CHAPTER 1

BizTalk Initiative

IN THIS CHAPTER:
The BizTalk Framework
BizTalk.org
Microsoft BizTalk Server 2000
Chapter in Review
To better understand BizTalk and the motivation behind the BizTalk initiative, it is worth mentioning the challenges system integrators have faced over the decades in their efforts to build Enterprise Application Integration (or EAI) and business-to-business (or B2B) electronic commerce (or e-commerce) solutions.

Typical enterprises use different software applications and platforms to support different areas of their business. Some of these applications may be built in-house, whereas others may be purchased from third party software vendors. For example, in an auto manufacturing company, the accounting department may use a Java application running against an Oracle database on a Sun Solaris UNIX system. The purchasing department may take advantage of a Visual Basic application with an SQL Server database backend running on a Microsoft Windows NT Server. Yet the human resources department may utilize a legacy system built in COBOL and running on a DB/2 database on an IBM AS400 mainframe system. The list goes on and on…. In order to conduct their business more effectively and stay competitive, organizations need to integrate these different applications and systems together as if they were one big application. As mentioned above, however, these applications and platforms are built using different programming tools, which run on different platforms, storing and processing data in different formats. Worse yet, these applications and platforms were never meant to work together—all of which has created many unique challenges to the EAI. Proprietary data storage formats and the application programming interfaces (or API) for accessing and manipulating these data make it extremely difficult for applications to exchange data efficiently and in a controllable manner.

The fast growth of the Internet and the World Wide Web has brought many business opportunities to organizations, allowing them to conduct business over the Web with their trading partners. At the same time, this has also brought organizations many new challenges in addition to issues regarding integration efforts within the organizational boundary. In B2B integration scenarios particularly, organizations also need to tackle such unique concerns as security, privacy, and repudiation.

The first step in addressing these issues is to implement standardized ways of describing common vocabularies used to illustrate and construct the data as well as the content models of the data itself. The data we are talking about here is often referred to as a *Business Document*—for example, a purchase order or an invoice. In order to alleviate data exchange between applications, some standards and specifications have been proposed, published, and adopted by standard bodies, organizations, and independent system integrators. These include earlier standards...
such as Electronic Data Interchange (or EDI) and emerging standards such as eXtensible Markup Language (or XML). Although EDI provided many benefits in helping streamline basic transaction flows between business trading partners, the complexity and high cost of implementation have restricted its use only to those large companies and organizations who can afford using it. On the other hand, XML, powered with its accompanying languages such as XSLT (eXtensible Stylesheet Language Transformation) and XML Schemas, has proved to be a more flexible and efficient language for describing the contents and structure of business documents. As a result, XML has been adopted by the industry at an astonishing speed.

**NOTE**

Appendixes A to C of this book contain comprehensive coverage of XML and its related technologies, including XML, XSLT, and XML Schema.

Using a standard language such as XML is only the first step in resolving issues involving EAI and B2B integration. The process of building these integration solutions is complicated and challenging. Which is exactly what BizTalk initiative wanted to address.

BizTalk initiative consists of three key components:

- **The BizTalk.org** Web site. This is a community started by Microsoft and supported by a wide range of organizations including SAP, CommerceOne, Ariba, and others. It provides a globally available repository and library for organizations and system integrators to publish and share schemas (vocabularies and structure) for constructing BizTalk documents.

- **The BizTalk Framework** is a specification for designing and developing XML messaging solutions. Built on top of many open Internet standards and protocols such as XML, SOAP (Simple Object Access Protocol), HTTP, and SMTP, this framework provides guidelines for constructing and passing BizTalk documents between applications and business partners.

  **NOTE**

  For information about the SOAP Protocol, visit MSDN SOAP Developer Resources at: [http://msdn.microsoft.com/soap/default.asp](http://msdn.microsoft.com/soap/default.asp)

- **Microsoft BizTalk Server 2000** is the industry’s first comprehensive tool that unites EAI and B2B integration with its unique BizTalk Orchestration technology, allowing organizations to easily build dynamic business processing between applications both within the organization and across business partnerships.
In this chapter, we discuss the BizTalk Framework 2.0, introduce BizTalk.org and BizTalk Server 2000. Chapter 2 will then explore BizTalk Server 2000 in more detail. The rest of the book will cover related technologies, tools, and how to use them to build BizTalk solutions, focusing on the use of BizTalk Server 2000.

