XML Messaging with the SAP Business Connector Part 1:

Direct IDoc-XML Data Exchange and Outbound IDoc-to-XML Data Mapping with Flow Services

Robert Chu

Robert Chu joined SAP at the end of 1996. He currently works for the Integration and Certification Center at SAP Labs. His current focus is the SAP integration technologies. Robert has been regularly teaching classes in this area at SAP training centers, and is the main author of the BIT531 training course and a few other internal workshops.

(complete bio appears on page 78)

The Business Connector (BC) is SAP's current solution for enabling SAP components to participate in the exchange of XML messages via the Internet, mainly for the purpose of integrating SAP components with other solutions.[1]

XML messaging between your SAP system and, say, your business partner's system typically involves exchanging Intermediate Documents (IDocs[2]) between your SAP system and your SAP Business Connector, mapping data between the IDocs and the desired external XML format, and then exchanging the XML documents between the BC and the business partner's system.

While the Business Connector does provide an overwhelming amount of documentation, the coverage of XML messaging is difficult to follow and confusing. This article and the one that follows ("XML Messaging with the SAP Business Connector Part 2: Outbound IDoc-to-XML Data Mapping with XSLT and Java Services, and Inbound XML-to-IDoc Data Mapping") address this lack and provide an in-depth examination of how to use the BC in real-world projects to enable IDoc-based XML messaging with your business partners.[3] These two articles will help SAP Business Integration Consultants, SAP Integration Architects, and SAP Integration Developers understand the different options available when it comes to IDoc-based XML communication, and gives practical guidance on project implementation.

[1] At the time of writing, the SAP Exchange Infrastructure (XI) is still under restricted piloting.

[2] IDocs are used for the asynchronous communication between SAP systems and other SAP or non-SAP systems. Application Link Enabling (ALE) and the EDI interface both use IDocs.

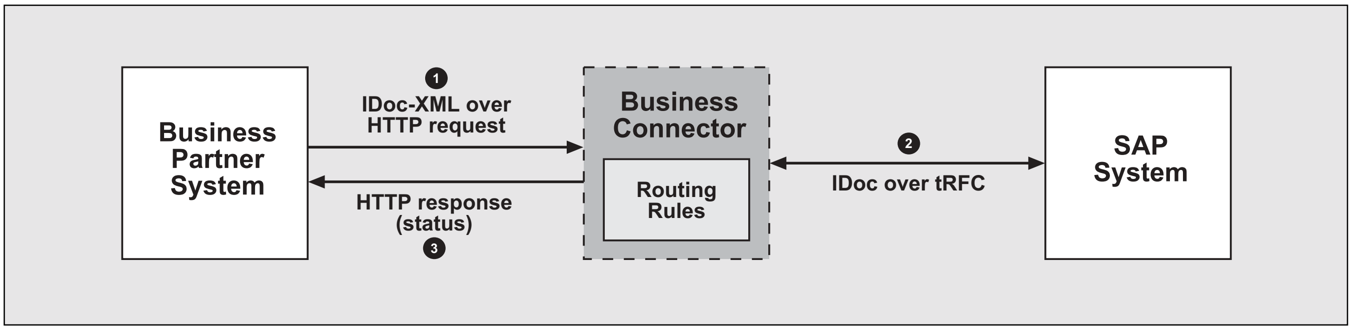
[3] SAP now offers the BIT531 training class ("SAP Business Connector Development"), which is a three-day class that Paul Medaille and I coauthored. These two articles will offer details in some areas that are not covered in the BIT531 class.

Listing 1: An Example IDoc-XML Document

```
<?xml version="1.0" encoding="iso-8859-1"?>
<BOMMAT03>
  <IDOC BEGIN="1">
    <EDI_DC40 SEGMENT="1">
      <DOCNUM>0000000000281751</DOCNUM>
      <DIRECT>1</DIRECT>
      <IDOCTYP>BOMMAT03</IDOCTYP>
      <MESTYP>BOMMAT</MESTYP>
      <SNDPRT>LS</SNDPRT>
      <SNDPRN>YSPCLNT400</SNDPRN>
      <RCVPRT>LS</RCVPRT>
      <RCVPRN>SPJ_M</RCVPRN>
      <!-- other control record fields omitted -->
    </EDI_DC40>
    <E1STZUM SEGMENT="1">
      <MSGFN>009</MSGFN>
      <!-- other E1STZUM segment fields omitted -->
      <E1MASTM SEGMENT="1">
        <MSGFN>009</MSGFN>
        <!-- other E1MASTM fields omitted -->
      </E1MASTM>
      <E1STKOM SEGMENT="1">
        <MSGFN>009</MSGFN>
        <!-- other E1STKOM fields omitted -->
      </E1STKOM>
      <E1STPOM SEGMENT="1">
        <MSGFN>009</MSGFN>
        <!-- first E1STPOM segment, other fields omitted -->
      </E1STPOM>
      <E1STPOM SEGMENT="1">
        <MSGFN>009</MSGFN>
        <!-- second E1STPOM segment, other fields omitted -->
      </E1STPOM>
      <E1STPOM SEGMENT="1">
        <MSGFN>009</MSGFN>
        <!-- third E1STPOM segment, other fields omitted -->
      </E1STPOM>
    </E1STZUM>
  </IDOC>
  <IDOC BEGIN="1">
    <!-- second IDoc instance in the IDoc packet, details omitted -->
  </IDOC>
  <!-- other IDoc instances in the IDoc packet, omitted -->
</BOMMAT03>
```

Here in this first article, I will walk you through how you and your business partners can use the IDoc-XML format to send and receive XML documents directly. However, not all of your partners will have this capability, so in the latter sections of this article, I will show you how to send documents to your business partners in other XML formats using a Business Connector *flow service*, which is a widely

Figure 1 *Inbound IDoc Communication Using IDoc-XML*



used technique. It is also possible to use XSLT transformation or Java services to send non-SAP XML documents to your business partners. These two techniques require additional knowledge of XSLT and Java, respectively, so I will discuss them in the second article of this two-part discussion, as well as show you how to enable your system to *receive* non-SAP XML documents from your business partners.

To get the most out of these articles, you will find the following prior knowledge very helpful:

- Basic understanding of SAP IDoc technology
- Basic understanding of Business Connector concepts and operations
- Basic understanding of the HTTP protocol
- Basic understanding of XML and XSLT
- Basic understanding of the Java programming language

Note that I will be using SAP Business Connector 4.6 in the discussion, since it is the latest recommended version from SAP.[4]

For your reference, sample BC services, along with Java source code and files, have been developed to illustrate all the major points in the articles. You can find those files, as well as the instructions to use them, at **www.SAPpro.com**.

[4] There is a new version (4.7) planned for Q1 2003.

SAP IDoc-XML or Not?

Out-of-the-box, the SAP Business Connector supports the so-called IDoc-XML vocabulary for IDocs. IDoc-XML is a straightforward mapping of IDoc data to XML: the IDoc type becomes the root element, with multiple IDoc children elements representing the multiple IDocs in an IDoc packet. Each IDoc segment becomes an XML element, with all its data fields and nested segments becoming children elements. The IDoc field value becomes the text content of the corresponding XML element. **Listing 1** is an example IDoc-XML document for IDoc type *BOMMAT03*.

If your business partner is able to support the IDoc-XML vocabulary, whether through prior agreement or because the partner is also running an SAP system, you can use the IDoc-XML documents directly for XML messaging. In this case, there is no development work that needs to be done in the Business Connector. You need only configure the routing rules appropriately in the BC to route the IDoc message to the desired destination. The translation between the IDoc-XML document and the IDoc message is performed automatically by the BC.

Inbound IDoc Communication Using IDoc-XML

Figure 1 is a diagram showing *inbound* IDoc communication using SAP IDoc-XML (with the SAP system as the receiver).

The major steps involved in inbound communication using IDoc-XML are as follows:

1. The business partner's system composes an IDoc-XML document and posts it to the BC IDoc *InboundProcess* service (URL path */pub.sap.transport.ALE/InboundProcess*).

2. The BC IDoc *InboundProcess* service extracts the *sender/receiver/msgType* information from the IDoc-XML message and forwards it to the BC gateway manager.[5] With this information, the BC gateway manager looks up and executes the predefined routing rule, using the tRFC[6] protocol to transmit the IDoc (translated from the IDoc-XML document) to the SAP system.

3. The BC sends the HTTP response, with a status code indicating the success or failure of the inbound IDoc transmission, back to the business partner's system.

To guarantee the once-and-only-once delivery of the IDoc, a transaction ID (TID) must be used. The business partner's system needs to generate a TID before posting the IDoc-XML document to the BC.

A TID is a globally unique string consisting of 24 characters (similar to a GUID[7]). The business partner's system can ask the BC to generate a TID via posting to URL path */invoke/pub.sap.client/createTID*, or it can calculate a TID using its own routine. After the TID is created, it must be attached to the initial posting of the IDoc-XML document using the HTTP header field *X-tid*. In case of failure, the TID must also be attached to all the subsequent repostings of the IDoc-XML document. Once the IDoc posting is successful, the business partner's system should confirm the TID by posting to the URL path */invoke/pub.sap.client/confirmTID*.

[5] The gateway manager is a BC component responsible for the routing of the messages.

[6] Transactional RFC (tRFC) guarantees the once-and-only-once delivery of IDoc (and other) messages.

[7] Globally Unique IDentifier.

