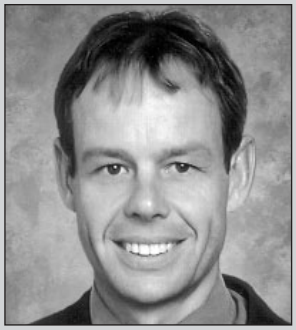
XML-Enable Your ALE Distributed Business Processes for Simplified, Standardized Data Exchange with Non-SAP Systems via HTTP

Arthur Wirthensohn



Arthur Wirthensohn is a senior consultant at EDS Switzerland and a member of EDS's international Technical Leadership Network. Currently, he is a project manager and technical lead of the Enterprise Application Integration (EAI) department, which delivers SAP-related services such as ALE/EDI, SAP Business Workflow, Web Application Server, Data Migration, and ABAP Programming.

(complete bio appears on page 76)

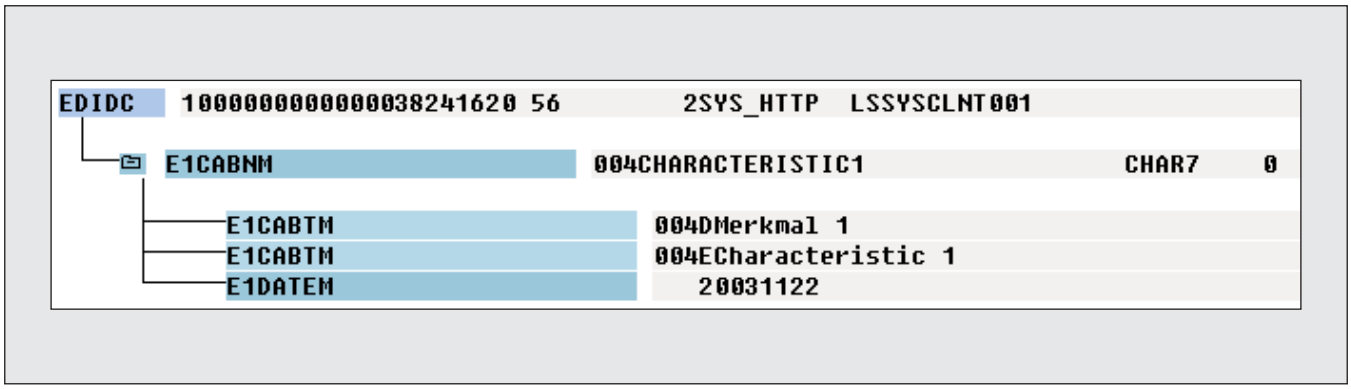
When you set out to build an ALE distributed business process, you have a choice of communication technologies to use in linking your distributed systems and exchanging the Intermediate Documents (IDocs) containing the message data to be distributed. You can use a transactional remote function call (tRFC), a file, an ABAP interface, or an HTTP connection. tRFC is usually the preferred technology. Its transaction logic ensures reliable interfaces, and many useful ALE features and functions are available only with tRFC (e.g., the ability to distribute ALE customizing, retry erroneous communications, and trace IDocs).

tRFC is an SAP proprietary technology, however, and the IDoc format used in tRFC distributions is another SAP proprietary technology. That means you can only use tRFC for distributions to SAP systems, or systems that support the RFC and IDoc technologies, such as some SAP partner products that have implemented the RFC library. If you need to send data to a partner with a non-SAP system that has no RFC or IDoc support, you have to utilize an additional application integration system, like the SAP Business Connector or SAP Exchange Infrastructure (SAP XI), or some kind of file interface, and transmit the data with a communication protocol provided by your application server,[1] like FTP.

There is good news, though. As of SAP Web Application Server (SAP Web AS) 6.20, new features in ALE allow you to send and receive messages with a standardized communication protocol (HTTP) and a standardized message data format (XML).

[1] In this context, an "application server" refers to the server itself, its operating system, and the tools running on that server that can be used by other applications, like an SAP system.

Figure 1 Example of an IDoc Structure



Before you adopt this standards-based approach as a silver bullet for your ALE data distribution needs, however, you must have a solid understanding of some key differences between using HTTP/XML and using tRFC/IDoc. If you are an application consultant or project manager, you will find that not all of the features available with tRFC/IDoc are available with HTTP/XML, and if you are a technical consultant or system administrator, you will find that the setup for HTTP/XML is a little different from tRFC/IDoc. In addition, there are certain types of distributions that may not be good candidates for native SAP Web AS HTTP/XML, such as those that require sophisticated data transformations and advanced integration features. This article acquaints you with the capabilities and limitations of ALE distributions via HTTP/XML, shows you how to set one up step by step, and explains what you need to know about handling potential distribution errors.

SAP ALE with XML — A Brief History

SAP introduced XML features into SAP Basis systems comparatively late. And until SAP Web AS 6.20, if you wanted to HTTP/XML-enable the distributed business processes of your SAP systems,[2] you had to employ an application integration system like the SAP Business Connector or SAP XI. The next sections provide a brief overview of SAP ALE support for XML.

The XML and XML File Port Types

SAP introduced XML into ALE in Release 4.6 with a new port type[3] called "XML," which enables IDocs to be exchanged as XML files. The XML port type has the same technical properties as the basic File port type, except that the resulting file has an XML structure instead of a standard IDoc structure. **Figure 1** shows an example of a message with an IDoc structure; **Figure 2** shows the same message in XML format.

Like the File port type, the XML port type writes to the file system of the SAP application server, and an executable program (e.g., a shell script) can be triggered (with the name and path of the XML file) to further process the XML file (e.g., to pass it to a partner system via FTP). The call of the executable program is done through an SAP executable program in the RFC library (*rfcexec* on Unix systems and *rfcexec.exe* on Microsoft Windows systems). SAP also delivers a standard program (function module *IDOC_XML_FROM_FILE*) for inbound processing that imports an XML file from the file system, converts it into IDoc format, and passes it to the ALE layer.

[2] Systems based on SAP Basis Releases 3.1 through 6.10.

[3] In ALE, the available communication channels are called "port types." A port type describes the technical properties of the ALE communication.

Figure 2 *Example of an XML Structure*

```
<?xml version="1.0" encoding="UTF-8" ?>
- <CHRMAS04>
  - <IDOC BEGIN="1">
    - <EDI_DC40 SEGMENT="1">
        <TABNAM>EDI_DC40</TABNAM>
        <MANDT>100</MANDT>
        <DOCNUM>0000000000038239</DOCNUM>
        <DOCREL>620</DOCREL>
        <STATUS>30</STATUS>
        <DIRECT>1</DIRECT>
        <OUTMOD>2</OUTMOD>
        <IDOCTYP>CHRMAS04</IDOCTYP>
        <MESTYP>CHRMAS</MESTYP>
        <SNDPOR>SYSSND</SNDPOR>
        <SNDPRT>LS</SNDPRT>
        <SNDPRN>SNDCLNT100</SNDPRN>
        <RCVPOR>SYS_HTTP</RCVPOR>
        <RCVPRT>LS</RCVPRT>
        <RCVPRN>RCVCLNT001</RCVPRN>
        <CREDAT>20031122</CREDAT>
        <CRETIM>212437</CRETIM>
        <SERIAL>20031122212437</SERIAL>
      </EDI_DC40>
    - <E1CABNM SEGMENT="1">
        <MSGFN>004</MSGFN>
        <ATNAM>CHARACTERISTIC1</ATNAM>
        <ATFOR>CHAR</ATFOR>
        <ANZST>7</ANZST>
        <ANZDZ>0</ANZDZ>
        <ATEIN>X</ATEIN>
        <ATDIM>0</ATDIM>
        <ATDEX>0</ATDEX>
        <ANAME>CAREMOTE</ANAME>
        <ADATU>20030312</ADATU>
        <VNAME>CAREMOTE</VNAME>
        <VDATU>20030312</VDATU>
        <ATMST>1</ATMST>
        <DATUV>00000000</DATUV>
      - <E1CABTM SEGMENT="1">
          <MSGFN>004</MSGFN>
          <SPRAS>D</SPRAS>
          <ATBEZ>Merkmal 1</ATBEZ>
          <DATUV>00000000</DATUV>
          <SPRAS_ISO>DE</SPRAS_ISO>
        </E1CABTM>
      - <E1CABTM SEGMENT="1">
          <MSGFN>004</MSGFN>
          <SPRAS>E</SPRAS>
          <ATBEZ>Characteristic 1</ATBEZ>
          <DATUV>00000000</DATUV>
          <SPRAS_ISO>EN</SPRAS_ISO>
        </E1CABTM>
```

In Release 4.6, the format of an XML version of an IDoc is specified in a document type definition (DTD) for that message. With the XML port type, you have the option of inserting the DTD at the beginning of the XML file by setting the DTD flag in the port definition (transaction *WE21*), which enables you to send all possible segments and fields that might occur in the actual message.