The BizTalk Framework

The rapid adoption of XML and XML-related solutions by the EAI and B2B integration industries provided a standard way for specifying the vocabularies and content models of business data. These efforts, however, only solved half of the problems. Organizations and business trading partners still have to face a broad range of interoperability issues. They need a platform- and technology-neutral specification for the design and development of XML-based messaging solutions for communication between organizations and business trading partners. This is exactly where the BizTalk Framework steps in. The BizTalk Framework 2.0 is a specification that provides a general overview of the BizTalk conceptual architecture, centering on two key concepts: BizTalk Documents and BizTalk Messages. BizTalk Framework provides detailed specifications for constructing BizTalk Documents, BizTalk Messages, and their secure transport over a number of Internet-standard and transfer protocols. It also provides guidelines for reliable delivery of BizTalk Documents, and instructions for handling BizTalk Document attachments. The following sections will introduce the BizTalk architecture, explaining BizTalk Documents, BizTalk Messages, and other terminologies. We will also discuss the structure of a BizTalk document, including its header, body, and attachments. We will look at how to reliably deliver (exchange) a BizTalk document, how to secure BizTalk documents, as well as how to bind different transports in BizTalk documents to ensure authentication, integrity, non-repudiation, and privacy. Finally, we will introduce the BizTalk Framework 2.0 Developer’s Toolkit, a technical design framework and programming interface for managing arbitrarily complex BizTalk Framework documents.

The BizTalk Architecture

The implementation model of the BizTalk Framework is composed of three distinct logical components or layers:

- The BizTalk Framework Compliant Server (or BFC Server) is implemented as a set of services providing the XML messaging functions specified in the BizTalk Framework 2.0. Microsoft BizTalk Server 2000 is an example of a BFC server.
NOTE

BizTalk Framework 2.0 introduced several new features over BizTalk Framework 1.0, including adoption of the SOAP 1.1 specification as the format of BizTalk Messages.

- **Application** that is a line-of-business system which stores and processes the business data and logic. Appropriate adaptors that are responsible for facilitating, emitting, and consuming Business Documents and communicating with a BFC server are also considered part of the application.

- **Transport.** This is the mechanism by which the actual interchange of BizTalk Messages between BFC servers takes place, ensuring messages are physically carried from the source business entity to the destination business entity. Examples of transports include HTTP, SMTP, and message-oriented middleware (or MOM) such as MSMQ, MQ Series, or JMS (Java Messaging Services). The BizTalk Framework is independent of transport protocols. The choice of a specific transport protocol is completely up to the solution implementer.

Figure 1-1 illustrates a typical example of how the BizTalk Framework may be implemented.

As shown in Figure 1-1, Application A constructs a BizTalk document, packs it into a BizTalk message, and sends it to BFC Server A to be processed. BFC Server A transmits the BizTalk message through appropriate transports and protocols to BFC Server B which may process portions of the document before passing it on to Application B.
The BizTalk document is a SOAP 1.1 message, also called a SOAP Envelope, which is essentially a well-formed XML document. A BizTalk document contains a header section and a body section. The header section includes instructions for document routing, document identification, delivery service requesting, attachment cataloging, document securing, and transport binding. The header section of a BizTalk document uses a special set of XML tags associated with the namespaces defined in the BizTalk Framework. These special tags are called BizTags in the BizTalk Framework specification. We will discuss these header instructions in detail later in this chapter. The body section of the BizTalk document contains one or more business documents—well-formed XML documents containing business data. These could be purchase orders, invoices, payments, or any other type of business information.

NOTE

XML uses elements as the basic building blocks for describing its data. BizTags are actually XML elements and are case sensitive. (XML elements and namespaces have been explained in greater detail in Appendix A.)

In Figure 1-1, Application A generates a BizTalk document which may contain one or more business documents (possibly with some attachments as well), and transmits these business documents (under the umbrella of the BizTalk document) to Application B by submitting it to BFC Server A. Either the application or the BFC server could be responsible for constructing the BizTalk document, along with the business documents, depending, of course, on the implementation of the BFC server. BFC Server A then processes the document and any attachments and constructs a BizTalk message which is appropriate for the underlying transport protocol. A BizTalk message is the unit of wire-level interchange between BFC servers. The BizTalk Framework compliant source or destination of a BizTalk message is also called an Endpoint (not shown in Figure 1-1). The BFC server (A in this case) uses information contained in the header section of the BizTalk document (defined by a set of BizTags) to determine the correct transport-specific destination address. The server then hands the BizTalk message to the transport layer which in turn transmits the BizTalk message to BFC Server B through the appropriate transport (Transport B in this case). The BizTalk Framework does not specify which interfaces should be used between the business applications, the BFC servers, and the transport layer. These interfaces are completely implementation-specific.
The Overall Structure of a BizTalk Document

Figure 1-2 depicts the structure of a BizTalk message which contains a BizTalk document (an SOAP envelope). The BizTalk document or SOAP envelope has two sections—the header section contains BFC server processing instructions (routine and content catalog, or manifest) described in BizTags, whereas the body section contains one or more business documents. Optionally, a BizTalk message can also have attachments and a Multipart Internet Mail Extensions (MIME) header.

The Body of a BizTalk Document

The body section of a BizTalk document (designated by a SOAP envelope Body element) holds a business document (a well-formed XML document that contains business data, such as a purchase order) or multiple business documents—for example, a purchase order document along with an associated shipping document.
In the latter case, the BizTalk Framework uses the SOAP encoding rule to encode data targeted by multiple references. It achieves this by using the XML ID attributes and relative URIs. In the case of multiple business documents, the BizTalk Framework also provides a way to uniquely distinguish business documents from other direct child elements of the Body element. It uses the SOAP-ENC:root attribute with a value of 1 to signal that the immediate child of the Body element is actually a business document.

