggering an action when a message arrives. MSMQ Triggers make it easy to setup application support on a queue to process messages. COM+ also provides support for Queued Components. Queued Components allow an application to call methods in another component across a queue.

These technologies make for a very rich middle tier to host your business logic components, components that can perform transactions against multiple databases, including your mainframe

systems, access data anywhere on your system, and expose your existing application logic as XML Web Services.

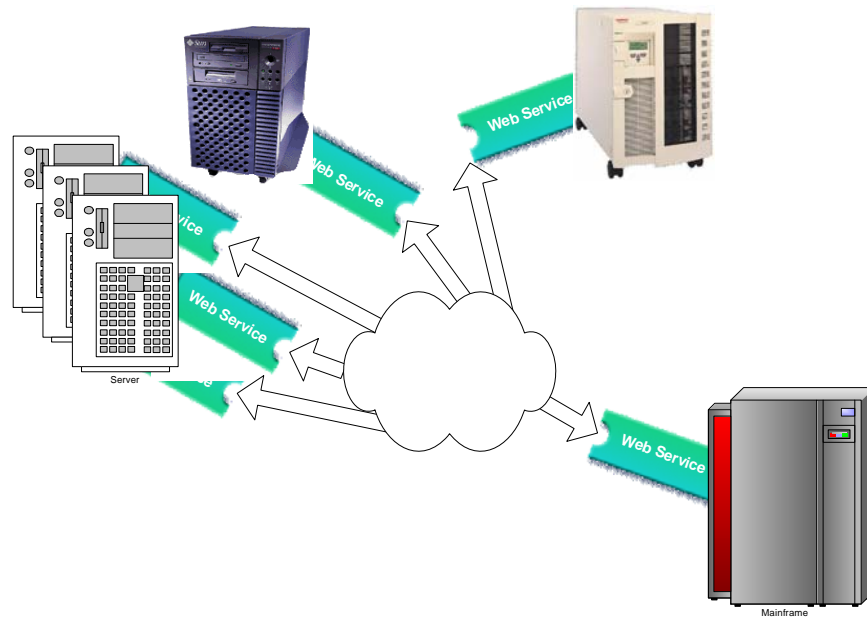
For more information, see the following topics:

- [MSMQ-MQSeries Bridge Performance Results](#)
- [MSMQ Home](#)

Integrating Applications through XML Web Services

XML Web Services are the next logical step in the integration process. XML Web Services allows you to easily expose features in an application to another system. This can be as simple as a Web service that provides a data interface to a system, or as sophisticated as a Web service that provides sophisticated business rules such as insurance claims processing. XML also allows you to set standards for data sent or received through a Web service.

The next figure provides a logical view of this process. The only requirement for this level of integration is an HTTP server on each system that is able to host Web services, and some type of connectivity that supports HTTP between the systems. Then the Web services on each system can interact as if they were components on the same system written in the same language. Most of the major Web servers support Web services, allowing applications on any of these platforms to expose and consume Web services. This is a tremendous advantage to any organization with disparate systems because you do not need to rip and replace software; instead, you can integrate it at a far lower cost. The following figure provides a conceptual view of how the Web services tie the different systems together.



To expose a Web service that pulls data from a SQL Server database and delivers it in XML as shown above is actually quite simple. The following code demonstrates how this is accomplished using Visual Basic .NET.

```
<WebMethod(> Function RetrieveCustomers() As DataSet
```

```

Dim ds As DataSet
Dim oDataAdapter As SqlClient.SqlDataAdapter
Dim sSQL As String
    Const ConnectionString = "server=localhost;uid=sa;pwd=;database=NorthWind"

Try
    sSQL = "Select * from customers"
    oDataAdapter = New SqlClient.SqlDataAdapter(sSQL, ConnectionString)
    oDataAdapter.Fill(ds, "Customers")
    oDataAdapter = Nothing
Catch e As Exception
    Throw New Exception("An exception occurred loading Customers table", e)
Finally
End Try

Return ds
End Function