> ✓ ***Note!***
>
> *The structure of the XML elements specified in the DTD, which specify all possible elements of the XML document, corresponds to the structure of the segments and fields of the IDoc — for example, an IDoc segment is represented as a node in the XML structure.*
>
> *You can create a DTD for an IDoc with transaction WE60 (Documentation for IDoc Types) by entering the IDoc name in the Object type field and selecting Documentation → Create DTD.*

One drawback of the 4.6 XML port type is that you can't create XML data in UTF-8 format,[4] which is a format that can handle special characters that are not part of the standard English character set (such as ä, for example). With port type XML, special characters must be replaced with substitute characters defined in conversion tables in the port maintenance transaction (*WE21*). Of course, the 4.6 XML port type also lacks support for HTTP communication, a technology that enables you to send information almost anywhere; with port type XML, you can only handle data on a local file system.

SAP Web AS 6.10 renames the port type from "XML" to "XML File." The XML File port type retains all the characteristics of the XML port type, and adds a choice of two XML formats:

- *SAP Release 46* — This option creates XML files in the Release 4.6 XML port type format.
- *Unicode* — This option creates XML files in UTF-8 format, so you no longer need to substitute non-standard English characters in transaction *WE21*.

The XML HTTP Port Type

SAP Web AS 6.20 introduces an additional port type called "XML HTTP" that supports HTTP as a communication protocol (note that the XML File port type is still supported as well). With port type XML HTTP, an IDoc can be sent as an XML message in UTF-8 format to a partner system via HTTP.

In the outbound process, the sender system acts as an HTTP client, sending the XML message as an HTTP request to the receiver system. The receiver system acts as an HTTP server and processes the request. The HTTP destination can be the URL of an SAP request handler or the URL of another HTTP address that is able to process an HTTP request and receive the XML message (e.g., a service of the SAP Business Connector).

The inbound processing of an XML message via HTTP takes place through an SAP request handler that is able to receive an XML data stream in UTF-8 format. This handler is defined in a service of the SAP Internet Communication Framework (ICF), an integrated component of SAP Web AS that handles HTTP requests.[5] The handler converts the XML data stream into the IDoc format and passes it to the ALE layer.

With these inbound and outbound communication features, SAP Web AS is well-equipped to handle

[4] Unicode Transformation Format-8 (UTF-8) is an efficient Unicode data format where one character is encoded as 8-bit integers (octets). According to the UTF-8 definition (ISO 10646), one character is encoded in sequences of 1 to 6 octets. For example, standard US-ASCII characters are represented as 1 octet, while other European characters (e.g., ö) are represented as 2 octets.

[5] For more information on the ICF, go to *SAP Web Application Server → Middleware (BC-MID) → Components of SAP Communication Technology → Communication Between ABAP and Non-ABAP Technologies → Internet Communication Framework* in the SAP Library.

distributed business processes in which data is sent and received in XML form.

Caveats to Using XML for ALE Data Distributions

There are certain scenarios in which an HTTP/XML distribution may not be optimal, in particular, the following:

- ☑ **Complex data conversions:** Because XML allows you to send messages using a standardized format, your business partners can use standardized tools to handle the messages. This does not mean, however, that a partner system will necessarily understand a message's content. Creating meaningful message content requires an agreement between partners on the formatting of that content. If both partners agree to an IDoc/XML format, no data conversions will be necessary. However, if both partners decide on another XML format, such as xCBL (XML Common Business Library) or XBRL (Extensible Business Reporting Language), then most likely a data conversion will need to be applied to the message content. Such conversions are typically too complex for the data transformation features of ALE, so another tool for XML data conversions (like the SAP Business Connector or SAP XI) will be required.

- ☑ **Transactional communication:** HTTP does not provide tRFC features like transactional control or repeat delivery in case of communication failure. If these features are required, you will need an application integration tool that communicates with the SAP system through tRFC/IDoc.

- ☑ **Increased processing needs:** Keep in mind that HTTP/XML processing requires slightly more system capacity than tRFC/IDoc processing, because incoming messages have to be converted from XML to IDoc format, and outbound messages have to be converted from IDoc to XML format before they are transmitted. An XML message is also larger than an IDoc — and therefore requires more bandwidth — because XML describes its content with tags. An IDoc just lists the content of the fields of one segment; the segment definition (transaction *WE31*) describes the order and length of the segment's fields.[6]

Setting Up an Example Distribution That Uses HTTP/XML

To familiarize you with using HTTP/XML for your ALE data distributions, in the following sections we will set up an example distribution that uses the XML HTTP port type. To follow along with the steps, you will need two SAP R/3 Enterprise (SAP R/3 4.7) systems that can communicate with each other via an HTTP port.[7] SAP R/3 Enterprise systems are required in order to support the message type we will use in the example, the material master message type MATMAS (the implementation of a distributed business process with the MATMAS message type is described under *SAP Web Application Server → Middleware (BC-MID) → Application Link Enabling (BC-MID-ALE) → ALE Quick Start* in the SAP Library[8]). Alternatively, if you are familiar with ALE, you can use a different message type, such as CHRMAS (characteristics of master data), as long as it is part of the standard SAP Web AS installation (note that CHRMAS is included in SAP Web AS 6.20 and higher systems).

Though the reason for using HTTP/XML is most often a distribution that involves a non-SAP system or interface, it is not practical to use a

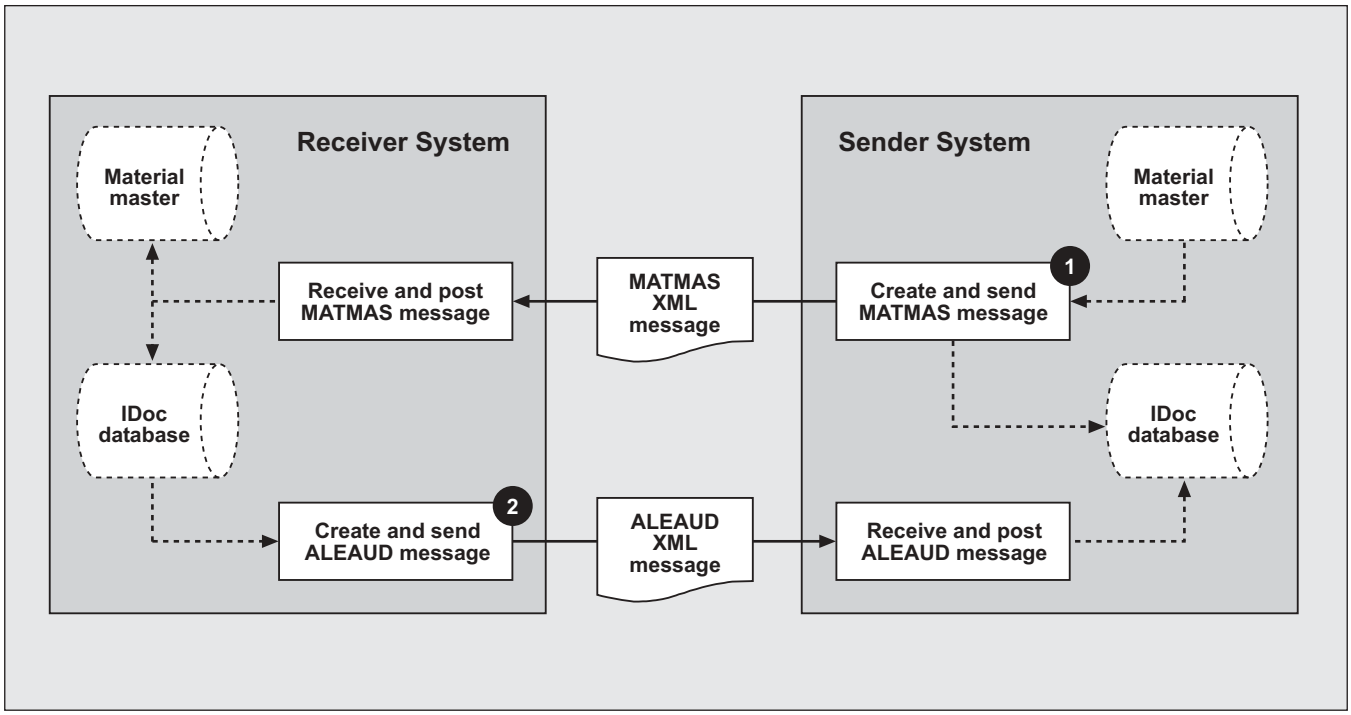
---

[6] See the *SAP Professional Journal* article series "XML Messaging with the SAP Business Connector Part 1: Direct IDoc-XML Data Exchange and Outbound IDoc-to-XML Data Mapping with Flow Services" and "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" (July/August 2003).

[7] The port that will be used for HTTP communication will be defined in the profile settings (more on this in the upcoming section "Step 1: Configure the HTTP Protocol").