It is also worth noting that in addition to business documents, the Body element of the BizTalk message can also contain another BizTalk message. In this case, however, the BizTalk message will be treated by the BFC server like any other business document, ignoring its header section (as you will see in the following paragraphs).

The Header of a BizTalk Document

The header section of a BizTalk document (designated by a SOAP envelope Header element) provides processing instructions for BFC servers using appropriate BizTags, including mandatory BizTags for describing routing (the endpoints BizTag), document identification, and properties (the properties BizTag), as well as optional BizTags for delivery services (the services BizTag), document cataloging (the manifest BizTag), and process management (the process BizTag).

Processing BizTalk Documents

In this section, we will take a close look at the processing instructions specified in the header section of BizTalk documents.

Routing

The endpoints BizTag under the header section of a BizTalk document contains from and to BizTags, which in turn contain two address BizTags, respectively. These BizTags provide source and destination information specified as names of business entities. The names used to represent business entities are usually business-related abstractions and are not necessarily reflections of physical transport endpoints. For example, “Purchase Orders” and “Purchase Approver” are preferable names for source and destination to “http://POServer/POSubmit.asp”, and “http://POServer/POProcess.asp.” This abstraction makes it possible to use multiple transports and/or endpoints and replace them over time without changing the names of the business entities defined in the BizTalk document. The exact routing and delivering logic of the BizTalk document is completely up to the BFC server that processes the BizTalk document.
The use of the *from* and *to* BizTags as well as the *address* subtags is mandatory. In addition, the BizTalk Framework also provides a mechanism that forces the BFC server to understand and process the *endpoints* header entries by means of a SOAP-ENV:*mustUnderstand* attribute with a value of 1.

**Document Properties**

Under the *properties* BizTag of the BizTalk document header section, there are four mandatory sub-tags: *identity*, *sentAt*, *expiresAt* and *topic*, as summarized in the following table:

<table>
<thead>
<tr>
<th>BizTag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>identity</td>
<td>A URL reference that uniquely identifies the BizTalk document. To guarantee universal uniqueness, you can use the Universally Unique Identifiers (UUIDs) or some cryptographic hash algorithms such as MD5 to encrypt the business document.</td>
</tr>
<tr>
<td>sentAt</td>
<td>The timestamp that indicates when the document was sent, or when the properties element was created in relation to the transmission-retry behavior (something discussed later in this chapter). The sentAt BizTag uses a combination of date and time of day values as defined in the ISO 8601 (called <em>timeInstance</em> data type in BizTalk Framework).</td>
</tr>
<tr>
<td>expiresAt</td>
<td>The expiration timestamps of the BizTalk document. Beyond the point specified by this BizTag, the document should be considered expired and should not be processed or acknowledged by the destination business entity even if it is successfully delivered. In synchronous scenarios, you should leave some room to accommodate the error in time due to the processing of the document. The expireAt BizTag uses the <em>timeInstance</em> data type.</td>
</tr>
<tr>
<td>topic</td>
<td>A URL that uniquely identifies the overall purpose of the BizTalk document. The topic BizTag can be used to verify that the content of a BizTalk document is consistent with its intent. The topic can be either specified by the sending application or be inferred by the BFC server by checking the namespace URI of the first business document.</td>
</tr>
</tbody>
</table>

**NOTE**

The ISO 8601 is an international standard for representing date and times in a standardized way so data will not be misinterpreted when transferred across national boundaries.

As with the routing BizTags, the properties BizTag also uses the SOAP-ENV:*mustUnderstand* attribute with a value of “1” to mandate the recipient BFC server to understand and process its contents.
Reliably Delivering BizTalk Documents

Delivery instructions of a BizTalk document are specified under the optional services BizTag. Under this services tag, there are two more child tags, deliveryReceiptRequest and commitmentReceiptRequest; both are optional. The deliveryReceiptRequest tag requests a delivery receipt from the destination business entity as a confirmation of a reliable delivery of the BizTalk document. The commitmentReceiptRequest tag asks for a commitment receipt from the receiving end so that the sender will know about the processing commitment of the destination business entity. Appropriate receipts (delivery and/or commitment) will be sent to the address specified under their relative sendTo subtags with a timestamp specified by the sendBy tag. The addresses are typically the source business entity.

When the services BizTag is present, the same SOAP-ENV:mustUnderstand = “1” attribute is used for the recipient BFC server to understand and process its contents (i.e., to send appropriate receipts upon request).

Both delivery receipt and commitment receipt are standard BizTalk messages, or SOAP envelopes. Even though the body of a delivery receipt is always empty, the empty Body BizTag is still required for the document to be qualified as a BizTalk message. The body of a commitment receipt on the other hand can either be empty or contain one or more business documents.