```

You might wonder how this code returns XML as the Return statement simply returns a DataSet. The DataSet serializes itself when returned in this manner, delivering a stream of XML over HTTP. A partial view of this XML is shown below:

```

<?xml version="1.0" encoding="utf-8" ?>
- <DataSet xmlns="http://32XURI.org/">
  <xs:schema id="NewDataSet" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xs:element name="NewDataSet" msdata:IsDataSet="true">
      <xs:complexType>
        <xs:choice maxOccurs="unbounded">
          <xs:element name="Customers">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="CustomerID" type="xs:string" minOccurs="0" />
                <xs:element name="CompanyName" type="xs:string" minOccurs="0" />
                <xs:element name="ContactName" type="xs:string" minOccurs="0" />
                <xs:element name="ContactTitle" type="xs:string" minOccurs="0" />
                <xs:element name="Address" type="xs:string" minOccurs="0" />
                <xs:element name="City" type="xs:string" minOccurs="0" />
                <xs:element name="Region" type="xs:string" minOccurs="0" />
                <xs:element name="PostalCode" type="xs:string" minOccurs="0" />
                <xs:element name="Country" type="xs:string" minOccurs="0" />
                <xs:element name="Phone" type="xs:string" minOccurs="0" />
                <xs:element name="Fax" type="xs:string" minOccurs="0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>

```

```

</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>
<diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
  xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
<NewDataSet xmlns="">
<Customers diffgr:id="Customers1" msdata:rowOrder="0">
  <CustomerID>ALFKI </CustomerID>
  <CompanyName>Alfreds Futterkiste</CompanyName>
  <ContactName>Maria Anders</ContactName>
  <ContactTitle>Sales Representative</ContactTitle>
  <Address>Obere Str. 57</Address>
  <City>Berlin</City>
  <PostalCode>12209</PostalCode>
  <Country>Germany</Country>
  <Phone>030-0074321</Phone>
  <Fax>030-0076545</Fax>
</Customers>
<Customers diffgr:id="Customers2" msdata:rowOrder="1">
  <CustomerID>ANATR</CustomerID>
  <CompanyName>Ana Trujillo Emparedados y helados</CompanyName>
  <ContactName>Ana Trujillo</ContactName>
  <ContactTitle>Owner</ContactTitle>
  <Address>Avda. de la Constitución 2222</Address>
  <City>México D. F. </City>
  <PostalCode>05021</PostalCode>
  <Country>Mexico</Country>
  <Phone>(5) 555-4729</Phone>
  <Fax>(5) 555-3745</Fax>
</Customers>
</NewDataSet>
</diffgr:diffgram>
</DataSet>

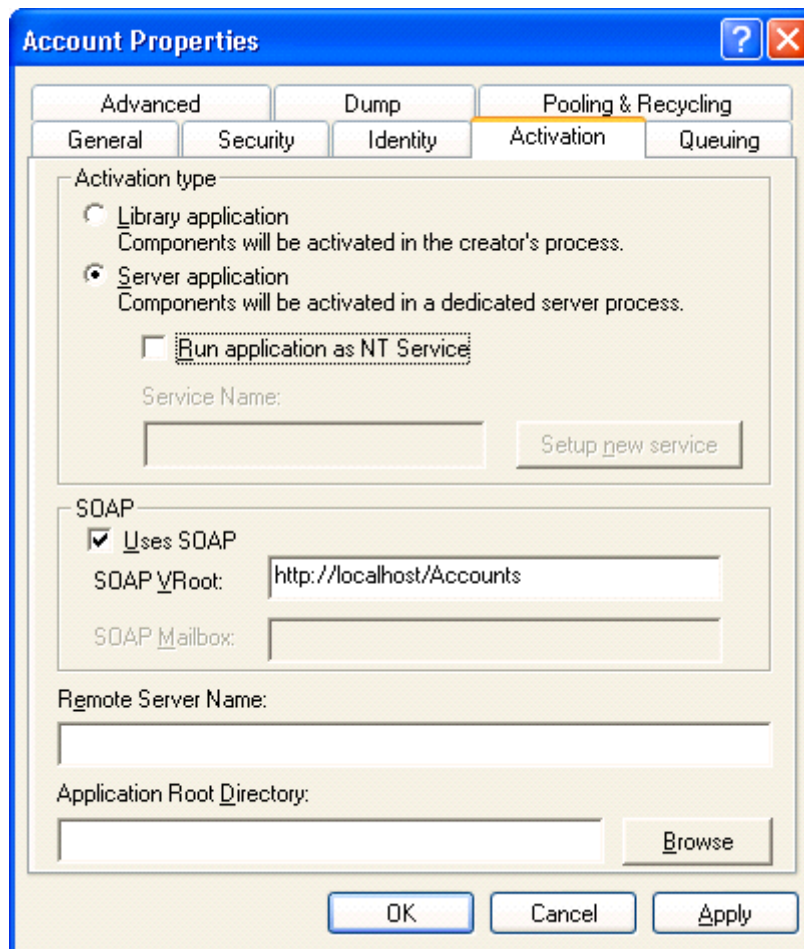
```

This XML only represents the first two rows of data, but it illustrates the small amount of code that is required to make this happen. It also shows that any system that calls the XML Web Service function will get back XML that it can use.