[8] The ALE Quick Start section of the SAP Library provides an overview of how to set up a standard distributed business process with material master data and tRFC as the communication technology.

Figure 3 *The Sample Scenario*



> ✓ ***Note!***
>
> *In the example scenario, each system will act as both a sender and a receiver system, so you have to configure the HTTP protocol and define the ICF service and RFC destination on both sides.*

non-SAP system in this example, as the instructions are different for every type of system. For your own implementations, you will need to configure the distribution as appropriate for the non-SAP side.

The first part of the example scenario — distributing a material master message — is similar to the scenario described in the ALE Quick Start section of the SAP Library. But instead of using tRFC as the communication technology and IDoc as the message format, we will use HTTP as the communication technology and XML as the message format. The second part of the scenario involves sending an ALE audit message to cope with some shortcomings of HTTP/XML compared to tRFC/IDoc. We will go deeper into that later on in the article.

The diagram in **Figure 3** illustrates our example scenario. In the first subprocess (labeled ❶ in the diagram), a material master message (MATMAS) is created and sent to the receiver system (and, of course, posted to the IDoc database on the sender system). The receiver system then posts the IDoc data to its own IDoc database, creates a material master, and documents the posting status (whether it was successful or not) of the material master creation in its IDoc database.

In the second subprocess (❷), an audit message (ALEAUD) is created on the system that receives the material master message and is sent to the system that initially created the material master message. The ALEAUD message is stored in the receiver system's IDoc database, and also updates the status (whether the material master was successfully posted on the receiver side or not) of the originating material master message in the sender system's IDoc database. The content of the ALEAUD message depends on the posting status of the MATMAS message.

There are five steps involved in setting up the example distribution scenario:

1. Configure the HTTP protocol on the sender and receiver sides.
2. Define an ICF service with an event handler that will receive the XML message and pass it to the ALE layer. An ICF service defines the linkage between the SAP Web AS communication layer — the Internet Communication Manager (ICM)[9] — and applications in the ABAP stack.
3. Define an RFC destination that addresses the event handler of the ICF service on the receiver system.
4. Configure the ALE distributed business process for exchanging the master data.
5. Test the distribution scenario.

Over the next sections we will walk through each of these five steps. Once these steps are complete, I will explain in detail an error-processing issue with HTTP/XML communication, which we circumvent in the example with an audit message, so when you set off to implement your own HTTP/XML distribution you won't be caught off guard by unexpected errors. But before we get into the details of maintaining an active distribution, we have to build one to maintain, and the first step is to configure the HTTP protocol.

Step 1: Configure the HTTP Protocol

SAP Web AS can act as both an HTTP server and an HTTP client. It can send HTTP requests (acting as an HTTP client) and process incoming HTTP requests (acting as an HTTP server). We will use this functionality for the technical communication of our example distributed business process.

The ICM is the component — a better term might be "process" — responsible for HTTP communication. The ICM supports the various protocols (e.g., HTTP, HTTPS, and SMTP) that can be configured in the system profile. For our example business process, we will configure HTTP. Of course, you could enhance the security of the transmission using HTTPS, which is HTTP protected by the Secure Sockets Layer (SSL) protocol. Securing transmissions with HTTPS involves installing the SAP Cryptographic Library, configuring SSL in the system profile, and customizing the SAP Trust Manager, which are beyond the scope of this article.[10]

Configuring the HTTP port consists of the following tasks:

1. Determine whether the ICM is running and HTTP is active.
2. Adjust the HTTP parameter settings as necessary.
3. Verify that the HTTP parameter settings are active.

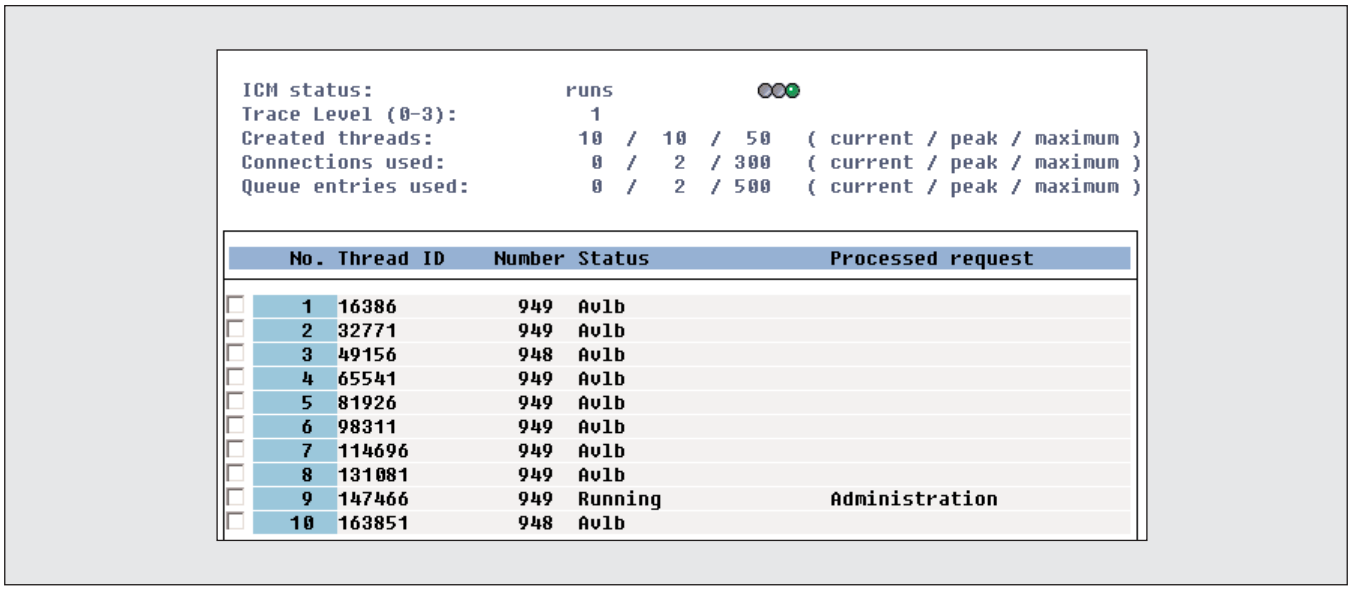
Task #1: Determine Whether the ICM Is Running and HTTP Is Active

The first step in configuring HTTP is to go to the ICM Monitor (transaction *SMICM*) and perform the following checks on your SAP Web AS:

[9] For more information on the ICM, go to *SAP Web Application Server → Client/Server Technology (BC-CST) → Architecture of the SAP Web Application Server → SAP Web Application Server Components → Internet Communication Manager (ICM)* in the SAP Library.

[10] For further information on how to configure SSL, please refer to SAP Note 510007 and the *SAP Web Application Server Security* and *Trust Manager* sections (under *SAP Web Application Server → Security (BC-SEC)* in the SAP Library).

Figure 4 The ICM Monitor Overview Shows the Status of the ICM



☑ **Verify that the ICM is running.** If the ICM is running, it has the status "runs" in the ICM Monitor overview screen (see **Figure 4**). If the ICM is not running, you will probably have to restart it in transaction *SMICM* or implement parameter *rdisp/start_icman* in the system profile.

☑ **Verify that HTTP is active.** In the ICM Monitor overview screen, click on the *Services* button or use the keyboard combination *Shift+F1*. The HTTP service status should be "active" and the port number of the service should be the one defined in parameter *icm/server_port_<x>* (more on this in Task #2), which should be clearly shown on the screen. If HTTP is not active, I recommend reading the following section on HTTP parameters and then making adjustments to the system profile.

Task #2: Adjust the HTTP Parameter Settings As Necessary

There are quite a few ICM parameters[11] in the system profile settings that control HTTP communication. We will focus on the following parameters, which are the ones that are essential to getting the HTTP service running:

- *icm/plugin_<x>* — This parameter, which is part of the instance profile settings, defines the location and name of the plug-in that handles HTTP requests. The placeholder <*x*> must be incremental from 0. For example, use *icm/plugin_0* for the HTTP plug-in, *icm/plugin_1* for the HTTPS plug-in, and *icm/plugin_2* for the SMTP plug-in.

> ***Caution!***
>
> *Inappropriate changes to the system profile might result in a system that cannot be restarted or that behaves incorrectly. If you are not a system administrator and you want to change the ICM profile parameters, the best course is to inform the system administrator of the changes you want to make. As the person responsible for this part of the system, the administrator will in most cases apply the parameter settings for you rather than allow you to apply them yourself.*

[11] For a good explanation of ICM parameters, go to *SAP Web Application Server → Client/Server Technology (BC-CST) → Architecture of the SAP Web Application Server → Parameterizing the ICM and the ICM Server Cache* in the SAP Library.