The content of a delivery receipt is specified by the following BizTags under the deliveryReceive BizTag of the header section:

<table>
<thead>
<tr>
<th>BizTag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>identity</td>
<td>A URL that uniquely identifies the original BizTalk document sent by the BFC server at the source business entity.</td>
</tr>
<tr>
<td>receivedAt</td>
<td>The receiving timestamp for the BizTalk document acknowledged by the delivery receipt.</td>
</tr>
</tbody>
</table>
The commitment receipt uses the `commitmentReceipt` BizTag under the header section with the following subtags:

<table>
<thead>
<tr>
<th>BizTag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>identity</td>
<td>Similar to the identity tag used for the delivery receipt.</td>
</tr>
<tr>
<td>decidedAt</td>
<td>The processing decision timestamp for the BizTalk document acknowledged by the commitment receipt.</td>
</tr>
<tr>
<td>decision</td>
<td>The actual decision, either positive or negative.</td>
</tr>
<tr>
<td>commitmentCode</td>
<td>Specifies a more specific status of the processing decision, analog to a fault code (optional).</td>
</tr>
<tr>
<td>commitmentDetail</td>
<td>Detailed description about the process decision (optional).</td>
</tr>
</tbody>
</table>

Both delivery receipt and commitment receipt requires that the recipient BFC server understand its contents by using the same mechanism described earlier.

Certain required behaviors have to be met for the BFC servers at both the source and destination business entities in order to ensure the reliable delivery of the business documents.

The BFC server on the sending side (the source business entity) has to do the following:

- Add a delivery receipt request at the header section of the BizTalk document via the `deliveryReceiptRequest` BizTag.
- Before submitting the BizTalk document, persist the document in a durable storage, such as a relation database.
- Establish a retry mechanism by specifying two parameters: a retry interval (either fixed or configurable) and a maximum retry count. The sending BFC server should keep retrying (resubmitting the BizTalk document) until a requested delivery receipt is received or the deadline specified by the `sendAt` BizTag expires or the maximum retry count is exceeded. The content of the BizTalk document being transmitted should not be altered in any way during retries (this includes changing the header). Specifically, even the timestamp specified by the `sentAt` BizTag should remain the same as it was when the document was first transmitted. The transports used for different retry transmissions do not have to be the same, though.
- At the end of the delivery process of a BizTalk document, if the sending BFC server has not received the expected delivery receipt from the receiving BFC server, then the delivery of the BizTalk document is considered failed. When this occurs, the sending BizTalk server needs to notify the source application appropriately.
The receiving BFC server needs to do a little bit more to ensure a reliable delivery of the BizTalk document:

- Upon receiving the BizTalk document, the receiving BFC server needs to persist the accepted document to a permanent storage.
- For each accepted BizTalk document, including those that are copies or duplicates of previously received documents (recognized by the identity), transmit a receipt back to the source BFC server at the address specified by the sendTo BizTag. The receipt is constructed using the structure described earlier in this section. If the document is received past the time instance specified by the expiresAt BizTag, no receipt will be sent. If the document is received before the time instance specified by the expiresAt BizTag but after that cited in the sendBy BizTag, a delivery receipt still needs to be sent.
- Additionally, the destination BFC server may perform an idempotent delivery to appropriate applications to make sure that the BizTalk document is delivered exactly once to its intended recipient application, despite the fact that the document may be received multiple times because of retries at the sending side or errors in transport behavior. Idempotent delivery or Idempotence refers to the ability of a BizTalk document to be transmitted and accepted more than once, with the same effect as being transmitted and accepted only once. Idempotence can be achieved by persisting (archiving) all the BizTalk documents (except for those that are expired) accepted to a durable store (such as a relational database). Usually archiving the identity of the BizTalk document is sufficient, provided that the identity is universally unique (for instance, the identity is represented by a UUID).

**CAUTION**

Though very rare, there is still a possibility that a BizTalk document may actually be successfully delivered, verified, and processed by the destination BFC server, yet for some reason (such as a transport failure) its delivery and/or commitment receipt may get lost. In this case, as a result, the source BFC server may generate a failure report based on the recommended behavior discussed earlier.

As you may have noticed from the previous discussion, there are three different deadlines that may be involved in a BizTalk document. The following table summarizes the distinct semantics of these deadlines:
### Deadlines Associated BizTags Explanation

<table>
<thead>
<tr>
<th>Deadlines</th>
<th>Associated BizTags</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delivery deadline</td>
<td>sentBy subtag of the deliveryReceiptRequest BizTag</td>
<td>This deadline concerns acceptance of the BizTalk document by the destination. The delivery receipt must be received by the source before the delivery deadline.</td>
</tr>
<tr>
<td>Commitment deadline</td>
<td>sentBy subtag of the commitmentReceiptRequest BizTag</td>
<td>This deadline concerns examination of content, verification of ability, and willingness to process the BizTalk by the destination. When requested, the commitment receipt must be received by the source before the commitment deadline.</td>
</tr>
<tr>
<td>Processing deadline</td>
<td>expiresAt</td>
<td>This deadline is the point in time beyond which the BizTalk document will be considered null and void if unprocessed. After this deadline, the document must not be delivered to the application at the destination end for processing, nor should any receipts (delivery and/or commitment) be sent to the source.</td>
</tr>
</tbody>
</table>

**NOTE**

The BizTalk Framework does not explicitly specify the required behaviors for the source and destination business entities regarding commitment receipts. Rather, it provides a standard framework which allows such application- and/or implementation-specific semantics to be expressed at the wire level.