XML Web Services can be used in many ways to integrate applications. On non-Microsoft platforms, you can use technology that is provided by different vendors such as IBM that allow you to host Web services.

If you have systems running on Microsoft platforms such as a Visual Basic 6.0 or a C++ application, you can integrate those applications with .NET applications in several ways. One way is to use the SOAP Toolkit from Microsoft. This toolkit is a free download and allows you to build XML Web Services and add them to those applications. Then your existing application can call XML Web Services on other systems, and other systems such as your .NET application can call XML Web Services that are part of the existing application.

COM+ 1.5 (Windows XP and Windows Server 2003) supports a new feature that allows either a developer or system manager to expose a COM+ class as a XML Web Service. This allows the XML Web Service to be called by any application that can invoke a XML Web Service. To implement this feature, you open the properties for the COM+ application and then check the Uses SOAP checkbox as shown in the next figure. Then enter the path for the virtual directory or Web site that will host the XML Web Service created. The next figure shows the interface that specifies the settings for the new Web service.



Between Visual Studio.NET, COM+, the SOAP Toolkit, and support for standards, XML Web Services are easy to implement.

For more information, see the following topics:

- [XML Web Service Basics](#)

- [XML Web Services](#)
- www.GotDotNet.com
- www.ASP.NET
- [SOAP Toolkit 2.0 SP2](#)
- [SOAP Toolkit 3.0 Beta](#)
- [W3C Web Services](#)
- Building High Performance Applications
- Building in Security for Applications

Coordinating Web Service Development with Enterprise UDDI Services

Windows Server 2003 comes with Enterprise UDDI Services, a dynamic and flexible infrastructure for XML Web services. This standards-based solution enables companies to run their own UDDI directory for intranet or extranet use, making it easy to discover Web services and other programmatic resources. You can use UDDI Services to publish information about your XML Web services internally, share them with your business partners, or expose them publicly. Internally, UDDI Services makes it easier for departments to work together. It promotes re-use and supports dynamic application configuration. The following table shows two core scenarios of UDDI Services.

Table 2: Scenarios for UDDI Services

Core Scenario	Description
Code Re-Use	When building applications, developers search UDDI Services for programmatic resources to re-use, such as a tax calculation service. UDDI Services exposes all of the information needed to invoke a service to make it easy for the developer to include in an application. UDDI Services provides a fast and effective way to discover and re-use Web services and other programmatic resources.
Dynamic Configuration	At runtime, an application queries UDDI Services to discover the current binding information for services and then connects directly to those services. An example of this is a stock broker application that queries UDDI Services first thing in the morning to get configuration information for the different services it consumes such as a stock ticker, customer service applications, settlement services, etc. Using UDDI Services, IT can provide highly available, reliable applications using dynamic, flexible infrastructure for Web Services in Windows Server 2003.

Active Directory provides the authentication and authorization backbone for UDDI Services. All access and permissions to UDDI Services, whether for reading, publishing, or coordination are assigned through a set of roles that can be configured to map to existing groups. Active Directory also provides one of the initial bootstrap mechanisms for finding UDDI Services servers on the network. UDDI Services can optionally be registered as a service publication within Active Directory and a simple AD query will return a list of all UDDI Services instances on the network that can be queried for rich Web service information.

Installing UDDI Services

UDDI Services is itself an XML Web service, accessed via either a Web-based UI or programmatically via SOAP APIs. Use the Add/Remove Windows Components tool to install UDDI Services. During installation, UDDI Services will install Internet Information Services (IIS) configured to use ASP.NET, unless IIS is already installed.

UDDI Services stores its information in a database, and uses the Microsoft Data Engine (MSDE) as the default data store. MSDE is included with Windows Server 2003. For high reliability and availability scenarios, UDDI Services can use Microsoft SQL Server™ 2000.