Figure 5 Sample Instance Profile Settings

```
ICM Parameter

Plugins

icm/plugin_0          = PROT=HTTP, PLG=.\httpplugin.dll
icm/plugin_1          = PROT=HTTPS, PLG=.\httpplugin.dll
icm/plugin_2          = PROT=SMTP,PLG=.\smtpplugin.dll

Services

icm/server_port_0     = PROT=HTTP,PORT=8000
icm/server_port_1     = PROT=HTTPS, PORT=443
icm/server_port_2     = PROT=SMTP, PORT=25025
```

The plug-in for HTTP is *httpplugin.dll* on Microsoft Windows systems and *httpplugin.so* on Unix systems. It resides in the executables directory */usr/sap/<SID>/SYS/exe/run* on Unix (where *<SID>* is the system ID) or *<D>:\usr\sap\<SID>\SYS\exe\run* on Microsoft Windows (where *<D>* is the drive and *<SID>* is the system ID). See **Figure 5** for an example of the instance profile settings in a Microsoft Windows environment (please note that the directory path .\ in Figure 5 represents the executables directory).

In my experience with SAP Web AS 6.20, if you do not specify the *icm/plugin_<x>* parameter, the system assumes that *icm/plugin_0* defines the standard HTTP plug-in that resides in the executables directory. This means that you only have to set this parameter if you want to change the default and use a different HTTP plug-in. However, because this behavior is not documented, and I do not know if it is true for all SAP Web AS 6.20 support packages or for higher SAP Web AS releases, I recommend setting the *icm/plugin_<x>* parameter manually in the system profile.

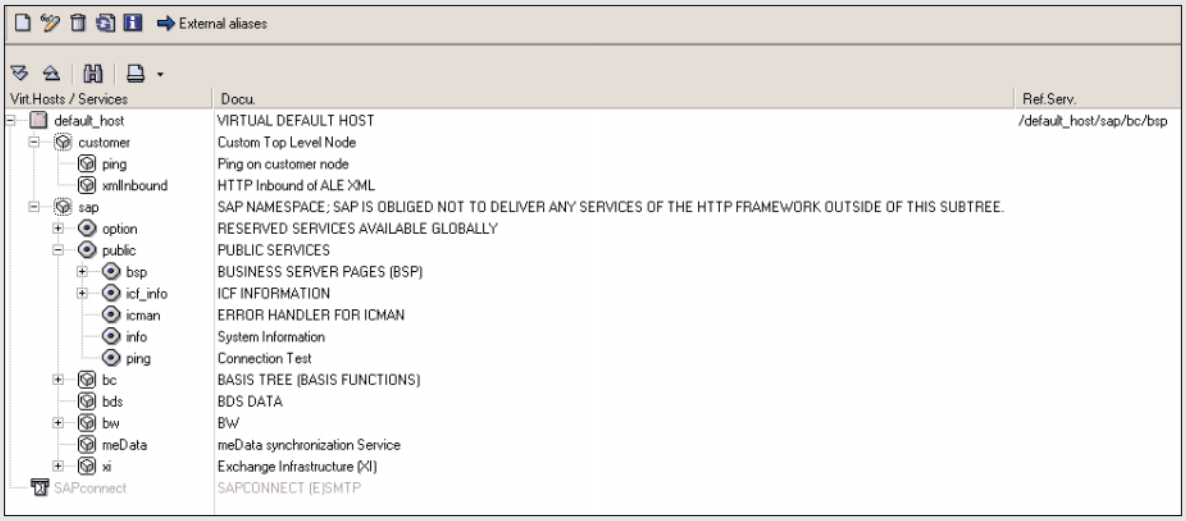
- *icm/server_port_<x>* — This parameter, which is also part of the instance profile settings, defines the port and some optional properties for the protocol. For example, the parameter value *PROT=HTTP, PORT=8000* defines port 8000 for protocol HTTP. The placeholder *<x>* must correspond to the same value defined in parameter *icm/plugin_<x>* for the same protocol. For example, if parameter *icm/plugin_0* specifies the plug-in for HTTP, then the HTTP port must be defined in parameter *icm/server_port_0*, as you see in Figure 5.

> ✓ ***Note!***
>
> *The port that you define for the ICM in the profile settings must not be used by another service or program. Otherwise, the service for the protocol cannot be started.*

- *rdisp/start_icman* — The ICM starts if this parameter value is set to *true* in the instance profile settings; if the value is *false*, the ICM does not start. By default the parameter value is *true*.

- *ms/http_port* — This parameter, which is part of the default profile settings, specifies the port on which the SAP message server accepts HTTP requests. If the SAP Web Dispatcher is installed, this port forwards HTTP (or HTTPS) requests coming from the Internet to the defined port.

Figure 6 Overview of ICF Services



> ✓ ***Note!***
>
> *Whether or not the SAP Web Dispatcher is installed, the port specified in the ms/http_port parameter must be different from the HTTP port specified in parameter icm/server_port_<x>.*

Goto → Parameters → Display. If you are not familiar with the ICM, and you have problems evaluating the settings, see your system administrator.

Once HTTP is active and the parameters are verified, you are ready to go to the next step and implement an ICF service that will process inbound XML messages.

Task #3: Verify That the HTTP Parameter Settings Are Active

To activate your HTTP parameter settings, you must restart the system. To verify that the HTTP parameter settings are active, go to the ICM Monitor (transaction *SMICM*) and check the following:

- ☑ **The ICM is still running, and HTTP is active.** These are the same checks you performed in Task #1.
- ☑ **The relevant ICM parameters are set correctly.** You can get an overview of these parameters by going to the ICM Monitor menu and navigating to

Step 2: Define an ICF Service

In SAP Web AS, the ICF provides an environment for handling HTTP requests; it also provides tools — like debugging and tracing — for analyzing the requests. You can access this environment via transaction *SICF* (HTTP Service Hierarchy Maintenance).

In transaction *SICF*, you get an overview of ICF services in a hierarchical tree-like structure (see **Figure 6**). Each service controls its corresponding request handler (or handlers). The tree structure also represents the URL for the services to be called. In other words, *default_host* represents the root of the

URL (*<server>:<port>*), and the branches of the tree structure represent the URL paths of the services. For example, the URL of service *default_host* → *customer* → *ping* is *<server>:<port>/customer/ping*.

There are three important things to keep in mind about this service structure:

- First, SAP delivers all of its standard services within the node *sap*. Do not create your own services under the *sap* node, because they could be overwritten by SAP (via support packages, upgrades, etc.).
- Second, all of the services under the node *public* do not require a logon to the system. They are processed with the SAP internal user *sapsys*. For security reasons, you should handle these public services with care.
- Third, all services are initially deactivated. A service is not available for an HTTP request until you activate it. You should activate only the services that you need, for the following reasons:
  - You do not want to publish services that you do not know about. If you don't know a service's functions and properties, you can't know what it does and what it allows on your system.
  - There may be improperly defined services (e.g., services with anonymous logon data) that could open the door to malicious intruders.

To define the service for our example, we need to complete the following tasks:

1. Create a new node.
2. Create a service for handling inbound XML messages.
3. Define the request handler.
4. Activate the service.
5. Create a service for testing.
6. Test the new services.

> ✓ ***Note!***
>
> *We will actually create two new services — one for the distribution itself and one for testing purposes.*