### Handling Attachments

A BizTalk document can optionally carry attachments with its business documents, as sometimes required by the business process. The attachments are usually in binary forms such as an image file—for example, a photocopy of a signed time card. As illustrated in Figure 1-2, the primary BizTalk document, along with one or more attachments, comprise the compound content of a BizTalk Message.

The BizTalk Framework specifies a standard way to associate a primary BizTalk document with one or more attachments in a multipart Multipurpose Internet Mail Extensions (or MIME) structure for transport. Most Internet transports can handle transporting MIME-encoded content. In case of the HTTP protocol, some special considerations are required (these will be discussed later in the chapter).

The document catalog, specified by the manifest BizTag under the header section, includes references (specified by the reference BizTags which are usually Universal
Resource Identifiers or URIs) to both the business documents and the attachments. The business documents are specified by the `document` BizTag underneath the reference tag with an `href` attribute (similar to the `href` attribute for HTML) which points to the URI of the business document. The attachments are specified by the `attachment` BizTag underneath the reference tag, which also uses `href` attributes to reference the URIs of the attachments. Both the business document and the attachments can optionally have some descriptions, specified by the `description` BizTag, under the reference tag.

Managing Processes

The BizTalk Framework uses an optional `process` BizTag under the header section to specify process-management information. If the process tag is present, the SOAP-ENV:mustUnderstand = “1” attribute must be used to mandate the recipient BFC server to understand its contents and successfully process the BizTalk document. The following table summarizes the subBiz Tags under the process BizTag:

<table>
<thead>
<tr>
<th>BizTag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>A URI reference that represents the type of business process involved. It is a pattern of interchange between multiple BizTalk documents. For example, the process of purchasing a computer laptop can be specified as “Laptop_Purchase_Process.”</td>
</tr>
<tr>
<td>instance</td>
<td>A URI reference that uniquely identifies a specific instance of the business process associated with the BizTalk document. A common way is to extend the <code>type</code> of a URI with a fragment identifier such as a sequence number. For example, an instance regarding the process of purchasing a laptop could be expressed as “Laptop_Purchase_Process#10001.”</td>
</tr>
<tr>
<td>Detail</td>
<td>An optional BizTag that allows you to include further information as needed for the process. This tag can contain custom tags (non-BizTags) which provide the flexibility to describe the process in more detail.</td>
</tr>
</tbody>
</table>

Securing BizTalk Documents

When trading partners conduct business over the Internet, security is an important consideration. The data being exchanged must be secured for authentication, integrity, non-repudiation, and privacy. BizTalk handles security at the individual message, document, or attachment level, instead of at the transport level. As a result, single-hop privacy security protocols, such as Secure Socket Layer (or SSL), are not sufficient. The BizTalk Framework supports Secured MIME (or S/MIME) protocol version 3 when securing BizTalk messages, documents, and attachments.
BizTalk offers three types of securing modes:

- Encryption
- Signing (only supports the detached signature, multipart/signed mode)
- Both encryption and signing

The BizTalk Framework allows the header of the BizTalk document to be encrypted separately—and differently—from its body and attachments. This granular securing capability satisfies most B2B security scenarios in which the BizTalk document can be carried in clear for intermediaries and infrastructure components processing without compromising the privacy. The standard SOAP referencing is used to identify the encrypted documents and/or attachments by their href attribute.

**Binding the Transports**

The BizTalk Framework currently supports two transport protocols, Hyper Text Transfer Protocol (or HTTP) and Simple Mail Transport Protocol (or SMTP).

HTTP is a request/response, or synchronous transport protocol. This does not quite fit into the asynchronous messaging architecture of the BizTalk Framework. As a result, an HTTP successful response code (2xx status code) does not have the conventional meaning in the BizTalk messaging context. In other words, the source BFC server should not rely on the standard HTTP response code for ensuring a successful delivery or commitment. Instead, the delivery and/or commitment receipts mechanism described earlier in this chapter should be used as specified by the framework. To work around the limitation posed by the synchronous nature of the HTTP protocol (and the SOAP protocol which currently uses HTTP as the transport protocol), some asynchronous technologies, such as message queuing, can be used in conjunction with the HTTP protocol to provide a more robust solution.

The SMTP protocol, on the other hand, fits pretty well into the BizTalk messaging infrastructure because SMTP is also an asynchronous protocol. As a result, binding SMTP transport is more straightforward. SMTP is completely compatible with MIME and fully supports its semantics.

**BizTalk Framework 2.0 Developer’s Toolkit**

The BizTalk Framework 2.0 Developer’s Toolkit (formally BizTalk Jumpstart Kit envelope and Plug-In components) is a programming framework which allows Visual Basic developers to quickly build COM components based on appropriate
BizTalk schemas (XML schema documents). The COM components created by using the toolkit provide an object-oriented abstraction of the underlying BizTalk documents, hiding the complexity of processing an XML document through the Document Object Model (or DOM).