UDDI Services may be deployed on a single server or across multiple servers. For example, IT administrators could distribute the Web-based user interface and APIs across one or more servers in a typical load-balanced Web farm configuration and run the database on a separate dedicated server running SQL Server 2000. Or IT administrators could run the database on a clustered instance of SQL Server 2000 using Microsoft's clustering technology in active-passive mode—a configuration that provides great scalability and reliability.

If you do not, however, have a database server already installed on the system and you are not using a SQL Server elsewhere in your enterprise, Windows Server 2003 will install MSDE when UDDI Services is installed.

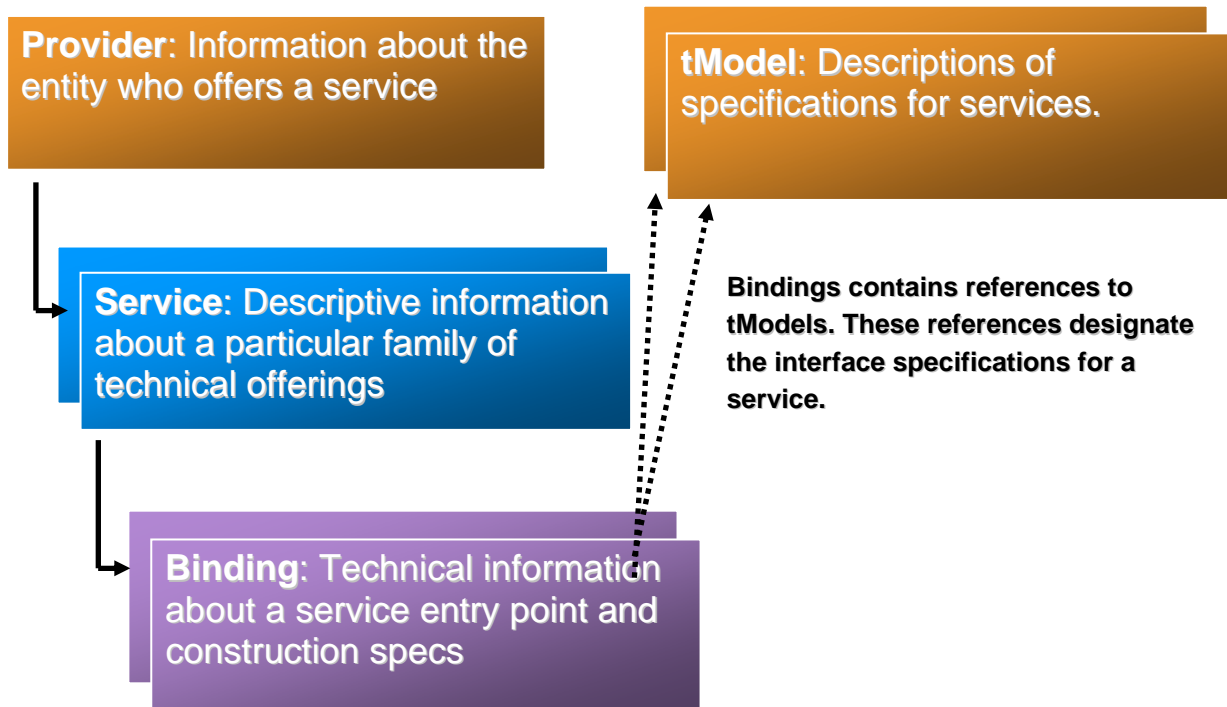
The installation wizard also provides the option of using SSL for publishing and administration connections. Using SSL is *strongly* recommended for publishing and administration as, in some configurations, the UDDI API passes username and password information in the SOAP body.

The UDDI Services configuration console is a Microsoft Management Console (MMC) plug-in. Once UDDI Services is installed, a link is available to the console under Administrative Tools in the Start menu. The console plug-in is also included in an Admin pack for installation on systems other than Windows Server 2003. All instances of UDDI Services servers can be managed from a single console.

The console also allows security settings to be configured. Role based security can be applied to UDDI Service, mapping existing groups to four predefined UDDI roles: Administrators, Coordinators, Publishers, and Users. Administrators get full rights to the service, including most of the console functions just mentioned. Coordinators can manage entity categorization schemes, manage UDDI data and view statistics. Publishers publish provider information, services and tModels, while users can only browse those entries.