When posting the IDoc-XML document to the BC, the following HTTP request header fields need to be set:

- **Content-type:** *application/x-sap.idoc*
- **X-tid:** *the_24_chars_tid*

The *sender/receiver/msgType* routing information must be specified using the following IDoc control record fields:

- **SNDPRN:** *sender*
- **RCVPRN:** *receiver*
- **MESTYP:** *msgType*

The routing rule must be configured as follows:

- **Transport:** *ALE*
- **Specify SAP Destination:** *the_SAP_system_to_post*

If the IDoc is posted successfully to the SAP system, the BC will return HTTP status code *200*. Any other status code indicates an error. If the BC answers with status code *200*, the business partner's system should confirm the TID. If the BC answers with any other status code, the business partner's system needs to schedule reposting with the same TID.

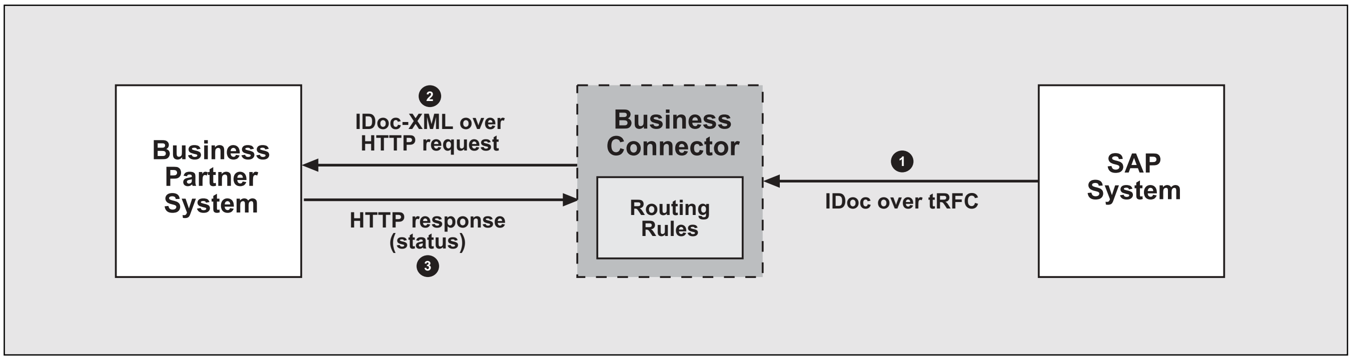
Outbound IDoc Communication Using IDoc-XML

Figure 2 is a diagram showing *outbound* IDoc communication using IDoc-XML (with the SAP system as the initiator).

The major steps involved in outbound communication using IDoc-XML are as follows:

1. In SAP ALE customizing, the BC is considered a logical system. Based on the ALE configuration, the SAP system sends the IDoc to the BC via the

Figure 2 *Outbound IDoc Communication Using IDoc-XML*



tRFC protocol.[8] (For an overview of setting up the ALE configuration, please refer to the upcoming section "Configure SAP to Send IDocs to the BC.")

2. The BC looks up and executes the predefined routing rule, converts the IDoc to an IDoc-XML document, and posts it to the business partner's system via HTTP request.

3. The business partner's system receives and processes the IDoc-XML document, and then sends the appropriate HTTP status code back to the BC.

The routing rule must be configured as follows:

- **Transport:** *XML*
- **Specify URL:** *the_target_URL_of_the_business_partner_system*
- **XML dialect:** *SAP-XML*

The business partner's system needs to handle the TID correctly by performing the following tasks:

- Extract the TID from the HTTP request header field *X-tid*.
- Use some kind of persistent storage medium (e.g., a database table or an operating system file) to store the processed TIDs.

[8] It is also possible to use the EDI interface to send EDI IDocs from the SAP system to the BC. However, using the ALE configuration is easier and more common.

- Check for duplicate TIDs to detect any possible retransmissions:
  - *For a TID received for the first time*, process the IDoc and log the TID in the chosen TID storage medium.
  - *For a TID already received and processed*, it is a retransmission, discard it.

The business partner's system needs to answer with a positive HTTP response (status code *200*) to indicate successful processing, including those of duplicate transmissions. If an unrecoverable error occurs, the system needs to send a negative HTTP response (a status code other than *200*).

XML messaging based on IDoc-XML is easy to implement. However, in many cases, your business partners will want to exchange XML documents in formats other than SAP IDoc-XML — e.g., xCBL, RossettaNet, or ebXML. In this case, you will need to perform additional data mappings between IDocs and the external XML formats in your BC service.

Outbound IDoc Communication with Non-SAP XML

For the remainder of this article, we will look at *outbound* IDoc communication using non-SAP XML, where your SAP system sends IDocs to the Business Connector, the Business Connector maps them to some external non-SAP XML format, and the

Business Connector then sends them to the business partner's system.

There are several main tasks involved in outbound IDoc communication when using non-SAP XML:

- Configure SAP to send IDocs to the BC.
- Configure the BC to receive IDocs from SAP.
- Capture a sample IDoc for development and testing.
- Perform the actual mapping of the data.
- Post the XML document to the external system.

So let's begin at the beginning.

Configure SAP to Send IDocs to the BC

As I mentioned earlier, from the SAP IDoc transmission perspective, the BC is considered a logical system. This means that setting up the SAP system to exchange IDocs with the BC is no different from setting up SAP to exchange IDocs with other systems in the ALE scenario.[9]

The following ALE configuration tasks must be in place in order to send IDocs from the SAP system to the BC:

- An RFC destination that points to the Business Connector must be defined inside the SAP system (transaction *SM59)*.
- A logical system must be defined for the BC. In order to generate partner profiles automatically, the names of the logical system and the RFC destination should be the same.
- An ALE distribution model view must be created. The desired IDoc message type, sender, and receiver need to be defined in the view.
- The partner profile needs to be generated for the distribution model view and the partner system.

[9] The details of setting up ALE distribution are beyond the scope of this article. Please refer to the corresponding SAP documentation and training courses for more information. For your convenience, I provide a high-level overview here.

This ALE configuration provides information to the ALE layer on how to distribute the IDoc once it is created.

Configuring the SAP system to *create* the IDocs is another issue. This depends on the IDoc you want to transmit and the application module involved:

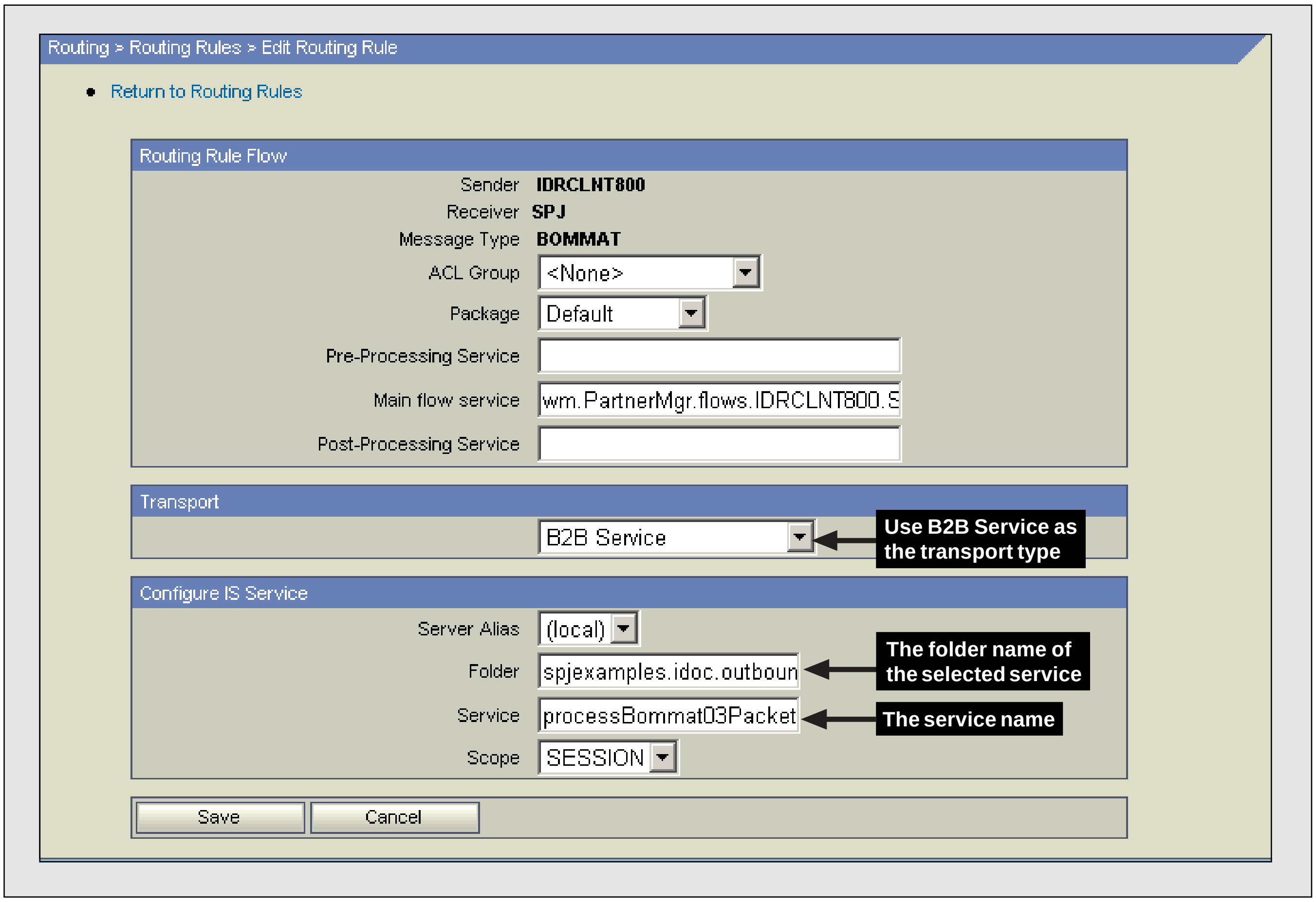
- For master data IDocs (e.g., *MATMAS*), a change pointer can be activated to record all the changes to the master data. Later, the change pointers can be processed to generate the IDocs.
- For master data IDocs, there are also SAP transactions available to manually generate IDocs. These transactions are available under *Tools → ALE → Master Data Distribution*. They are useful for initial data synchronization and for testing.
- For MM/SD IDocs (e.g., *ORDERS*), the message control (a.k.a. output determination) mechanism is used to generate the IDoc automatically, once the corresponding application document (e.g., sales order or purchase order) is posted.
- For some other applications, special transactions/ reports can be executed to generate the IDoc.