**TIP**


The toolkit can be used on both Windows 2000 and NT 4 with SP 4 or above.

**Introduction of the Toolkit**

The design of the toolkit is based on two logic abstractions, the *envelope* and *plug-in* concepts that map to the SOAP envelope and the business documents contained in the Body section of the BizTalk document as defined in the BizTalk Framework 2.0.

The SOAP envelope is represented by a COM component with a ProgId of BTFEnvelope.Envelope in a form of three ActiveX dll files, BTFEnvelopeXX.dll. Here the XX represents the different version of MSXML DOM parsers to be used, 20 for 2.0, 26 for 2.6, and 30 for 3.0, respectively. You will need to register the appropriate BTFEnvelope object on the target machine before you can use it. To register, run the Regsvr32.exe at the command line prompt. For example:

```
Regsvr32 C:\BTFDTK\BTFEnvelope30.dll
```

**NOTE**

You can register only one of the three BTFEnvelopes because they all share the same CLSID. Choose the version that is the same as the MSXML DOM parser version registered on the target machine. The default BizTalk Server 2000 installation comes with MSXML parser 3.0. We will discuss BizTalk Server 2000 installation in Chapter 2. MSXML parsers are discussed in Appendix C.

In addition, the toolkit also provides a plug-in COM interface along with a code generation wizard in the form of a Visual Basic 6.0 Add-in.

The plug-in interface is provided as a type library which only defines the MSPlugIn.PlugIn2 interface but does not contain any implementation. As with the BTFEnvelope, the plug-in type library comes with three forms, msplugin20.tlb, msplugin26.tlb, and msplugin30.tlb, corresponding to the three versions of MSXML.
DOM parsers. To register these type libraries, you need to run the accompanying registry file, Plugin2.reg. Before you can use this file, however, you need to open it with a text editor such as NotePad and modify the paths underneath the ..\Win32 key so that they correctly point to the location of the MSPluginXX.TLB file.

**NOTE**

The registry file Plugin2.reg that comes with the toolkit is a Windows 2000 registry file. In order to make it work under NT 4, you also need to replace the first line “Windows Registry Editor Version 5.00” with “REGEDIT4”.

To set up the Code Generation Wizard (Visual Basic 6.0 Add-in), you need to do the following:

1. Copy the SchemaPlugInGenerator.dll to `<Visual Studio Installation>\Common\MSDev98\AddIns` and register it. Visual Basic IDE will use this dll to launch the Code Generation Wizard and generate the plug-in code.
2. Copy the pigen.ini (the plug-in add-in configuration file) to the same location as the SchemaPlugInGenerator.dll.
3. Copy the class templates, Microsoft Plugin class instance.cls, Microsoft PlugIn collection instance.cls and Microsoft Plugin Main.cls, to `<Visual Studio Installation>\VB98\Templates\Classes`.
4. Copy the XML helper module modXMLHelper3.bas to `<Visual Studio Installation>\VB98\Template\Code`.

**Use the Toolkit**

The toolkit comes with two Word documents, EnvelopeAndPlugin.doc and UsingPluginGenerator.doc. The former explains the design and architecture of the toolkit in great detail, including the relationship between the BTFEnvelope and the plug-ins. The latter demonstrates how to use the plug-in generator to create the plug-in COM objects from a sample BizTalk schema, and how to use the generated COM objects in different scenarios. In this section, we will walk through the plug-in code generation process by using a sample purchase order BizTalk schema and the generated COM objects in different scenarios. In this section, we will walk through the plug-in code generation process by using a sample purchase order BizTalk schema that can be downloaded from BizTalk.org (we will introduce BizTalk.org in the next section of this chapter): [http://schemas.biztalk.org/BizTalk/z0124pf.xml](http://schemas.biztalk.org/BizTalk/z0124pf.xml). This sample purchase order BizTalk schema defines the structure of the business document, the purchase order XML document. You can directly view this downloaded BizTalk schema in Internet Explorer 5 or as shown in Figure 1-3.
The schema has a reference to an XSL stylesheet which outputs the XML document to an HTML document, as you see in Figure 1-3. If you are not connected to the Internet, you will need to download the stylesheet (g9boxj2.xsl) from the BizTalk.org Web site at: http://schemas.biztalk.org/BizTalk/g9boxj2.xsl and save the stylesheet to the same directly as the purchase order BizTalk schema (zi0123pf.xml). You also need to modify the stylesheet reference in the schema so it points to the local version of the stylesheet. To do this, open the schema (zi0123pf.xml) in a text editor and change the second line of text from...
NOTE

Don’t worry if you are not comfortable with the XML, XML schema, XSL style sheet, or XML DOM terminologies used in this section. In Appendix A, B, and C of this book, you will learn about XML and all its related technologies.