Task #1: Create a New Node

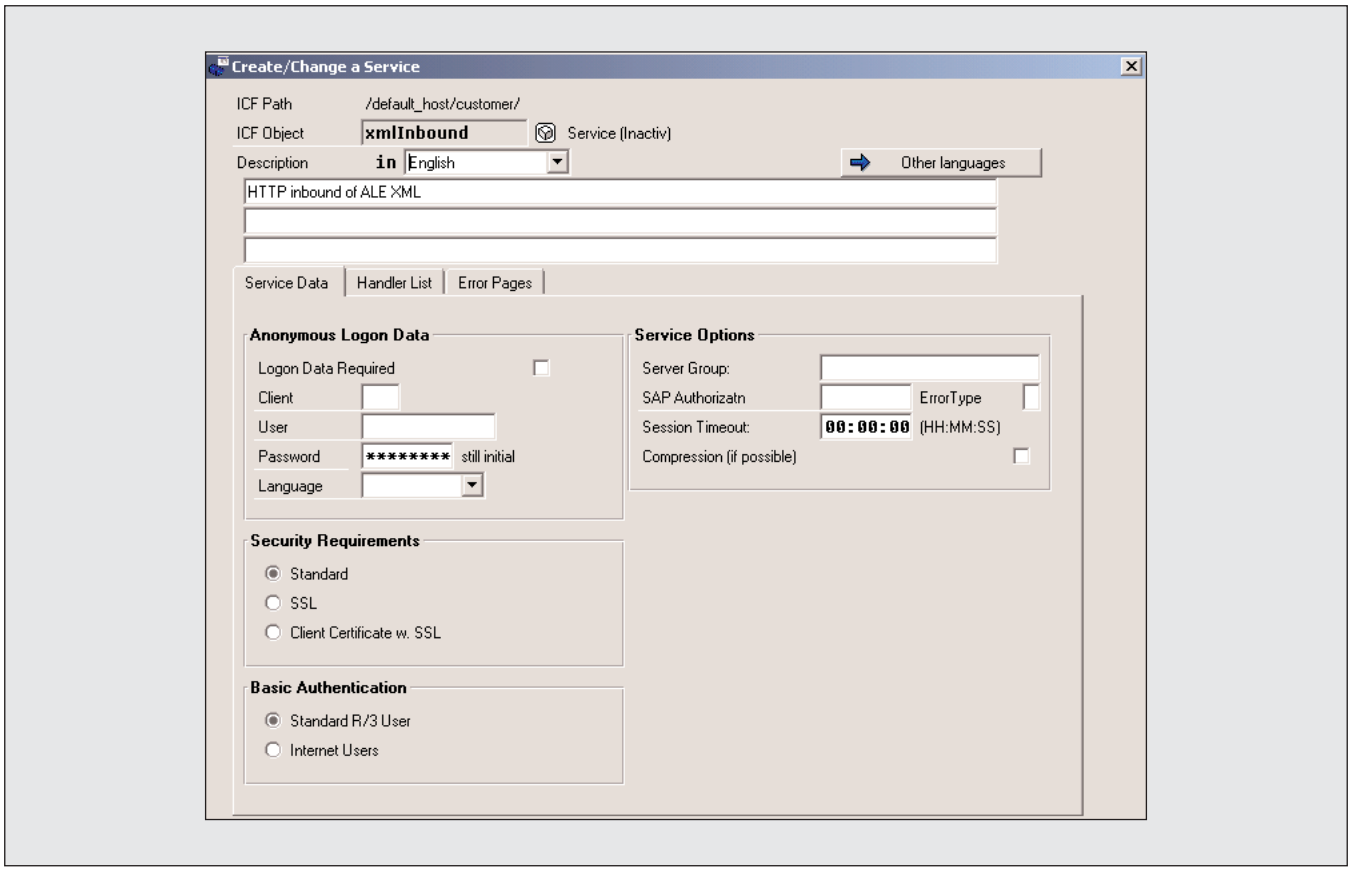
The first step in creating the ICF service for our example distribution is to create a new node to serve as the parent hierarchy element for our two new services.

To create the new parent hierarchy node, click on the *New* icon ( ) in transaction *SICF*. In the pop-up window that appears, give the service a name (e.g., *customer*) and select the *Independent Service* radio button. You will be shown a screen where you can define the properties of the service.

Because we want to define a node simply for the purpose of grouping our two new services, the node does not require any other properties. Just add a description (e.g., *Custom Top Level Node*), and save the service. Remember that the service is initially deactivated.

> ✓ ***Note!***
>
> *When creating a new top-level node like "customer," you must notify the ICM that there is another top-level node (i.e., in addition to the "sap" node and any other top-level nodes you have created); otherwise, the ICM cannot forward HTTP requests to services belonging to that node. So, after you have created the new top-level node, call the ICM Monitor (transaction SMICM) and navigate to Goto → HTTP Application Server → Load URL prefixes. The message bar at the bottom of the SAPGUI will indicate that the URL for the new service has been loaded with the message "Operation successfully executed."*

Figure 7 ICF Service Definition Screen



Task #2: Create a Service for Handling Inbound XML Messages

To create the service for handling inbound XML messages, place the cursor on the newly created node. This marks the new starting point. Now click on the *New* icon ( ), name the service (e.g., *xmlInbound*), select the *Independent Service* radio button, and press *ENTER*. A screen in which you can define the service pops up (see **Figure 7**).

As with the parent node (*customer* in the example), you must add a description for this service (e.g., *HTTP inbound of ALE XML*). It is not necessary to specify any of the service data on the *Service Data* tab of the service definition, but for your reference we'll take a quick look at the available settings.

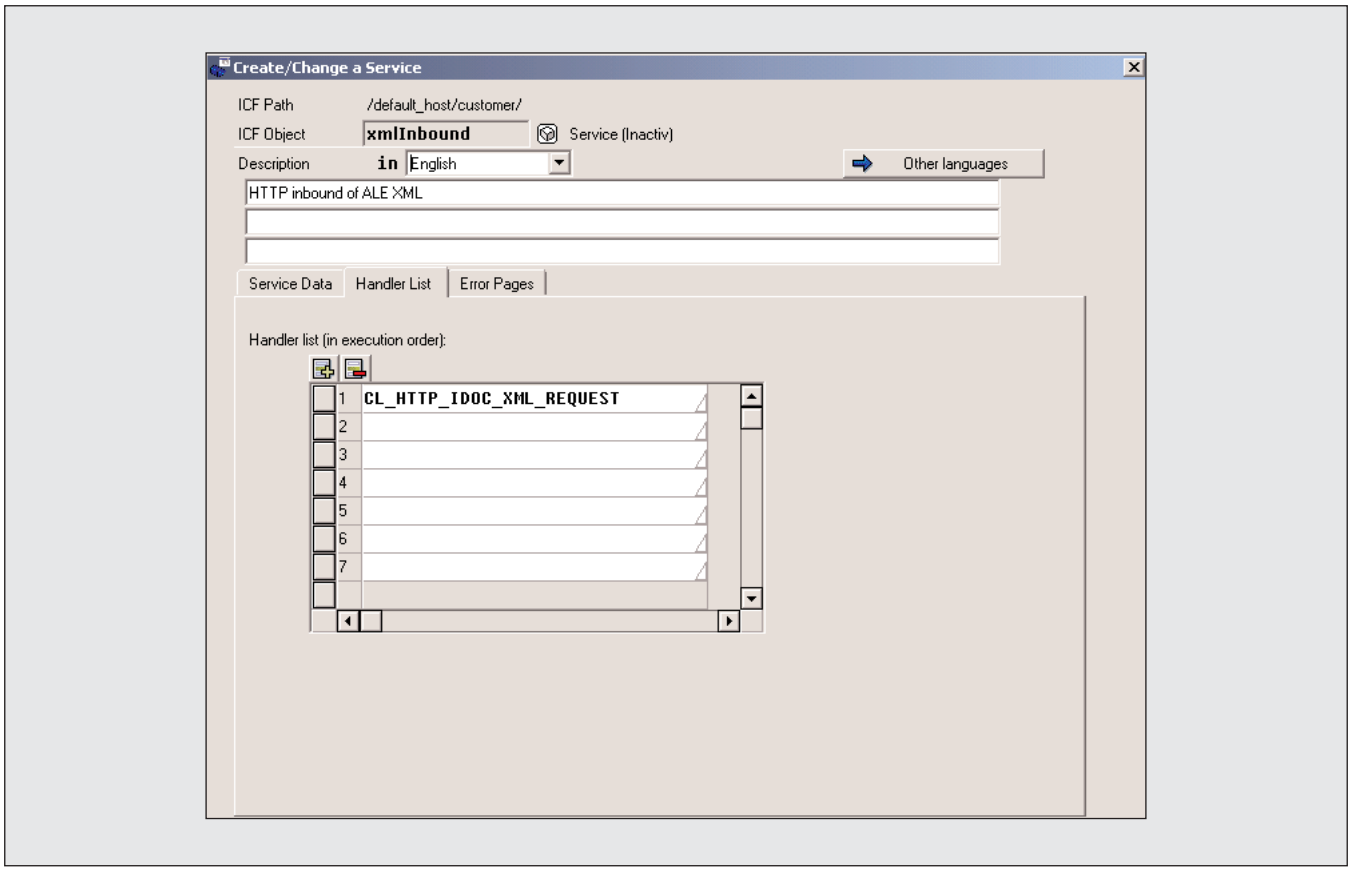
In the *Anonymous Logon Data* field group on this screen, you could specify data that would log on to the SAP system when the service is called. Beware of doing so, however, because you could circumvent the system's authorization mechanism and create a security issue.

In the *Security Requirements* field group, you can control security features like HTTPS. In our sample scenario we want to use HTTP, so we leave the default selection *Standard*.

The *Basic Authentication* field group defines whether the logon will be validated against a standard SAP user (a transaction *SU01* user) or an Internet user (a transaction *SU05* user). Only an SAP user setting will allow SAP authentication using roles, so we leave this default setting untouched.

The *Service Options* field group is where you can specify miscellaneous options of the service. For example, you can define a server group where the

Figure 8 Handler Definition for the Example ICF Service



service can be processed. You can also opt to use the compression feature, which is helpful when the service generates an HTML response with a considerable amount of data — e.g., an SAP BW Business Explorer (BEx) Web report. In our case, the compression feature doesn't make much sense; we will be sending the payload data in the HTTP request, and the HTTP response will be very small.