Once you've configured your system to create and send IDocs to the BC, you have to enable the BC to *accept* the IDocs. We will look at this task next.

Configure the BC to Receive IDocs from SAP

When the Business Connector receives an IDoc from the SAP system, by default it will use the following information for routing:

Figure 3 *An Example Routing Rule Specification*


Routing > Routing Rules > Edit Routing Rule

- Return to Routing Rules

Routing Rule Flow

Sender	IDRCLNT800
Receiver	SPJ
Message Type	BOMMAT
ACL Group	<None>
Package	Default
Pre-Processing Service	
Main flow service	wm.PartnerMgr.flows.IDRCLNT800.S
Post-Processing Service	

Transport

B2B Service

Use B2B Service as the transport type

Configure IS Service

Server Alias	(local)
Folder	spjexamples.idoc.outboun
Service	processBommat03Packet
Scope	SESSION

The folder name of the selected service

The service name

Routing Key	Value
sender	Logical system name of the SAP client
receiver	Logical system name of the BC
msgType	The IDoc message type

The routing rule controls what the BC does with the IDoc. Some typical routing configurations include the following:

- Use XML transport to forward the IDoc-XML directly to an **external HTTP server**. As discussed previously, if the business partner agrees to receive the IDoc-XML document directly, you can set up the routing rule using the transport type *XML* and specify the URL of the business partner's system.

- Route the IDoc to a **BC service** (that you have created), where it will be further processed. In the BC service, you can perform any tasks that are desired (e.g., mapping the IDoc data to a different XML vocabulary) and then transmit the document to the business partner.

Figure 3 shows an example routing rule of type *B2B Service* where the *BOMMAT* IDocs received by the *SPJ* BC from the *IDRCLNT800* system are routed to a BC service called *processBommat03PacketWithFlow* in the *spjexamples.idoc.outbound* folder.

Now that the BC is configured to route the IDocs it receives, it is a good time to capture a sample IDoc in the pipeline[10] and use it later to develop and test the BC service. Doing this now will save you a lot of time and effort later — you will not need to generate and send an IDoc from the SAP system each and every time you want to test your service!

Capture a Sample IDoc for Development and Testing

To capture the sample IDoc, you can invoke the *pub.flow:savePipeline* service from within your flow service[11] to save the contents of the pipeline (which includes the IDoc that the partner manager routes to this service) into memory, identified by the *$name* parameter value. (Alternatively, you can use *pub.flow:savePipelineToFile* service to save the pipeline to a file, identified by the *fileName* parameter value, which will survive any BC restarts.)

In the SAP system, you need to generate and send to the BC the comprehensive sample IDoc that contains all the necessary segments and fields. When the BC partner manager receives the IDoc, it invokes your flow service, which will capture the IDoc by saving the pipeline.

After successfully capturing the IDoc pipeline, you can delete or disable the *savePipeline* (or *savePipelineToFile*) step from the flow, then insert an invocation to *pub.flow:restorePipeline* (or *pub.flow:restorePipelineFromFile*). When you run the flow service, you will see the restored pipeline containing the IDoc data.

In the example download, the file *Bommat03PacketPipeline.xml* is a saved pipeline file containing three *BOMMAT03* IDocs received from the SAP system. The file *Bommat03SinglePipeline.xml* is another saved pipeline file containing one *BOMMAT03* IDoc received from the SAP system.

[10] With the BC, "pipeline" refers to the data container that contains all the input data available to a BC service, as well as all the results data produced by the execution of the BC service.

[11] Using a flow service is the easiest way to capture a sample IDoc in the pipeline.

Perform the Actual Mapping of the Data

With your SAP system configured to send IDocs and the BC configured to receive them, and with a sample IDoc at your disposal for testing, you are now ready to perform the actual data mapping that converts the IDoc data into the desired XML format. There are three options available for this task:

- Perform data mapping using a custom BC flow service.
- Perform data mapping using XSLT transformation with a custom XSLT stylesheet.
- Perform data mapping using a custom Java service.

In the following sections, I will discuss in detail the first option, which is the most widely adopted one in XML messaging scenarios. The other two options, which require additional knowledge of XSLT and Java, will be covered in the second article of this two-part discussion. Let's now take a detailed look at what's involved in using a custom BC flow service to map your IDoc data to an external XML format.

Data Mapping Using a BC Flow Service

A flow service is a service that is written in the BC "flow" language. This simple yet powerful language lets you encapsulate a sequence of BC services within a single service and manage the flow of data between them. Over the past few years, many data mapping projects have taken advantage of BC flow services. For example, version 2.0 of the SAP MarketSet Connector[12] was developed using mainly flow services for the data mapping.

Using a flow service to perform data mapping in the BC has the following advantages:

- BC flow services were developed using the BC Developer, which provides a capable and easy-to-use development environment. No additional development tools are needed.

- BC flow services were developed in an intuitive, graphical, drag-and-drop fashion, so very little, if any, coding is necessary.

- A flow service allows for easy manipulation of the data in the pipeline, which is very useful in performing data mapping.

- There are many built-in services readily available, which you can easily use in your own flow service.

The main potential disadvantage of using a BC flow service for data mapping is that it is a proprietary Business Connector technology. Thus it is difficult to port the flow-service-based mapping to other environments (e.g., the future SAP Exchange Infrastructure).

Mapping IDoc data to an XML format using a BC flow service typically involves the following four steps:

1. Create a record definition for the IDoc type.

2. Create a record definition for the XML target.

3. Transform the IDoc into a hierarchical structure.

4. Map the IDoc to the target XML format. This step includes the following considerations:
   - MAP and LOOP operations
   - Handling an IDoc packet or recurring IDoc segments
   - Generating XML from the mapping results
   - Performing value lookups

Let's take a closer look at each of these four steps and the related considerations involved in each.

Step 1: Create a Record Definition for the IDoc Type

To perform data mapping using a flow service, you first need to create the BC record definition[13] for the source IDoc type. A BC record definition (also known simply as a BC record) can be used to specify input and output parameters for a BC service, and can also be used to build a record variable (an instance) or a record list variable (an array of instances) in the service. There are several different ways to create a record definition for an IDoc.

Reference the IDoc Schema Available from the SAP IFR

The SAP Interface Repository (IFR) is a public web site (**http://ifr.sap.com**) containing XML schema documents for all the SAP standard interfaces, including all the standard SAP IDoc types. The following is the procedure for creating a BC record definition based on an IFR schema:

1. Download the appropriate version[14] of the XML schema document for the IDoc type from the SAP IFR site.

[12] The SAP MarketSet Connector is a connectivity tool that enables mySAP backend systems to connect with MarketSet-based marketplaces via the Internet.

[13] A BC record definition is essentially a type definition.

[14] The 4.6 BC release supports the W3C XML Schema Recommendation (May 2001).

2. In the BC Developer, create a new record (not schema!). Select *XML Schema* in the *Select a source for the XML format* dialog box, then select the downloaded schema file. A new record definition and a new schema[15] will be created.

3. Modify the created record definition by changing the type of the element *IDOC* (which is directly below the top-level element corresponding to the IDoc type) from a record to a record list. This is necessary because the IFR IDoc schema incorrectly marks the *IDOC* element as occurring only once, although it may actually occur multiple times in an IDoc packet.

✓ *Note!*

The SAP Interface Repository contains schemas only for SAP standard IDocs.

Reference the DTD of the IDoc Generated from Transaction WE60

The IFR does not contain schemas for custom-developed IDocs or extended/reduced IDocs. In this case, if you have a 4.6 or later SAP system, you can generate a DTD for the IDoc type in the SAP system using transaction *WE60*:

1. In the initial screen, enter the name of the IDoc type, choose menu path *Documentation* → *Create DTD*, select the path and file name for your DTD, and choose *Save*.

2. Indicate the root element by creating an XML wrapper file for the DTD (e.g., assume we have a DTD file for IDoc type *BOMMAT03* called *BOMMAT03.DTD*) as follows:

```
<?xml version="1.0"?>
<!DOCTYPE BOMMAT03 SYSTEM
  "BOMMAT03.DTD">
<BOMMAT03/>
```

3. Create the record definition in the BC Developer by selecting *XML* in the *Select a source for the XML format* dialog box, and then selecting the XML wrapper file for the DTD. A new record definition and a new schema will be created.

Capture a Sample IDoc Instance in a Flow

If you are working with non-standard IDocs, and the SAP system release is 4.5 or earlier, the only option left for creating a record definition for the IDoc type is to capture a representative sample IDoc instance in a flow service, as described earlier in the section "Capture a Sample IDoc for Development and Testing." You can create the record definition for the IDoc type by copying and pasting the IDoc data structure from the captured IDoc sample.

The BC record definition for the IDoc type has the following characteristics:

- The *IDOC* element is a record list, representing the IDoc packet.
- The control record of the IDoc is either the *EDI_DC40* element for version 3 IDocs (as used by R/3 4.0 and later) or the *EDI_DC* element for version 2 IDocs. They contain all the control record fields.
- Data segments are organized in the correct hierarchy as defined by the IDoc type.
- Each data segment contains all its data fields and nested segments (if any).
- The dimension of a data segment is either a record, if the segment occurs only once, or a record list, if it can occur multiple times.