Now, let’s use this sample purchase order BizTalk schema to demonstrate how to use the plug-in Code Generator Wizard to create COM objects, and how to use the generated COM objects to manipulate the instances of purchase order documents without explicitly using the more complicated XML DOM programming model.

First, you need to start a new ActiveX DLL project from Visual Basic 6.0, name the project POSample, and remove the default class module by right-clicking the class module from the Project Explorer and selecting Remove Class1, as shown in Figure 1-4.

Figure 1-4  Remove Class1 from the POSample Visual Basic project
Now you are ready to run the plug-in Code Generator Wizard. If the toolkit has been installed and set up correctly as described in the preceding section, you should be able to access it from the Add-In Manager of the Visual Basic IDE, as shown in Figure 1-5.

After verifying that the plug-in code generator has been appropriately installed, you can start using it for this example: From the Add-Ins menu of Visual Basic IDE, select MS Schema Plug-in Generator. You will see an introductory screen of the MS Plug-in Generator Wizard. Click the Next button to take you to Step 1 of the wizard. Here you can select the XML schemas from which you can generate the corresponding plug-in object. You can load the schema either from a URL or from a local file. In this example, you choose the purchase order schema file you downloaded (zi0123pf.xml). The wizard screen will now look like the one in Figure 1-6. To view the entire schema in the wizard, click anywhere on the schema and press the down-arrow key on your keyboard.

Click Next to see Step 2 of the plug-in generator wizard (Figure 1-7) where you can specify a namespace for the XML document (XML namespaces will be explained in Appendix A). You can either specify the namespace here or keep the default value “YOUR_ORGANIZATION” and manually change it later from the classes generated by the wizard. For now, just keep the default and click Next.

Now you should see Step 3 as shown in Figure 1-8, the last step of the plug-in generator wizard. In this step, you will have a chance to name the classes created by the wizard. By default, the wizard will use the element names (tags) in the schema as the

**Figure 1-5** The BTF Developer’s Toolkit Schema Plug-in Generator in the Visual Basic Add-In Manager
class names. It may replace some special characters that violate Visual Basic naming conventions (such as classes containing an underscore character). In this example, you will use the default names for the classes. You can also specify the specific MSXML
parser version used on your computer in this step. Let’s assume that you have MSXML 3.0 installed, so you will select Version 3.0 as shown in Figure 1-8.

Clicking Next will bring you to the last screen of the plug-in generator wizard. There you can save the current settings as the default for the future. If you click the Finish button, the wizard will start generating the code for you. After a few seconds, you will end up with a completed Visual Basic ActiveX DLL project, with all the classes created by the wizard and ready for compilation. Figure 1-9 illustrates the Project Explorer view of the SamplePO Visual Basic project.

Now, let’s see what the plug-in generator wizard has done. As you can see in Figure 1-9, eleven class modules have been generated for you. All the class modules except for PO.cls are marked as PublicNotCreatable. The PO.cls corresponds to the top-level element (tag) underneath the Body element of the BizTalk document, the PO element. The rest of the class modules created by the wizard correspond to sub-elements of the PO element, or collections of some sub-elements. Therefore, you can only access the sub-elements through the top-level element—in this case, the PO element. As a result, the integrity of the business document has been guaranteed.

You can download the sample code, POSampleClient.vbp, which shows you how to use the wizard-generated plug-in objects to create a PO purchase order document defined by the BizTalk schema from the Web site of this book, located under the
Chap01\POSample\Client\ directory. The following listing illustrates the code that uses the plug-in objects to create an XML purchase order document:

```vbnet
Private Sub cmdCreatePO_Click()
    'Assume that you set the following references in the Project:
    'BTF Envelope Object (for example, BTFEnvelope30.dll);
    'Microsoft Plutin2 Type Library (for example: msplugin30.tlb);
    'Microsoft Scripting Runtime (scrrun.dll);
    'SamplePO.dll (the one you just created using the Toolkit).
    Dim oPO As New SamplePO.PO
    Dim oEnv As New BTFEnvelope.Envelope
    Dim oPlugIn As MSPlugin.PlugIn2
    Dim oFSO As New Scripting.FileSystemObject
    Dim oFile As Scripting.TextStream
    Set oPlugIn = oPO
    On Error GoTo CreatePO_Err
    oEnv.manifest.addReference oPlugIn.DocumentElementName, _
        oPlugIn.Namespace, "MY_Namespace", oPO

    With oPO.POHeader
        .poNumber = "1234567890"
    End With

    CreatePO_Err:
        'Error handling...
```

Figure 1-9  The generated code modules of the POSample Visual Basic project
.custID = "10001"

With .POShipTo
  .attn = "John Smith"
  .Addstreet = "123 Rose Road"
  .city = "Any town"
  .stateProvince = "Any State"
  .postalCode = "AS 12345"
  .country = "USA"
End With

With .POBillTo
  .attn = "John Smith"
  .Addstreet = "123 Rose Road"
  .city = "Any town"
  .stateProvince = "Any State"
  .postalCode = "AS 12345"
  .country = "USA"
End With

.Contact.Item(1).contactName = "John Smith"
.shipType = "UPS Next Day Air"

With oPO.POLines.Item(1)
  .Line = "1"
  .partno = 123
  .qty = 20
  .unitPrice = 1.99
  .totalAmount = .qty * .unitPrice
End With

Set oFile = oFSO.CreateTextFile(App.Path & "\SamplePO.xml", True)
oFile.WriteLine oEnv.XML
MsgBox "Sample PO has been created!"