Task #3: Define the Request Handler

The *Handler List* tab of the service definition dialog is where we define the request handler. As shown in **Figure 8**, insert *CL_HTTP_IDOC_XML_REQUEST* as the handler name. This is the standard request handler delivered by SAP for processing IDoc XML data. This handler converts the inbound XML data to IDoc format, passes the IDoc to the ALE layer, and sends a response back to the sender of the HTTP request. As its name implies, *CL_HTTP_IDOC_XML_REQUEST* is an ABAP class. Each request handler that can be used in an ICF service is an ABAP class that implements the *HANDLE_REQUEST* method of the *IF_HTTP_EXTENSION* interface. The implementation of the method, which is called by the ICF, provides the processing logic of the handler and creates the HTTP response.

For instance, *CL_HTTP_IDOC_XML_REQUEST* checks the incoming data (for UTF-8 compliance, for example), calls a Business Add-In (BAdI) to remove any envelopes,[12] starts the XML/IDoc inbound

[12] The XML data in an inbound HTTP/XML communication could be wrapped in an envelope containing control and processing information, which cannot be handled by an SAP request handler. An HTTP request with XML data could be sent with a SOAP envelope, for example, or with a Microsoft BizTalk envelope. The BAdI *IDOC_XML_ENVELOPE_IN* can be used to remove such envelopes for the SAP request handler.

process, and creates an HTML response message. If there are no processing errors, it issues the message *IDoc-XML-inbound ok*. If the UTF-8 check or the XML/IDoc inbound process returns errors, it issues an error message and initiates a transactional rollback.

The available features on the *Error Pages* tab, such as the ability to define a redirect to another URL, are especially helpful for HTML applications. Since we do not handle logon errors in our scenario, we do not need to make any entries on this tab for our example service definition.

Save the new ICF service. Note that the activation status of the service (just to the right of the service name in Figure 8) is still inactive. The next step is to activate it.

Task #4: Activate the Service

Close the service and right-click on the service name in the ICF service hierarchy (Figure 6). In the list of possible actions, choose *Activate Service*. A pop-up dialog window will prompt you to confirm the service. Clicking on the *Yes* button implicitly activates all services that are a parent to this service, so the *customer* service node we set up earlier to contain this new service is also activated.

> ✓ ***Note!***
>
> *If you deactivate a service that has child services, all of those child services will be deactivated. For example, if you were to deactivate the customer service, the xmlInbound service would be deactivated as well.*

Task #5: Create a Service for Testing

SAP provides a public ping service (under the *public* node, as shown in Figure 6), but as mentioned earlier, public services do not require a logon. We can use the public ping service to test whether we can send a request to our ICF service and get a response, but we will also need to create our own ping service to test whether our service can successfully log on to the SAP system. Our custom ping service will be called by an HTTP client and respond with an HTTP processing code that indicates whether the communication was successful or not.[13]

To create the custom ping service, repeat the previous tasks, except this time name the service *ping*, describe it as *Ping with Logon*, and insert *CL_HTTP_EXT_PING* into the handler list. Save the newly created service and activate it.

Task #6: Test the New Services

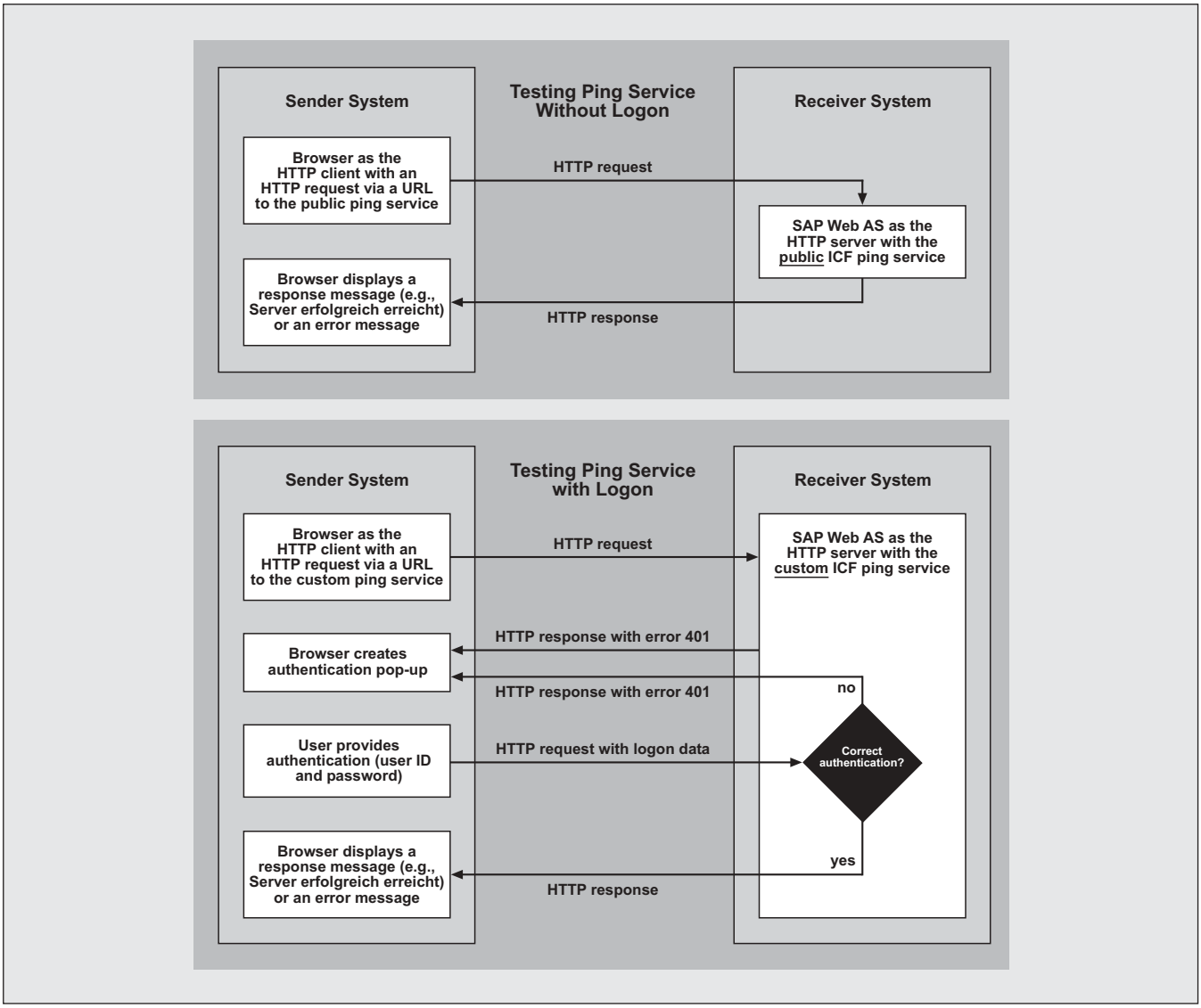
We're ready to test our newly created services. **Figure 9** shows the two test scenarios we want to perform.

First we want to test whether we can forward an HTTP request to and get a response from the new ICF service we created to handle inbound XML messages. For this purpose, we can use the public ping service that is located under the *public* node of the ICF service hierarchy (see Figure 6). In Figure 9, this is the test labeled "Testing Ping Service Without Logon." To perform this test, the public ping service must be activated. You can check its status by looking at the service name under the *public* node (the text of an inactive service is grayed out). You can also double-click on the service and look at its activation status on the service definition screen, to the right of the service name. If the service is not activated, you must activate it.

Open your browser and enter the URL *http://<server>:<port>/sap/public/ping*, where *<server>* is the address (e.g., the IP number) of your SAP Web AS and *<port>* is the HTTP port defined in the SAP Web AS system profile.

[13] SAP only accepts return code 200 as a successful return code. The World Wide Web Consortium (W3C) defines all return codes from 200 to 299 as successful.

Figure 9 Test the Newly Created ICF Services with a Browser



If the ping is successful, the browser displays *Server erfolgreich erreicht*, which means that the server was successfully reached. The text is in German because it is hard-coded in the public ping service. If the ping is not successful, you get an error message such as *You are not authorized to view this page* (the result of HTTP error code 403).

The next test involves authenticating with the SAP system (in Figure 9, this test is labeled "Testing Ping Service with Logon"). Enter the URL of the custom ping service we created — for example, *http://<server>:<port>/customer/ping*.

The SAP system returns HTTP error code 401, which causes the browser to prompt for your user identification. Enter and submit your SAP user ID and password. If the logon is successful, the browser displays the success message *Server erfolgreich erreicht*; otherwise, you will be prompted again for your user ID and password.

If both tests are successful, then you know that the receiver system is configured properly to receive HTTP requests and forward them to a request handler. Our next task, then, is to configure the sender side.