[15] The BC schema is different from the XML schema. It is a byproduct of creating the BC record definition, and can be used to validate an XML document.

This IDoc record definition can be used to derive other record definitions if necessary — for example, by copying the data structure of a particular segment to create a new record definition. These derived record definitions can be useful in dealing with recurring elements using the LOOP operation. (For details, please refer to the upcoming discussion of the LOOP operation in Step 4.)

In the example BC package *SPJ* included with the download, you can find the following examples of BC record definitions for IDocs:

- *spjexamples.records:Alereq01Rec* — The record definition for IDoc type *ALEREQ01*
- *spjexamples.records:Bommat03Rec* — The record definition for IDoc type *BOMMAT03*
- *spjexamples.records:Orders02Rec* — The record definition for IDoc type *ORDERS02*

Once you've created a record definition for the source IDoc type, using one of the options just described, you next need to create a definition for the target XML format.

Step 2: Create a Record Definition for the XML Target

If your business partners require a non-SAP XML format, normally they will provide you with the XML schema document, the DTD of the desired XML format, or a sample XML document. To create the BC record definition, you would refer to the schema document, DTD, or XML sample accordingly. This target record definition can then be used as a record reference in the pipeline of your flow service to define a Pipeline Out variable.[16] You will map the IDoc data to this variable.

[16] In other words, you declare a Pipeline Out variable as an instance of the record definition.

In the example package *SPJ*, the record definition *spjexamples.records:ExtBomRec* is created by referring to the DTD file *ExtBom.dtd*. The record definition *spjexamples.records:MaterialDataRequestRec* is created by referring to the DTD file *MaterialDataRequest.dtd*.

With the record definitions now in place for the source IDoc and the target XML structure, you must now transform the IDoc into a structure that is compatible for mapping between them. We'll look at this task next.

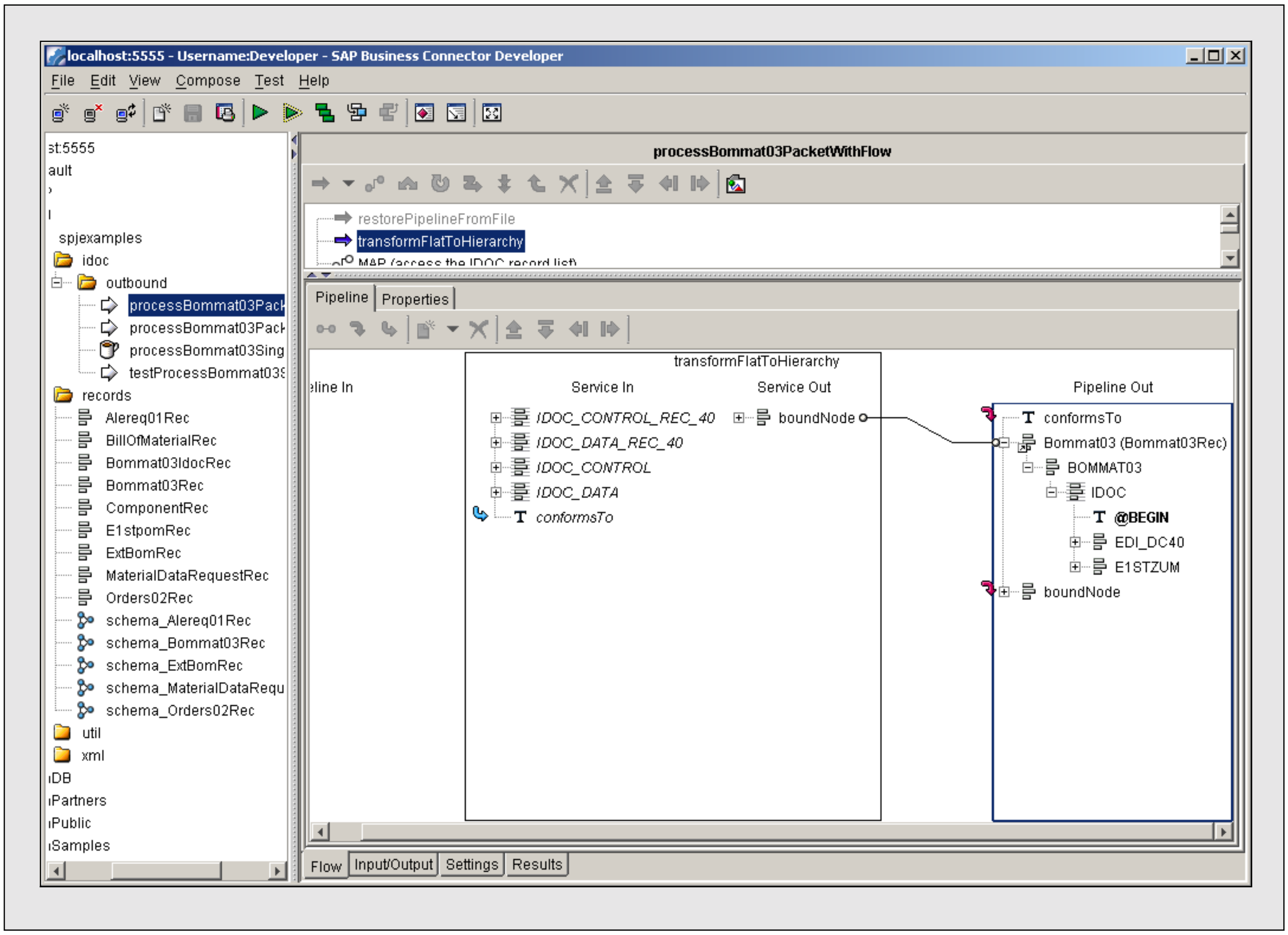
Step 3: Transform the IDoc into a Hierarchical Structure

Once the BC receives the IDoc from the SAP system via tRFC, it will apply the predefined routing rule to route the IDoc to the BC flow service that you want to use. The BC flow service will receive the IDoc in a "flat" structure, which contains *IDOC_CONTROL* and *IDOC_DATA* in the pipeline for version 2 IDocs, or *IDOC_CONTROL_REC_40* and *IDOC_DATA_REC_40* for version 3 IDocs. *IDOC_CONTROL* and *IDOC_CONTROL_REC_40* are Table objects containing the IDoc control record's data. *IDOC_DATA* and *IDOC_DATA_REC_40* are Table objects containing the IDoc data records.

This flat structure cannot be used directly in the mapping. To map from or to an IDoc, you need to transform the IDoc into a "hierarchical" structure.[17] To do this, insert the BC built-in service *pub.sap.idoc:transformFlatToHierarchy* into the flow

[17] One IDoc packet can contain multiple IDoc instances; a single IDoc instance typically contains multiple segments; and one IDoc segment typically contains multiple fields. Some segments also contain nested segments. All of these together form a hierarchical tree structure. The IDoc in its hierarchical structure allows mapping from or to the IDoc segments and fields.

Figure 4 Converting the IDoc to a Hierarchical Structure



(see **Figure 4**). This service transforms the IDoc into a *boundNode* variable, which contains the contents of the IDoc in a format that you can map from and to. The service has an optional parameter called *conformsTo*, which is the fully qualified name[18] of the record definition of the IDoc (for example, *spjexamples.records:Bommat03Rec*). It is highly recommended that you always specify this parameter. With the *conformsTo* parameter providing the record structure of the IDoc, the *pub.sap.idoc:transformFlatToHierarchy* service will be able to distinguish between an IDoc segment that occurs only once and one that occurs multiple times, and will thus correctly transform any segment into a record or a record list. Otherwise, the IDoc segments will always be transformed into a record list.

To access the segments and fields of the IDoc at design time, you also need to apply (bind) the IDoc record definition to the *boundNode* output variable of *pub.sap.idoc:transformFlatToHierarchy*. This makes the IDoc segments and fields accessible in the subsequent flow steps at design time. You can do this by adding a Pipeline Out variable as a record reference to the IDoc record definition, and then linking the Service Out variable *boundNode* to it (shown in Figure 4).

With the IDoc now in a workable format, you are finally ready to tackle the actual mapping process, which involves several considerations.

[18] With the format *folder.subfolder:record*.

Figure 5 *Pipeline Modifiers*

Use this modifier…	With this icon…	To…
Map	⚯	Map a source variable to a target variable. The Map modifier lets you resolve variable name and data structure differences by mapping (copying) the value of one variable to another at runtime.
Drop	⤵	Drop a variable from the pipeline. The Drop modifier removes extraneous variables from the pipeline.
Set Value	↪	Assign a value to a variable. The Set Value modifier hard-codes a value or sets the default value for a variable.
Insert	🗋 ▾	Insert a variable in the pipeline as a Pipeline In/Out variable, or, for a transformer, as a Service In/Out variable. When adding a variable, you must select a type for it (e.g., string, record reference, record reference list, etc.).

Step 4: Map the IDoc to the Target XML Format

Before you can implement the mapping in the BC service, you need to carefully study the source IDoc structure and the target XML format, identify the relationship between the available data in each, and determine the transformation necessary to produce the desired format. Once you have made these determinations, you can then implement the transformation with BC flow operations.

Among the different flow operations to consider, the following are particularly useful in implementing the transformation: the MAP operation and the LOOP operation.

The MAP Operation

The BC MAP operation lets you adjust the contents of the pipeline at any point in a flow service. By selecting the *Pipeline* tab for the MAP operation, you can access the Pipeline Editor. The Pipeline Editor offers a graphical representation of all your data. In the Pipeline Editor of the MAP operation, you can map input variables to output variables (i.e., create links) or insert transformers.