Set oPO = Nothing
Set oEnv = Nothing
Set oPlugIn = Nothing
Set oFSO = Nothing
Set oFile = Nothing
Set oPlugIn = Nothing
Exit Sub
CreatePO_Err:
  MsgBox Err.Description
End Sub
Copy the sample project to your local drive and verify that all the preferences are appropriately set. Start the project and click the Create PO button on the form. You will see a message box indicating the PO document has been created (you can find the created PO XML document in the same directory where you installed the sample project on your local computer). Its file name is SamplePO.xml. Figure 1-10 illustrates the generated purchase order in Internet Explorer 5.5 (for brevity, the SOAP envelope tags are not shown in the figure).

In addition to creating a full BizTalk Framework 2.0–compliant document (the SOAP envelope) as you did in the preceding example, you can also use the COM objects generated by the toolkit to build regular, well-formed and valid XML documents.
documents. For other scenarios regarding the toolkit, refer to the documentation, UsingPluginGenerator.doc, that accompanies the toolkit. This document also lists some known issues of the toolkit and appropriate workarounds.

The BizTalk Framework 2.0 Development Toolkit provides a great way to simplify creating BizTalk documents or business documents based on a specific XML schema. This can be extremely helpful, as you will see later in this book, when we discuss using the BizTalk Editor (an XML tool that comes with BizTalk Server 2000) to create, edit, and manage Specifications (BizTalk Server-specific XML schemas).

**BizTalk.org**

The second part of the BizTalk initiative is the BizTalk.org web site, accessible at http://www.biztalk.org/. The original BizTalk.org steering committee consisted of more than a dozen industry leaders in EAI and B2B integration firms, including Microsoft, Ariba, Boeing, CommerceOne, Compaq, Dell, I2, NEON, RosettaNet, SAP, and Siebel Systems. The committee has since served its purpose and is disbanded. The BizTalk.org web site nevertheless carries on their work, providing a rich repository of BizTalk schemas and a growing community devoted to the exchange of best practices in BizTalk and XML. It also contains links to other related resources.

**NOTE**

There will be some significant changes to BizTalk.org in the coming months. By the time this book is published, the BizTalk.org may look very different than what is described in this section.

**A Repository of BizTalk Schemas**

As an important part of its makeup, BizTalk.org provides a library of BizTalk schemas contributed by more than 50 organizations, including such industries as Accommodation, Food, Agriculture, Forestry, Fishing, Construction, Education, Finance, Insurance, Health Care, Social Assistance, Information, Management, Manufacturing, Professional Services, Scientific, Real Estate, Rental, Leasing, Retail, and Wholesale Trade, etc.

Membership to the BizTalk.org library is free. You can browse the library for appropriate BizTalk schemas or search for a specific schema by keyword. Figure 1-11 shows 41 BizTalk schemas found using the keywords “purchase order.”
You can download the BizTalk schema and read related documentation. You can also contribute (register) your own schema to the library so it can be shared around the globe.

A BizTalk Developer Community


**NOTE**

*By the time you read this, these BizTalk discussion forums will have migrated to MSDN.*
A Source of BizTalk Resources

The BizTalk.org web site also contains rich links to other important resources, including BizTalk Framework specifications and white papers, third-party vendors, Microsoft XML Parser download, BizTalk Server 2000 downloads, and more.

Microsoft BizTalk Server 2000

The third part of the BizTalk initiative is Microsoft BizTalk Server 2000. Released on December 12, 2000 by Microsoft, BizTalk Server is the first BFC server to be completely compliant with BizTalk Framework 2.0 specifications. In addition, BizTalk Server 2000 provides a wealth of services, including messaging and orchestration services that greatly facilitate integrations within the organizational boundary (EAI) and between external business trading partners (B2B integration). This book will focus on BizTalk Server 2000. Chapter 2 will give you an introduction to the basics, including BizTalk Server architecture and its important services. Part II through Part IV will have in-depth and comprehensive coverage of BizTalk Server 2000, including its messaging and orchestration services, server administration, application development, the integration of BizTalk Server with other Microsoft .NET servers, as well as ways of extending BizTalk Server services. Part VI contains some case studies that will showcase how to use BizTalk Server 2000 in real world applications.

Chapter in Review

In this chapter, we have introduced you to the BizTalk initiative, including:

- BizTalk Framework 2.0, the backbone of BizTalk initiative
- The BizTalk.org web site, a dedicated BizTalk schema repository and developer community
- BizTalk Server 2000, the first BizTalk Framework 2.0-compliant server that reliably delivers XML-based messaging and orchestration services