Figure 10 Settings for a Ping Service in Transaction SM59

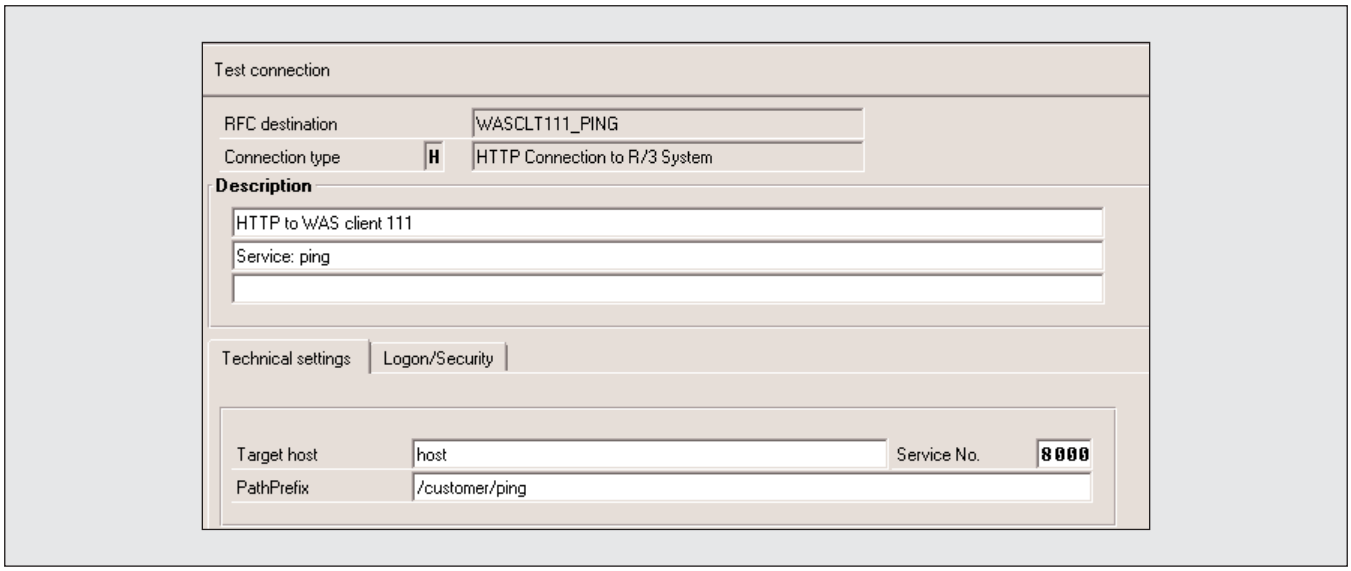
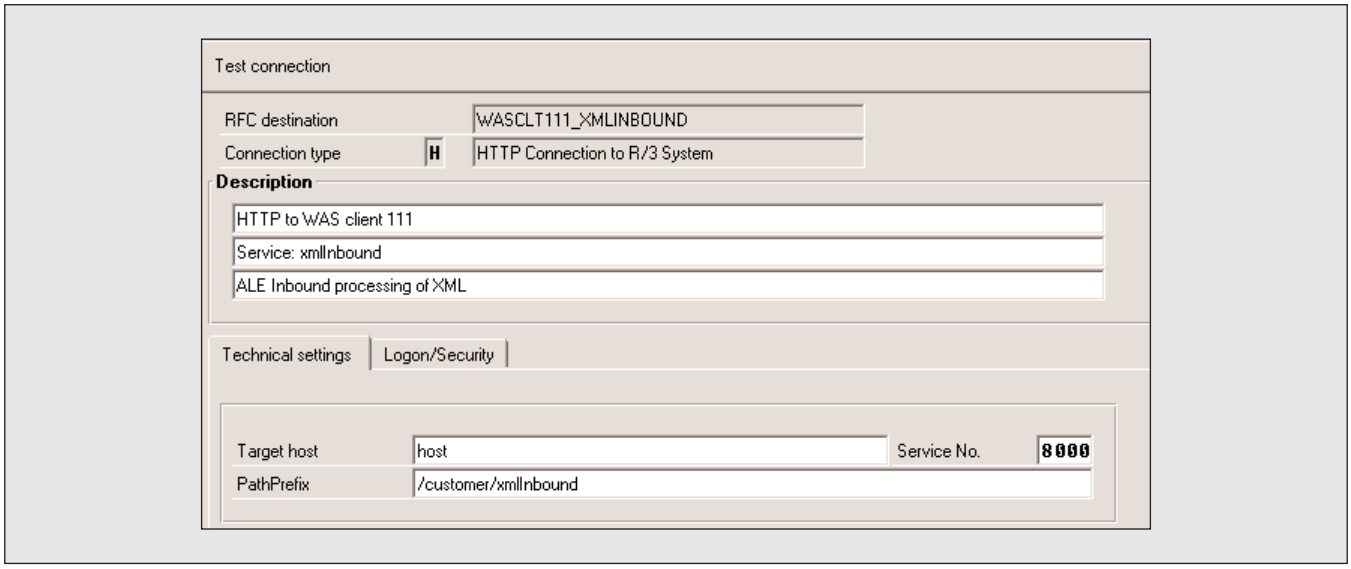


Figure 11 Settings for an Inbound ALE XML Service in Transaction SM59

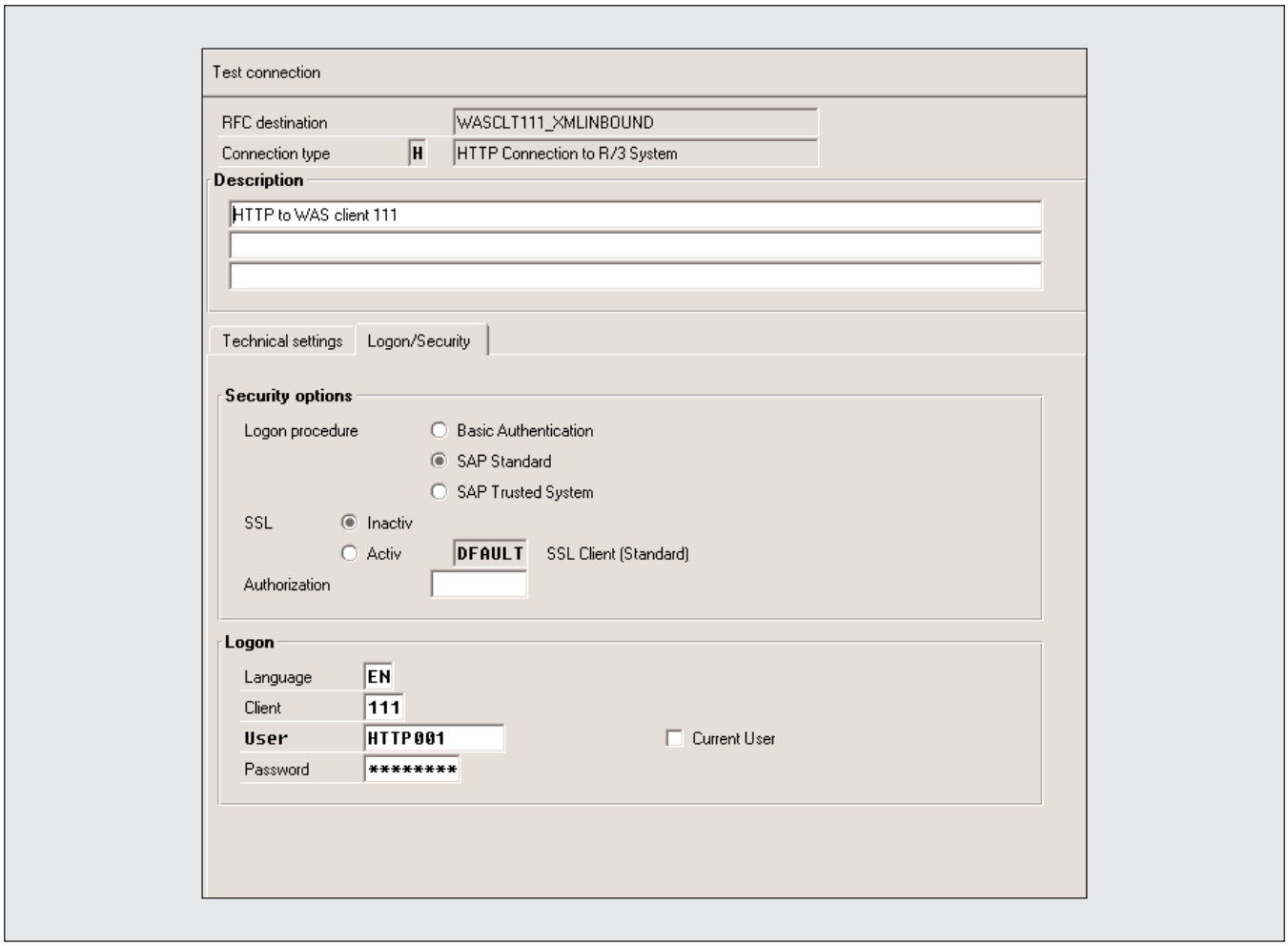


Step 3: Define an RFC Destination

To configure SAP Web AS to send an HTTP request to the HTTP server, we will use its client-side features.