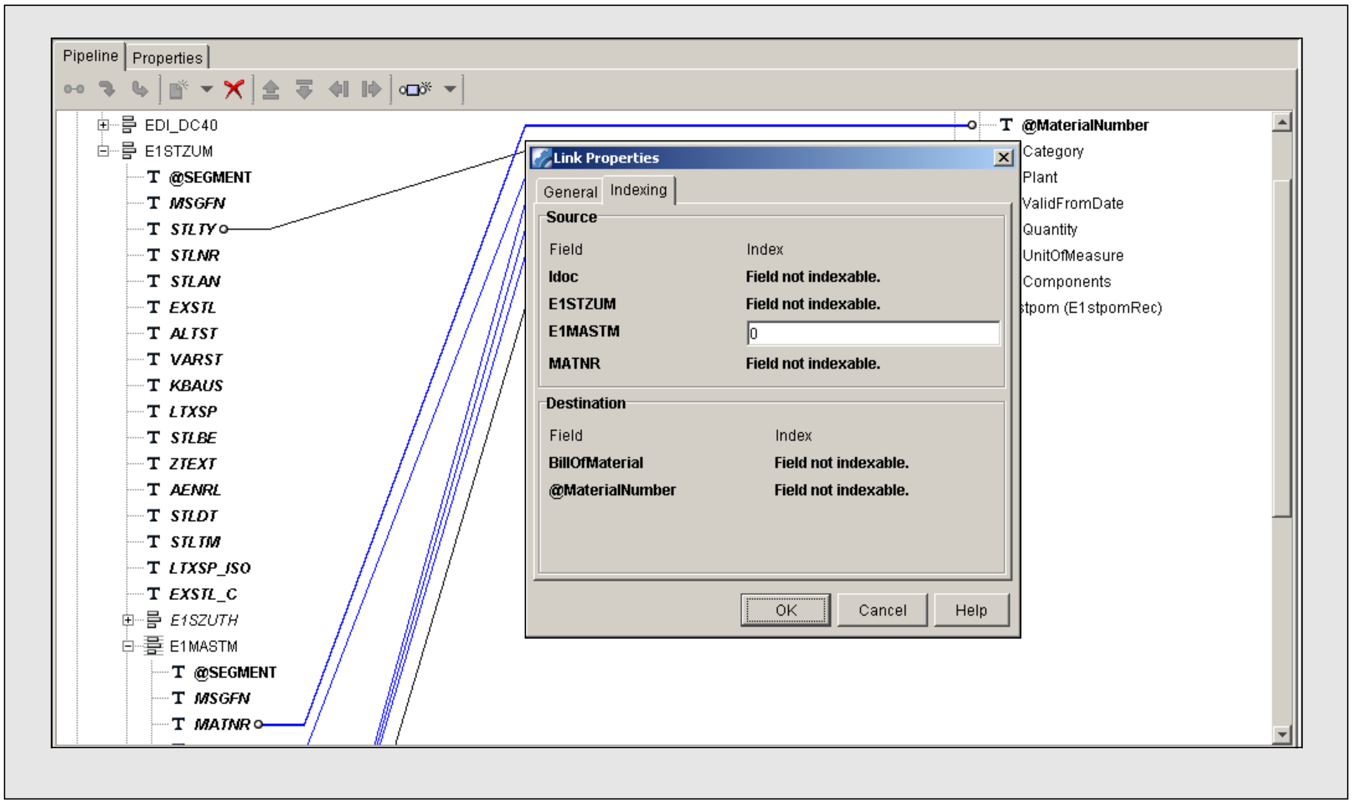
The Pipeline Editor of the MAP flow step displays two sets of variables: Pipeline In and Pipeline Out. Between these sets of variables, the Pipeline Editor displays transformers.[19] Let's take a closer look at these three columns:

- The **Pipeline In** column represents input to the MAP flow step. It contains the names of all the variables in the pipeline at this point in the flow.
- The **Transformers** column displays any services inserted into the MAP flow step to perform value transformations.
- The **Pipeline Out** column represents the output of the MAP flow step. It contains the names of variables that will be available in the pipeline when the MAP flow step completes.

When you first insert a MAP step into your flow, the contents of the Pipeline In and Pipeline Out columns are identical. Using the Pipeline Editor, you can insert pipeline modifiers, which are special commands you can use to adjust the contents of the pipeline at runtime (see **Figure 5**).

[19] Transformers in MAP operations are actually invocations of other BC services to convert certain input data to the pertinent output data (i.e., to perform a value transformation).

Figure 6 The "Link Properties" Dialog Box



Link Properties

General | Indexing

Source

Field	Index
Idoc	Field not indexable.
E1STZUM	Field not indexable.
E1MASTM	0
MATNR	Field not indexable.

Destination

Field	Index
BillOfMaterial	Field not indexable.
@MaterialNumber	Field not indexable.

In the Pipeline Editor, you can perform many mapping tasks, including mapping variables, adding variables to the pipeline, adding transformers to the MAP flow step, and so on.

You can copy the value of a variable to another variable by creating an explicit link between them using the Map modifier (). This is how you accomplish name and structure transformations required in a flow.

When executing a mapping between variables at runtime, the BC does one of the following:

- **Copies the value from the source variable to the target variable:** This is called *copying by value*. The BC copies by value when the source or target variable is a string.

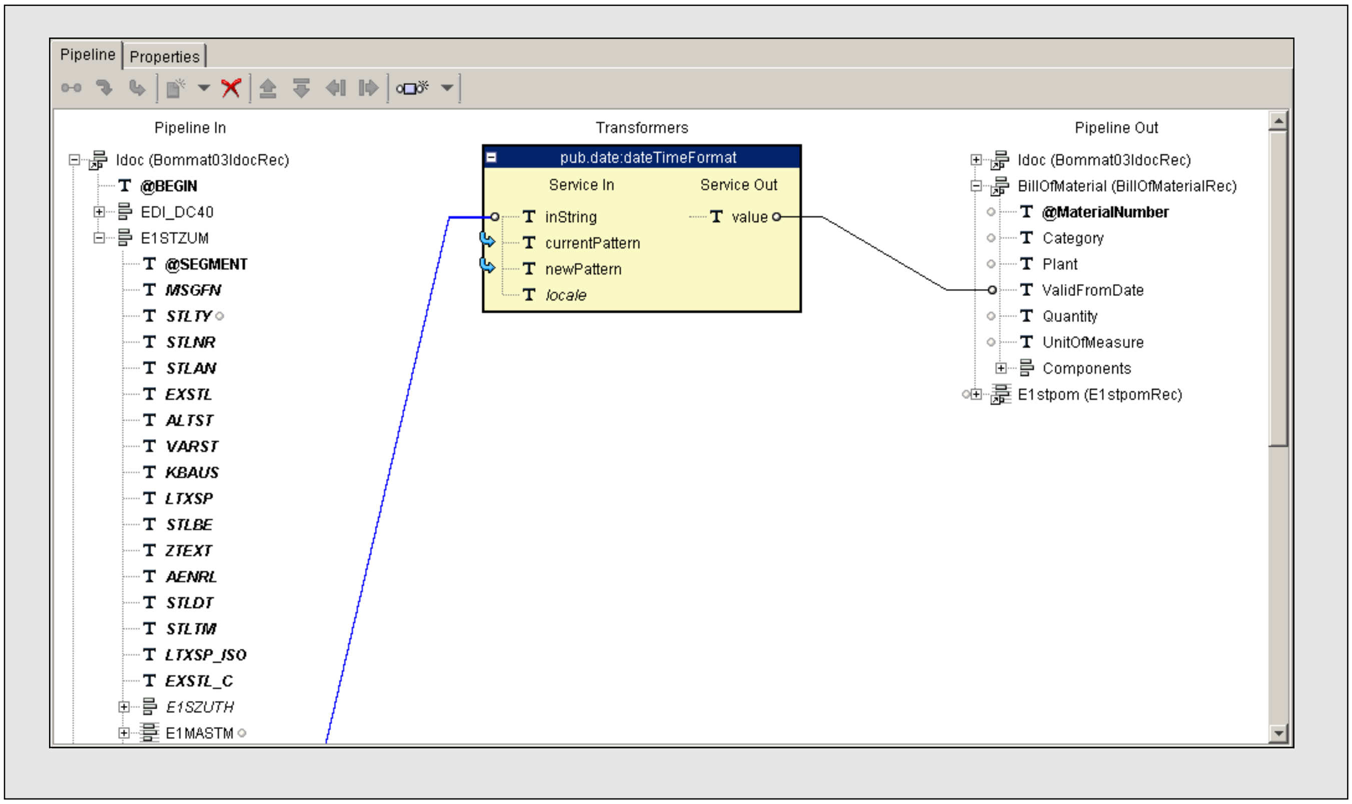
- **Creates a reference to the source variable and assigns it to the target variable:** This is called *copying by reference*. When executing mappings between all other types of variables, the BC copies by reference.

When you map to or from an array variable,[20] you can indicate which element in the array you want to map to or from. To do this, access the *Link Properties* dialog box shown in **Figure 6** by right-clicking on the link. On the *Indexing* tab, specify the index value that represents the position of the element in the array. As you can see in Figure 6, the array index numbering begins at zero.

You can also specify a conditional expression (e.g., *Copy only if…*) on the *General* tab of the *Link Properties* dialog box. At runtime, the BC will then evaluate the condition expression and execute the mapping (e.g., copy the value) only if the condition evaluates to true.

[20] Also known as a record list variable.

Figure 7 *Inserting a Transformer in a MAP Flow Step*



Sometimes you may want to add variables that were not declared as input or output parameters of the flow service, add input or output variables for services that the flow service invokes as transformers, or create temporary variables for use within the flow. To do this, highlight the Pipeline In or Pipeline Out column, click on the Insert modifier (), select the desired type, and then specify the variable name. If the record definition already exists for the variable you want to add, add the variable as a record reference or a record reference list.