UDDI Data

UDDI data is structured in four entities: Provider, Service, Binding, and tModel (technical model) plus categorizations that can be applied across Providers, Services and tModels. For Web Services, tModels are typically synonymous with WSDL files. They define the types used by the Web service as well as the message and operation definitions for the Web service. Services and Bindings are part of a Provider entry. tModels are defined separately. A Binding references a tModel. These four entities relate to one another as shown below.



A developer might also think of a tModel as an interface. Just as multiple classes can implement the same interface, multiple services might implement the same tModel. And just as a class might implement multiple interfaces, an XML Web service might implement multiple tModels. The benefits are essentially the same. If you write an application to use a particular interface, it can use any service that implements that same interface. Keeping the tModel separate from the details of a particular implementation makes it possible to do some interesting run-time scenarios we'll cover later.

A provider or business entry has the information shown in table 3.

Table 3: UDDI Provider/Business entry information

Item	Description
Name	The name of your business or organization. This may also be the name of an application that provides services or a department.
Description	A paragraph or so of information describing the provider. You can create descriptions in multiple languages using the xml:lang namespace.
Contacts	People to contact who can help out with anything needed from the provider.
Identifiers	Pieces of data unique to the provider, such as a company register listing number.
Business classifications	These identify the location of the provider, as well as what the provider does. UDDI Services supports several standard categorization schemes including the North American Industry Classification, Universal Standard Products and Services Codes, and ISO 3166 Geographic Taxonomy. You can browse the current listings to get an idea of how these categorizations work. You can also create and import custom categorization schemes.

Discovery URLs	These provide locations where details about the provider can be found.
Services	Names and descriptions of services you provide, and possibly bindings and references to the tModels (WSDL files) your company exposes.

Adding Records to UDDI Services

UDDI Services is exposed through an XML Web service with a Web interface. Information can be added to UDDI Services through either the Web interface or the API. Use the Web interface is recommended for entries that are relatively simple, or stable. However, if the entry is frequently updated or more complex, the registration process can be scripted using the Microsoft UDDI SDK.

The UDDI SDK provides all of the tools required for a Visual Studio .NET programmer to interact with UDDI Services at design time or run time. The SDK can be used to add Web service registration features to development tools, installation programs, or any other software that needs to locate and bind with remote Web services. It can be obtained the Microsoft UDDI SDK through the Microsoft Developer Network (MSDN). You can use the SDK to find and retrieve information from UDDI, issuing complex API-based queries.

For more information, see [The Microsoft UDDI Software Development Kit](#).

UDDI Services Runtime Advantages

UDDI Services is a great tool for finding services and building applications to use those services. Those are its design time advantages. UDDI Services can also be used at run-time to dynamically discover services that implement a particular interface, or to find the access point for a back-up service when the primary service is unavailable. A service could move, or be placed on a new server. If its location has been hard-coded into an application, the application will break when it moves. A different methodology would be to build the application to query UDDI Services at run-time to obtain current information for the service(s) the application wants to use.