✓ *Note!*

If you create a new variable in a flow, you must immediately map to or from it, set a value for it, or drop it. Otherwise the Pipeline Editor automatically clears it from the mapping the next time it refreshes the Pipeline tab.

Often you need to execute some code or logic to perform value transformations — that is, you need to invoke a service. You can insert invocations to other services as transformers in the Pipeline Editor.

You can use any existing service as a transformer. You can also add multiple transformers (i.e., invoke multiple services) in a single MAP step. The BC provides many built-in services specifically designed to translate values between formats. These services can be found in the following BC folders: *pub.date*, *pub.list*, *pub.math*, *pub.record*, and *pub.string*. They are often used as transformers. Of course, you can write your own services and use them as transformers as well.

When you insert a transformer (see **Figure 7**), you need to map variables between the pipeline and the transformer (i.e., create links between the Pipeline In/Out variables and the transformer Service In/Out variables).

The LOOP Operation

The LOOP operation repeats a sequence of child steps once for each element in an array that you specify. To specify the sequence of steps that make up the body of the loop, you indent those steps beneath the loop.

The LOOP operation requires you to specify an input array that contains the individual elements to be used as input to the one or more steps in the body. At runtime, the LOOP operation executes one pass of the loop for each member in the specified array. You specify the name of the input array in the *in-array* field on the LOOP operation's *Properties* tab.

When you design your flow, bear in mind that the services within the loop operate against individual elements in the specified input array; they must therefore be designed to take *elements* of the array as input, not the entire array.

If one iteration of your loop produces an output variable that you want to collect, you can use the *out-array* field on the LOOP operation's *Properties* tab to specify the name of the variable. At runtime, the server will automatically create an array (a record list) variable that contains the output of each iteration of the loop.

Most of the time, the IDoc segment you will want to loop over and the resulting record list you will want to collect with the loop are embedded in a deeply nested tree-like hierarchy of the actual source and target pipeline variables. It will be a lot easier to create some temporary, top-level pipeline variables to hold the record list you want to loop over and the resulting list to be produced. The typical steps for performing an effective loop are as follows:

1. Create the necessary record definitions, describing both the structure of the source record and the structure of the target record, for one loop iteration. You can copy and paste the record elements from the existing record definitions of the actual source and target variables.
2. Prepare the pipeline for the loop. Create a top-level Pipeline Out variable as a record reference list, and reference the source record definition you just created. Map the original embedded record list that you want to loop over to this new top-level variable.
3. Add the LOOP operation and specify the temporary top-level source variable as the in-array.
4. Add the loop body. Within the loop body, add a new Pipeline Out variable as a record reference, and reference the target record definition. Add the necessary operations in order to map from the source record data to the target record data within the loop body.
5. Specify the target variable name as the out-array for the loop. After the loop, all the target records produced during the loop iterations will be collected into an array with the same name.
6. Clean up the pipeline and assign the out-array variable to the embedded element of the actual target variable in the pipeline.

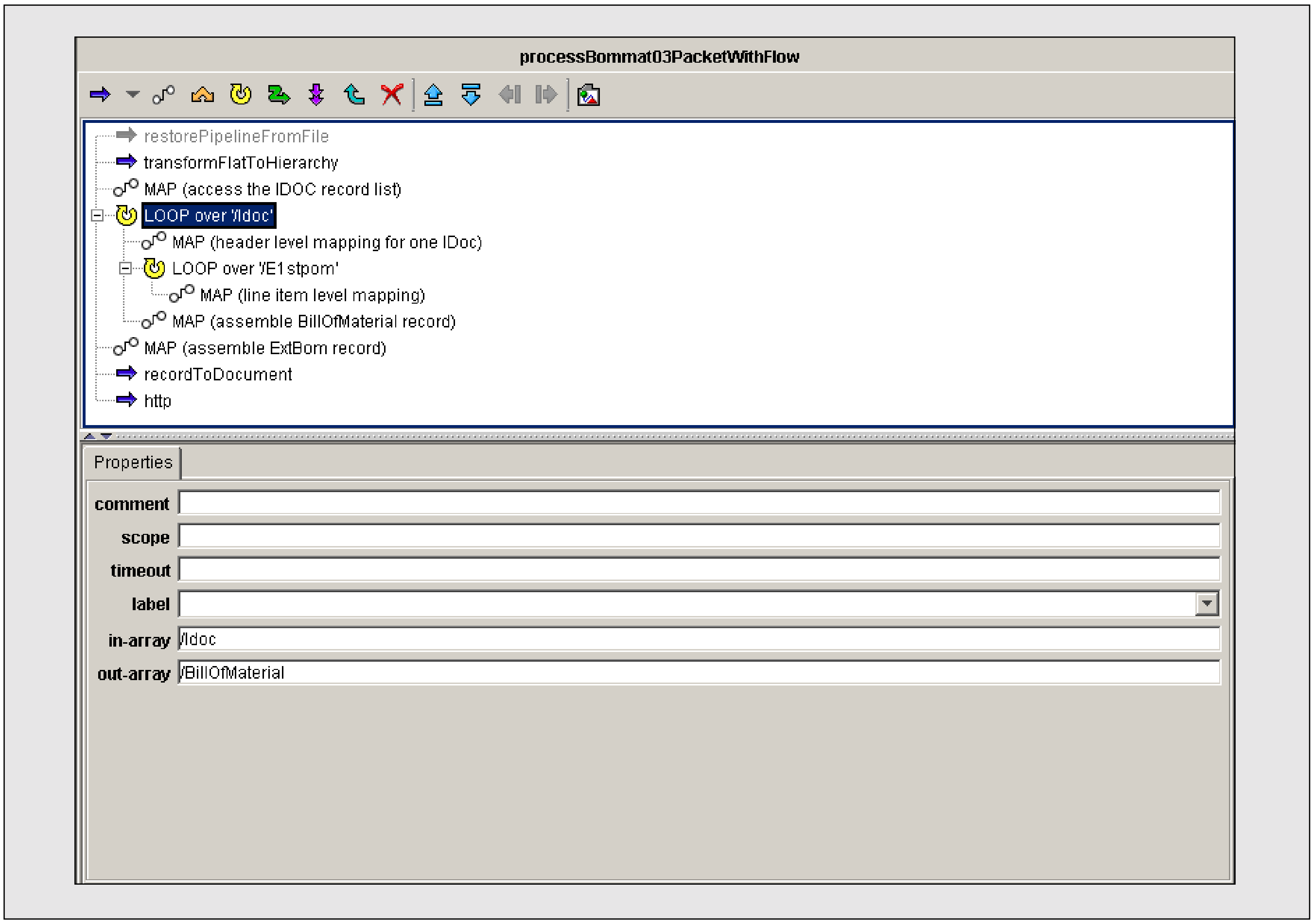
The example flow service included with the download, *spjexamples.idoc.outbound: processBommat03PacketWithFlow,* uses this technique to loop over multiple IDocs and recurring IDoc segments.

Handling the IDoc Packet or Recurring IDoc Segments

When performing mapping with a BC flow service, you must also take into account that the SAP system can be configured to send either a single IDoc to the BC or multiple IDocs together in one package (i.e., an IDoc packet). Because this cannot be predicted in advance, the *pub.sap.idoc:transformFlatToHierarchy* service always produces the *IDOC* element of the resulting *boundNode* as a record list.

If you know that only one individual IDoc will be

Figure 8 Use a Temporary Top-Level Variable to Simplify Looping


processBommat03PacketWithFlow

```
restorePipelineFromFile
transformFlatToHierarchy
MAP (access the IDOC record list)
LOOP over '/Idoc'
    MAP (header level mapping for one IDoc)
    LOOP over '/E1stpom'
        MAP (line item level mapping)
    MAP (assemble BillOfMaterial record)
MAP (assemble ExtBom record)
recordToDocument
http
```

Properties

comment	
scope	
timeout	
label	
in-array	/Idoc
out-array	/BillOfMaterial

received, you can access that one IDoc in the record list by specifying an index value of *0* in the *Link Properties* dialog box (refer back to Figure 6) when you map from the IDoc data.

If you know that multiple IDocs will be received in a package, or if you are not sure, you need to use a LOOP operation to loop through all the occurrences of the IDoc. In the first loop iteration, the first instance of the IDoc will be available for processing in the pipeline; in the second loop iteration, the second IDoc instance will be available for processing in the pipeline; and so on, until all IDoc instances in the package are processed.

To make the loop easier, you can use a temporary top-level variable (e.g., *Idoc*) as the alias for the embedded *IDOC* variable (see **Figure 8**). So, later in the flow — e.g., when specifying the in-array of the loop — you can simply refer to the IDoc record list by *Idoc* instead of by the lengthy multi-level name (e.g., *Bommat03/BOMMAT03/IDOC*[21]). This will also make the loop body much cleaner.

Within each loop iteration, a single IDoc instance is processed. Typically, this processing will produce a results record (e.g., *BillOfMaterial*). You can indicate that you want to collect all the results records into a record list as the output of the LOOP operation by specifying the record name in the out-array property of the LOOP operation.

[21] The BC notation for elements embedded in a variable. *Bommat03/BOMMAT03/IDOC* means the *IDOC* element in the *BOMMAT03* element that is embedded in the *Bommat03* pipeline variable.

In the IDoc type definition, a segment can be defined either as occurring once and only once, or occurring multiple times. If a segment can occur only once, it will be transformed into a record by the *pub.sap.idoc:transformFlatToHierarchy* service. Otherwise, it will be transformed into a record list.

To access a particular instance of an IDoc segment for which there may exist multiple instances, you access the particular record from the record list. Again, if you know the index value of the record that you want to access, you can specify the index value in the *Link Properties* dialog box for the link in your MAP flow step (e.g., if you know for sure that the segment will appear only once at runtime, even if the IDoc type definition specifies that it occur multiple times, you can access this single instance directly by specifying an index value of *0* when mapping from the segment data).

If you want to access only a particular instance of the IDoc segment that matches certain criteria (e.g., with the qualifier field equaling a certain value), you can specify the MAP condition (e.g., the *Copy only if…* expression) in the *Link Properties* dialog box.

If you need to process all the occurrences of an IDoc segment, you can use a LOOP operation to loop through all of the records in the record list of the IDoc segment and process one record — i.e., one segment occurrence — at a time within the loop iteration. The technique discussed earlier for LOOP operations can also be applied here to simplify the process of looping over the multiple occurrences of the IDoc segment.

Generating XML from the Mapping Results

In the flow service, you map the IDoc data to a target record format. However, most of the time an XML document is needed as the end result of the transformation. The BC built-in service *pub.web:recordToDocument* can be used to convert the record variable to an XML string.

The service recursively traverses through the given record variable, building an XML string from the values. Record keys are turned into XML tag names, and record values are turned into the tag bodies. Key names starting with the *attribute* prefix (i.e., @) are turned into attributes of the parent XML tag.

After converting the mapping results to the target XML document, you can transmit the document to the desired destination (e.g., your business partner's system). This will be discussed in the upcoming section "Post the XML Document to the External System."

Performing Value Lookups

Quite often, you need to perform value lookups during the mapping. For example, your business partner may be using a different set of codes for material numbers, item categories, etc.

There are a few options available for performing value lookups in the flow service.

The *pub.string:lookupTable* Service

You can pre-populate the input parameter *lookupTable* (which has the type string table) with name/value pairs. When you invoke the service *pub.string:lookupTable* with the lookup key, it will return the lookup results in the output parameter *value* of type string.

This option is suitable for a lookup with a small number of possible entries; it would be very expensive for lookups with thousands of possible entries. Also, keep in mind that changing the lookup table requires a modification to the flow service itself.

A Simple Text File

You can put the name/value pairs of the lookup entries in a simple text file, using the popular Java properties file format, as follows:

Listing 2: Using a Static Data Member to Cache Lookup Data

```
// use static data member to cache the lookup data
private static Properties lookupProp;

// use static initializer to pre-load the lookup data from the file
static {
  try {
    String lookupFilename =
      "c:/sapbc46/Server/packages/SPJ/resources/SbcLookupFile.txt";
    FileInputStream fis = new FileInputStream(new File(lookupFilename));
    lookupProp = new Properties();
    lookupProp.load(fis);
    fis.close();
  } catch (Exception ex) {
    ex.printStackTrace();
  }
}
```

Listing 3: Executing a Lookup Using a Java Properties Object

```
// get pipeline input
IDataCursor pipelineCursor = pipeline.getCursor();
String key = IDataUtil.getString( pipelineCursor, "key" );

// perform the lookup using the pre-loaded lookupProp
String value = lookupProp.getProperty(key, "not found");

// pipeline output
IDataUtil.put( pipelineCursor, "value", value );
pipelineCursor.destroy();
```

```
name1=value1
name2=value2
…
```

Then you can write a simple Java service, taking the lookup key as an input argument, performing the lookup using the file, and returning the results as the output argument. To avoid unnecessarily reading the file from the file system multiple times, you can use the static Java *initializer* construct, putting the code in **Listing 2** into the *Shared Source* section of the Java service.

The code in Listing 2 will be executed only once at the service startup. The properties file will be read, and its contents will be loaded into the *lookupProp* static data member. The actual Java lookup service is very simple, as you can see in **Listing 3**.

This sample Java service has one input argument that is the lookup key. The input argument is retrieved from the input pipeline. It then performs the lookup with the *getProperty()* method of the preloaded *lookupProp* properties object, specifying the lookup key and the default value *not found* as arguments. Finally, the lookup results are returned to the output pipeline in the *value* argument.

By using a file to store the lookup contents, you can easily change the lookup entries without modifying your service. This approach is suitable for lookups with hundreds of possible entries.

Database Tables

You can alternatively store the lookup entries in the table of a relational database and then create a BC service to access the database table and perform the lookup. You will need to maintain a relational database and make sure that the lookup table is populated with the lookup entries. You will also need to find a suitable JDBC driver for the database and make it accessible to the BC.[22]

Since the BC provides built-in database access to flow services, you can actually create a flow service that accesses the database lookup table without any coding by following these steps:

1. **Define a database alias (data source) for the database.** Open your browser and access the BC Administration screen (the location is typically *http://<sbchost>:5555*). Select *Adapters* → *Database* → *Alias management* → *Add* and specify the alias (e.g., *SbcLookup*); the database URL (e.g., *jdbc:odbc:SbcLookup*); the database user name and password (optional); and the database driver (e.g., *sun.jdbc.odbc.JdbcOdbcDriver*). Click *Submit* to add the data source to the list.

[22] Please refer to the chapter "Configuring Access to Database Systems" in the *SAP Business Connector Administration Guide* for details on setting up the JDBC driver. The guide can be found in the *doc* folder of your BC server installation.

2. **Create a flow service for the database lookup.** Select *Database* → *Service Generation*, choose the data source, and specify the package, folder name, service name, and ACL group for the flow service. Click *Generate from SQL* and type in the SQL statement for the lookup, which will look something like the following:

```
select value from SbcLookupTable
  where key =?
```

   Click on the *Evaluate* button and specify the parameter name (e.g., *key*) and type (e.g., *VARCHAR*). Click on the *Generate* button, and a new flow service will be automatically generated.

3. **Modify the generated service in the BC Developer.** Open the service in the BC Developer. You may need to lock the service (right-click on the service and select *Lock*) before you can make any changes to it. In the generated flow service, the results of the SQL query are stored in the output argument *results*, of type record list. If the SQL query found the lookup results, it should be the only record in the *results* record list. If the lookup did not find any matching results, *results* will be empty. To make using the lookup service easier, you can add a MAP operation after the SQL query step to return the lookup results if there are any, and the default value if no matching results were found.

Alternatively, you can write a simple Java service that uses the JDBC API and SQL statements to access the database lookup table directly. I will discuss this in more detail in the next article ("XML Messaging with the SAP Business Connector Part 2: Outbound IDoc-to-XML Data Mapping with XSLT and Java Services, and Inbound XML-to-IDoc Data Mapping") when I discuss Java extension functions for XSLT.

Obviously, performing lookups with a database table is the most scalable approach and has the best performance results for lookups with thousands of possible entries.

✓ ***Tip***

No matter which approach you use to build the lookup service, you can always invoke it as a transformer from the main flow service.

An Example BC Flow Service

A flow service called *spjexamples.idoc.outbound: processBommat03PacketWithFlow* is provided with the downloadable *SPJ* package (available at **www.SAPpro.com**) as an example of IDoc outbound processing with flow. The flow service is designed to illustrate all the important techniques discussed in this article, and can be used as a reference for building your own IDoc processing flow service. Here, I will briefly walk you through the example.

The example flow service will map the *BOMMAT03* IDoc packet received from the SAP system to an external XML document, as described by the DTD *ExtBom.dtd*. After the mapping, it will transmit the resulting XML document to an external HTTP server listening at *http://localhost:4242*.

In the beginning of the flow, the service *pub.sap.idoc:transformFlatToHierarchy* is invoked to transform the IDocs from a flat structure to a hierarchical structure. Please note that in the Pipeline Out, we bind the record definition *spjexamples.records:Bommat03Rec* to the *boundNode* Service Out variable.

In the next MAP flow step, we first define a temporary top-level record list variable called *Idoc* as a record reference list to the record definition *spjexamples.records:Bommat03IdocRec* (created by copying and pasting the relevant portion from *spjexamples.records:Bommat03Rec*), and then link the embedded *Bommat03/BOMMAT03/IDOC* element to it.

In the following outer loop, we loop over the *Idoc* record list and collect *BillOfMaterial* as an out-array.

In the MAP operation at the beginning of the outer loop, we define a Pipeline Out variable called *BillOfMaterial* as a record reference to *spjexamples.records:BillOfMaterialRec* (created by copying and pasting the relevant portion from *spjexamples.records:ExtBomRec*). We then map the header-level data from the record variable *Idoc* to the record variable *BillOfMaterial*. A transformer is used to invoke the *pub.data:dateTimeFormat* built-in service to convert the date format for field *BillOfMaterial/ValidFromDate*. To prepare for the inner loop over the line items, we also define a top-level variable *E1sptom* as a record reference list to *spjexamples.records:E1stpomRec* (created by copying and pasting the relevant portion from *spjexamples.records:Bommat03IdocRec*), and link the embedded *Idoc/E1STZUM/E1STPOM* element to it.

In the inner loop that follows, we loop over the *E1stpom* record list and collect *Component* as an out-array.

In the MAP step within the inner loop, we first define a new Pipeline Out variable called *Component* as a record reference to *spjexamples.records:ComponentRec* (created by copying and pasting the relevant portion from *spjexamples.records:BillOfMaterialRec*), and then perform the mapping from the record variable *E1stpom* to the record variable *Component*.

Another transformer is used here to invoke the custom service *spjexamples.util:lookupStringTable* to perform string-table-based value lookup for the *Component/ItemCategory* field. You can replace this service with *spjexamples.util:lookupFile* for file-based value lookup, or *spjexamples.util:lookupDB* for database-table-based value lookup. The Java service *spjexamples.util:lookupFile* reads the file *SbcLookupFile.txt* in the *resource* directory of the *SPJ* package. This file contains the lookup entries. To use *spjexamples.util:lookupDB*, you need to define a database alias with the name *SbcLookup*. The file *SbcLookup.mdb* in the *resource* directory of the *SPJ* package is a Microsoft Access Database file, which contains one table called *SbcLookupTable*. It can be used for database-table-based value lookup. To use it, you first need to define an ODBC data source pointing to the MDB file; after that you need to specify the JDBC-ODBC bridge as the JDBC driver when defining the database alias.