A good way to do this is to include the UDDI Services instance and service registry key in the application's XML configuration file, like so:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="UDDI_URL" value="
      http://internalhost/uddipublic/inquire.asmx" />
    <add key="bindingKey" value="f46fced9-2b8a-4817-b957-f8d8aca0a2f9" />
  </appSettings>
</configuration>
```

This information can then be used to query UDDI depending on different scenarios. A list of services could be gathered that implement a particular interface, and then the application could pick the service closest based on its geographic classification, or poll those services and aggregate their information dynamically by querying UDDI Services at run time. UDDI Services could also be used to publish an interface which business partners would implement in their Web

service. Then an application could periodically query the registry for all services implementing that interface and publish information to those services, perhaps refreshing their catalog data. UDDI Services is a powerful tool for coordinating efforts with business partners and keeping XML Web Service information current.

For more information, see the following topics

- [What's New in Enterprise UDDI Services](#)
- [The Microsoft UDDI Software Development Kit](#)
- [Using UDDI at Run Time](#)
- [Using UDDI at Run Time, Part II](#)

Process Integration

We started out by saying that there were three layers of enterprise integration. To achieve the final layer of integration, you can match Web services integrating your data and applications to your business processes. Here we move beyond merely hooking applications together to orchestrating and automating the processes that keep your business running every day. With middle tier Web services in place, it's possible to roll custom solutions for your business, modeling the workflow of a department, for example. The Alchemy project is a good example of how you can do this with a bit of business logic tying the Web services together.

In a large business, however, you run into the same problem that always thwarted process integration with remote procedure calls. The process logic can quickly become unmanageable, particularly when business rules change. Just as application integration required a shared infrastructure for the cooperation of middle-tier components, Process Integration requires an infrastructure for defining, managing, and maintaining business process logic. BizTalk Server 2002 provides that infrastructure by providing a layer of services that exist above the core Web services technology provided by Windows Server 2003. BizTalk Server makes it fundamentally easier to integrate applications both within and between organizations. The two key services provided by BizTalk Server are Orchestration Services and Messaging Services.

BizTalk Server can be described in many ways; it's an integration development environment; it's a server with two major subsystems, messaging and orchestration services. It's also an integration broker, and it's the engine and tools that you use to connect a lot of stuff together. However you describe it, BizTalk derives much of its power from the underlying Windows Server 2003 which has all these data and application integration tools baked right in. BizTalk Server is how you orchestrate those services.

BizTalk Orchestration Services

As explained in the Integration Overview paper, Microsoft uses the Web Service Description Language (WSDL) to provide all the information a client might need to know about a Web Services to use it properly. BizTalk uses another XML specification called XLANG to glue Web services and applications together in meaningful ways. XLANG schedules define the behavior of Web services that participate in a business processes.

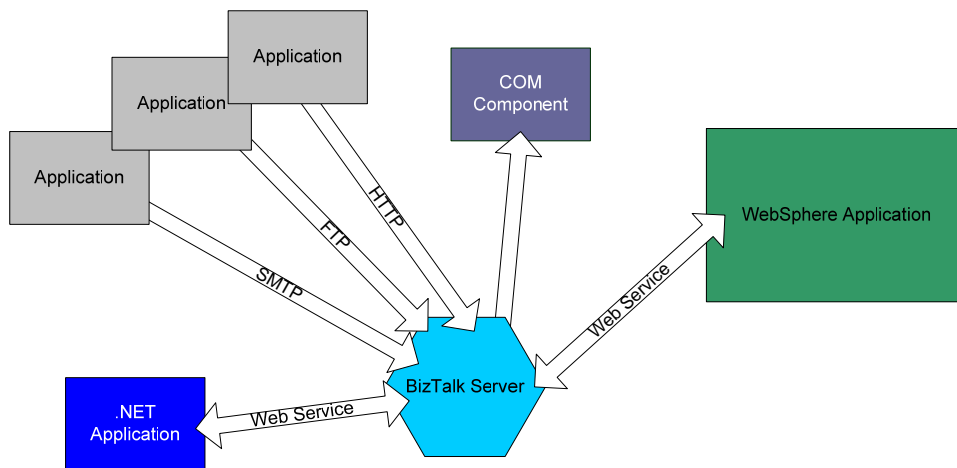
You compile XLANG schedules from drawings made in the BizTalk Orchestration Designer. An analyst can use the graphical interface to chart out business processes. From there, a developer can take over with the same drawing, putting in the instructions on how to fulfill each step in the process. Steps in the process might be receiving an order from a customer, validating their credit, handing it off to the order processing system, handing it off to the warehousing system so someone can pick, pack, and ship it. After they pick, pack, and ship it, they tell the order processing system they shipped it, they to tell the accounting system they can bill for it, etc. All business processes can be distilled into multiple messages going back and forth between cooperating entities. Those entities might be Web services, they might be other applications, and they could even be people. Messages go back and forth. This one is an XML Web Service interface, that one is a file, that one is an API.