In the next MAP operation, which is a child of the outer loop, we copy the *Component* record list (an out-array of the inner loop) to the embedded element *BillOfMaterial/Components/Component*.

In the following top-level MAP operation, we copy the *BillOfMaterial* record list (an out-array of the outer loop) to the embedded element *ExtBom/BillOfMaterials*.

Then the built-in service *pub.web:recordToDocument* is invoked to convert the results record *ExtBom* into an XML document.

Finally, the XML document is transmitted to the business partner's system via HTTP POST using the *pub.client:http* built-in service.[23]

The *PerlServer.exe* file can be used as a test HTTP server. By default, it will listen at TCP port *4242* and dump all the HTTP request data received to the console.

To send *BOMMAT* IDocs from the SAP system, you need to configure the ALE distribution using transaction *SALE*. After the ALE configuration is complete, you can send the IDoc by using transaction *BD30*.

[23] How to transmit the XML document to a business partner's system will be discussed in detail in the next section "Post the XML Document to the External System."

You need to configure the routing rule to route the *BOMMAT* IDoc message to the BC service by specifying the following (refer back to Figure 3):

- **Transport:** *B2B Service*
- **Folder:** *spjexamples.idoc.outbound*
- **Service:** *processBommat03PacketWithFlow*

Remember to enable the routing rule after saving it!

Alternatively, a captured pipeline containing a sample IDoc packet is provided as file *Bommat03PacketPipeline.xml*. You can use *restorePipelineFromFile* as the first step in the flow to restore the pipeline from it, thus simulating the receipt of the IDoc packet from the SAP system.

Post the XML Document to the External System

So, now that you've converted your SAP IDoc to the desired XML format, how do you get it from the Business Connector to your business partner?

There are two ways to send XML data out from the BC:

- Use the XML transport in the routing rule for the outgoing IDoc (as described earlier in the discussion of outbound communication with IDoc-XML).
- Build a BC service that invokes the *pub.client:http* service to post the XML document to an arbitrary target URL.

The XML transport can be used only if the business partner is able to accept the SAP IDoc-XML document directly as it is.

If a different XML vocabulary is expected, you will need to use a BC service to map the data to the desired format, as I just described.[24] After that, you can use the built-in service *pub.client:http* to HTTP POST the resulting XML document to the target URL by specifying the following argument values:

- **url:** The target URL, which usually begins with *http:* or *https:*
- **method:** The HTTP method you want to use, which is typically *POST*
- **data/string:** The mapped XML document that needs to be transmitted
- **headers:** Any HTTP request headers you wish to specify, like specifying *Content-type* and *text/xml* as a name/value pair, for example

Let's take a look of some of the considerations involved when posting an XML document to a business partner's system.

Guaranteed Delivery of the Communication

What happens if, at the time of HTTP transmission, the target HTTP server of the business partner is unavailable? Or the network breaks down during the HTTP transmission?

HTTP is a simple, stateless protocol; it does not provide guaranteed delivery. This shortcoming ideally should be addressed at the protocol level in the future. In the meantime, while such a protocol is not available, a simple "handshake agreement" can be implemented between the BC and the target HTTP server to safeguard the HTTP transmission.

For the outbound IDoc scenario, if the *pub.client:http* service cannot establish the connection to the target HTTP server, if the connection is broken before the transmission is complete, or if the HTTP response code indicates an error, the Business Connector will automatically feed the error status back to the SAP system. In the SAP system's tRFC monitor (transaction *SM58*), you will see these errors logged. The SAP system will automatically schedule a background job to retry the outbound IDoc transmission at a later time. So, in this way, data will not be lost due to HTTP errors.

But there is still a risk that data transmission may be duplicated. Let's say the target HTTP server successfully received the HTTP request, but before it can send back HTTP status *200*, the network connection is broken. In this case, the BC and SAP system will consider the transmission a failure and will retransmit the same data again at a later time — but we are now facing a problem of duplicate data transmission!

To avoid this kind of duplication, you can use the handshake agreement. For example, suppose in the HTTP request header you've attached a unique "transaction identifier" field. The target HTTP server application, when receiving an incoming HTTP request, will first check the transaction identifier against its log of already processed transaction identifiers. If the transaction identifier indicates that it has been received and processed before (which means the transmission is a duplicate), the HTTP server application will simply send back HTTP status *200* without further processing. If the transaction identifier is a new one, the HTTP server application can log it and process it normally.

On the SAP BC side, you need to make sure that the same transaction identifier will be used for all retransmissions. This is actually quite easy. When

[24] Up to now, I've discussed mapping with a BC flow service. I will discuss mapping with XSLT transformation and mapping with a Java service in the second article of this two-part discussion.

Helpful Hints

The following is a quick reference of the helpful hints presented throughout the discussions in this article.

IDoc-XML Messaging

- ☑ If your business partner is able to create and accept IDoc-XML documents, use this approach. It is the easiest to implement, and no BC development is needed.
- ☑ Simply set up the routing rules correctly to route the IDoc message or IDoc-XML document to its desired destination.

Outbound IDoc Communication Using Non-SAP XML

- ☑ Capture a sample IDoc for development and testing purposes, using the *savePipelineToFile* and *restorePipelineFromFile* services (or the *savePipeline* and *restorePipeline* services).

Data Mapping Using a BC Flow Service (Outbound)

- ☑ Create an IDoc record definition by loading from an IFR schema, from an XML-wrapped DTD, or by copying and pasting from a captured sample IDoc in the pipeline.
- ☑ Use the *transformFlatToHierarchy* service to convert the outbound IDoc to a record, and bind the IDoc type record definition to it to enable it for mapping.

the SAP system sends an IDoc to the BC, it uses the tRFC protocol, which means there is already a transaction ID (TID) created and used by the SAP system — the *$tid* variable in the pipeline. It will always contain the same value for all the subsequent retransmissions initiated by the SAP system for the same IDoc. You can simply propagate the *$tid* value to the target HTTP server as the transaction identifier of the HTTP transmission. By doing so, you now have a unique transaction ID protecting the entire end-to-end transmission, from the SAP system to the BC, then from the BC to the target HTTP server.

There is third-party software available that provides queuing functionality and supports guaranteed delivery over the Internet. These solutions can also be used together with the Business Connector to guarantee the transmission of the XML document. However, they are typically expensive, and using them will add another layer of overhead to the solution. Depending on your requirements, the simple handshake approach discussed here may or may not be adequate for implementing guaranteed delivery in your own outbound IDoc-based XML communication project. You make the call!

- ☑ Create temporary top-level pipeline variables in the flow for easy looping. Create the necessary record definitions for these temporary top-level variables by copying and pasting the relevant portions from the existing record definitions.
- ☑ Use transformers in the MAP operation to perform value transformation. Any existing services, including your own custom services, can be used as transformers.

Performing Value Lookup

- ☑ The options available for performing value lookups include the string-table-based, text-file-based, and database-table-based approaches.
- ☑ When using text-file-based lookup, avoid unnecessary reading of the file by using a static initializer.
- ☑ Database-table-based lookup is the most scalable approach to value lookup with good performance results.

Guaranteed Delivery (Outbound)

- ☑ For outbound transmission, the SAP system will automatically retransmit the IDoc at a later time in case of error.
- ☑ To implement a simple handshake agreement with the business partner's system for tracking and guaranteed delivery, the TID of the tRFC call from the SAP system to the BC can be used as a transaction identifier for the outgoing HTTP transmission.

Sending XML Documents to Multiple Targets

Sometimes it is necessary to send the same transformed XML document to multiple target systems. At other times an IDoc needs to be transformed into different XML documents and sent to different target systems. These tasks can be easily accomplished using BC services.

For the former case, you first need to perform the necessary transformation. After that, you can use multiple flow steps, each invoking the *pub.client:http* service with the unique URL for one target system. After all the flow steps are executed, the transformed XML document is then transmitted to all the target systems.

For the latter case, you can create several flow services, each performing one unique transformation. Then, from the main flow (the BC service specified in the routing rule), you can invoke all the transformation flows one-by-one, each followed by the invocation to *pub.client:http* to post the results to the appropriate target system.

Conclusion

The SAP Business Connector is the current tool of choice for enabling SAP components to participate in XML messaging over the Internet. Mostly, this involves the use of IDocs.

Here, I discussed when to use the IDoc-XML format for direct communication and when to develop a BC service to map the IDoc to an external XML format, including how to use a custom flow service to perform the mapping. You can also perform the mapping using XSLT transformation or a Java service. In the article that follows ("XML Messaging with the SAP Business Connector Part 2: Outbound IDoc-to-XML Data Mapping with XSLT and Java Services, and Inbound XML-to-IDoc Data Mapping"), I will explore these additional mapping options and also outline the special considerations involved in inbound IDoc communication using non-SAP XML.

By trying the sample solutions provided in the download, and using what you have learned in the discussion, my hope is that you will be able to jumpstart your own XML messaging projects.

Robert Chu joined SAP at the end of 1996. He currently works for the Integration and Certification Center at SAP Labs in Palo Alto, California. Prior to this, he worked in technical consulting and training at SAP America and SAP Asia. Robert's current focus is the SAP integration technologies. He has been regularly teaching classes in this area at SAP training centers and is the main author of the BIT531 training course, as well as a few other internal workshops. Robert has spoken at the past three SAP TechEd events. In addition to his SAP expertise, he is also an SCEA (Sun Certified Enterprise Architect) for J2EE, an MCSD (Microsoft Certified Solution Developer), and an MCSE (Microsoft Certified System Engineer). Robert can be reached at robert.chu@sap.com.