Compiled, the XLANG schedule scripts the progress of a process. The processes can be grouped in a transaction that either succeeds or fails and rolls back any changes made along the way. The BizTalk orchestration server tracks the progress of these processes, recording the state of the process at each point along the way. Where database transactions take place over a few moments, business transactions may take days, weeks, or even months to complete. A key benefit of BizTalk Orchestration Services is robust, long-running, loosely-coupled business processes that span organizations, platforms, and applications.

BizTalk Orchestration provides the tools to define, implement, and manage business processes. BizTalk Orchestration is designed to manage business processes. BizTalk Messaging Services are designed to support the receipt of messages that flow into a business process, or to send messages that flow out of a business process. Together they coordinate events and consequences of those events. An event is a service request, and consequences are the set of other events that service request triggers.

BizTalk Messaging Services

The BizTalk server can communicate with a variety of different systems using common standards and protocols. You can also create custom components to adapt BizTalk Server to other systems. Essentially the messaging system receives messages, queues them for delivery, and then delivers them to another system. It can receive messages in a number of ways: via HTTP, SMTP, FTP, a dropped File, from MSMQ, MQSeries, COM, or DCOM. The system is extremely flexible. It can also map one form of document to another. It may receive an XML document in any of the above standards based way, but then transform it to a flat file, or a different XML Schema, whatever the recipient needs. The following figure demonstrates how BizTalk Server can be used to integrate different applications. In this case BizTalk Server is the hub that ties together the WebSphere applications with .NET and other applications.



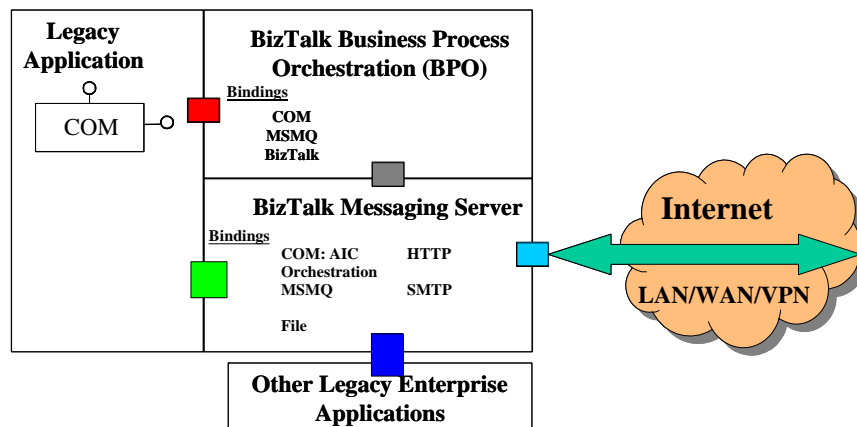
The BizTalk Mapper provides a visual tool for manipulating the Mapping and transformation services offered by the Messaging engine. You map the transformation between two different data formats by dragging the source to the destination field. Any transformation of the data can be expressed as a functoid, a built-in reusable function for data transformation.

If you need to create, edit, or manage a data structure for a message, you can use the BizTalk Editor, another graphical tool included with BizTalk Server. Using a tree-structure metaphor, you can use the same to create virtually any structure.

Putting it Together

From the development side, you can create components or scripts that can be wired together with the Orchestration features. You are not limited to a set of features that BizTalk provides. You can also use one of the over 300 Application Adapters that are provided by Microsoft and others or you can create your own adapters.

The following diagram is from the BizTalk Adapter Developers Guide and illustrates how the various BizTalk components work together.



The above figure illustrates the way in which the Orchestration and messaging aspects of BizTalk Server combine to unify the business-to-business external communications with legacy Enterprise Application Integration (EAI). The messaging portion of BizTalk Server primarily facilitates external (business-to-business) connectivity. The external connectivity can also be between your own systems inside the firewall or between different divisions. Applications are integrated by wrapping and exposing the application's functionality in a BizTalk Orchestration or by binding the application directly to BizTalk Messaging via an Application Integration Component (adapter).

If you are serious about process integration, then BizTalk is certainly the tool you are looking for. It turns huge, time consuming, costly integration projects into manageable, fast, and inexpensive projects. Many more features, including Security, and tools to handle rendering Web pages to mobile devices, are covered in other articles in this series.

Marks and Spencer used BizTalk Server to integrate a sophisticated point of sale system with their own systems and with their customers. See the Marks & Spencer [case study](#).

For more information, see the following topics:

- [BizTalk Developer site](#)

- [Introducing BizTalk Server and Visual Studio .NET](#)
- [Microsoft BizTalk Server 2002 Toolkit for Microsoft .NET](#)
- [BizTalk Server Adapter Developers Guide](#)
- [Microsoft BizTalk® Accelerator for HIPAA](#)
- [Coca Cola AG Case Study](#)

Conclusion

As you can see from this white paper, there are many ways that you can use Microsoft technologies to integrate various applications. Windows Server 2003 provides a reliable and scalable platform to facilitate this, providing many of the services required to develop and host the applications. It also provides an open platform that other vendors can develop software for.

Using industry standards like XML and XML Web Services, your investment in Windows Server 2003 will pay off in the future. As these standards evolve, Microsoft will continue to enhance Windows Server 2003 and other products and technologies to support those standards. Integration today will ensure that your systems can evolve over time as your needs change.