Because we want to send the XML message via an HTTP request to the receiver system, we have to configure an RFC destination of type *HTTP Connection to R/3 System*. We will do this in transaction *SM59* (Display/Maintain RFC Destinations). We will also configure an RFC destination for the ping service. The term "RFC destination" is a bit strange when used in conjunction with an HTTP connection, because the destination is not a remote function call destination; it's an HTTP destination. (The reason for this terminology is historical, as you might have guessed. Before SAP Web AS 6.20, RFC connections were

Figure 12 Example Logon Settings in Transaction SM59



used for all destinations defined in transaction *SM59*.) Destinations of type *HTTP Connection to R/3 System* utilize methods of ABAP class *CL_HTTP_CLIENT* to send the HTTP request via the ICF.

Defining an RFC destination for an HTTP sender is quite simple. Call transaction *SM59* and click on the *Create* button. Give the destination a name that follows a naming convention such as *<sid>CLT<ccc>_<service>*, where:

- *<sid>* is the system ID
- *<ccc>* is the client number
- *<service>* is a reference to the service you call

So, for example, the RFC destination for our custom ping service could be *WASCLT111_PING*, as shown in **Figure 10**. For our HTTP/XML inbound service, the RFC destination could be *WASCLT111_XMLINBOUND* (see **Figure 11**).

Choose connection type *H*, which stands for *HTTP Connection to R/3 System*, add a description, and save your settings.

After saving, the screen will change and you can enter the server, port, and path to the service you want to call. See Figure 10 for an example path to the custom ping service, and Figure 11 for an example path to the HTTP/XML inbound service.

Next go to the *Logon/Security* tab. Leave the security options as they are and enter the logon information. **Figure 12** shows some example settings.

> ✓ ***Note!***
>
> *An RFC destination with connection type G (HTTP Connection to External Server) is very similar to one with connection type H, except for the logon options. With type G, you can't specify language and client parameters, and you don't have the option of using SAP-specific logon procedures like "trusted system." You would use connection type G when addressing a non-SAP Web AS system like the SAP Business Connector.*

After saving, you can test the connection. Click on *Test connection* at the upper left of the screen. For the ping service, you will get the *Server erfolgreich erreicht* message displayed as a result. For the HTTP/XML inbound service, the response is *HTTP/1.0 409 encoding="UTF-8" is missing*. Don't worry, nothing is wrong. The *xmlInbound* handler will always issue that response if the header tag of the message does not contain the string *encoding="UTF-8"*, because UTF-8 encoding is a prerequisite for that service.

> ✓ ***Remember!***
>
> *Before you can move on to the next step and set up the ALE scenario, you have to implement the systems that you want to use for the sample scenario as both sender systems (i.e., with an RFC destination) and receiver systems (i.e., with an ICF service).*

Step 4: Configure the ALE Distributed Business Process

In taking you through the setup of the example ALE distributed business process with HTTP and XML, I'm assuming that you already know how to set up a distributed business process using a traditional SAP connection. If you are not sure how to do that, please go through the ALE Quick Start lesson in the SAP Library first and refer to the corresponding literature.

> ✓ ***Note!***
>
> *The SAP Library and previous SAP Professional Journal articles are useful resources for understanding ALE distributions:*
>
> - ***SAP Library:*** *SAP Web Application Server → Middleware (BC-MID) → Application Link Enabling (BC-MID-ALE) → ALE Introduction and Administration.*
> - ***SAP Professional Journal articles:*** *"Real-Time, Outbound Interfaces to Non-R/3 Systems Made Simple with Change Pointers, Message Control, and Workflow" (Premiere Issue); "Data Transformation in SAP Standard ALE Distributed Business Processes: How to Ensure an Efficient, Effective Implementation" (July/August 2003); "Understanding and Optimizing Your ALE Data Distribution: Controlling ALE Processing" (November/December 2003); and "Understanding and Optimizing Your ALE Data Distribution: Minimizing Data Volume" (January/February 2004).*

> ✓ ***Remember!***
>
> *If you do not have two SAP R/3 Enterprise systems for testing, you will not be able to use the message type MATMAS, because it is not implemented in non-SAP R/3 systems. If you have two SAP systems available with SAP Web AS Release 6.20 or higher (e.g., SAP BW 3.0, SAP CRM 4.0, or plain SAP Web AS 6.20), however, you can use message type CHRMAS for distributing characteristics, which is available on all standard SAP Web AS systems.*

Building the HTTP/XML Distributed Process from Scratch

If you decide that you want to set up your HTTP/XML distribution without using an existing tRFC/IDoc process as a template, keep in mind that distributing a distribution model view in transaction *BD64* (Change Distribution Model) does not work with HTTP, because the message type that sends the model view (SYNCH) works only with tRFC. If you set up a distribution model with HTTP/XML and you want to distribute the model view, you have to perform the following steps:

1. Create the distribution model view in transaction *BD64*.
2. Generate the partner profile or create it manually.
3. Make sure that the receiver port in the outbound parameters of the partner profile (transaction *WE20*) for message type SYNCH is a tRFC port.
4. Make sure that the receiver port (or ports) in the outbound parameters of the partner profile for the message type (or types) of the distributed business process are XML HTTP ports.
5. Distribute the distribution model view in transaction *BD64*.

This technique for changing port type XML HTTP to a tRFC port for message type SYNCH is also helpful for other ALE features that work with a synchronous call, such as the IDoc trace in transaction *BD87* (Status Monitor for ALE Messages).

Configuring an ALE distributed business process consists of the following tasks:

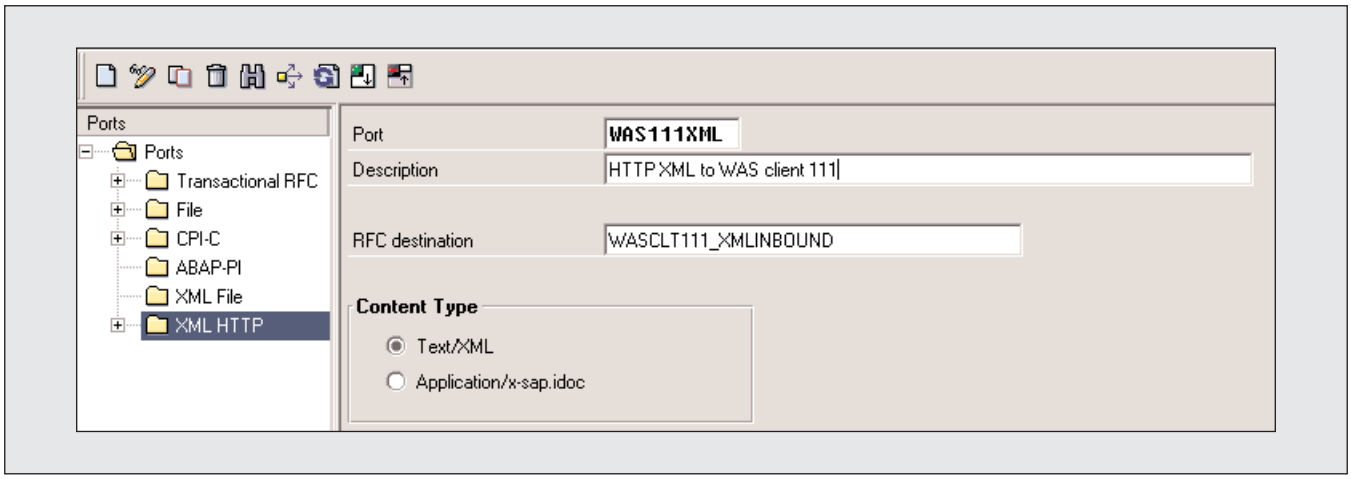
1. Build a template tRFC/IDoc distributed business process.
2. Define an ALE port of type XML HTTP.
3. Adapt the partner profile.

Task #1: Build a Template tRFC/IDoc Distributed Business Process

The first step in configuring the example ALE distributed process is to build a tRFC/IDoc distributed business process that we can use as a template for the HTTP/XML connection. Using a tRFC/IDoc template connection makes the setup easier and more convenient; however, if you are experienced in both ALE and SAP's HTTP technology, you have the option to build the HTTP/XML distribution without basing it on an existing process. If that is your choice, please refer to the sidebar above.

Set up a distributed business process with an SAP connection via tRFC, and then test it in both directions: send master data (MATMAS or CHRMAS) to the receiver system, and send an ALEAUD message back to the sender of the master data. Check the results in the IDoc monitoring transaction *WE02*. After receiving the ALEAUD message, the status of the master data message on the sender system should be 41 (*Application document created in target system*). If the status is not 41, you will have to search for errors. (See the upcoming section on error processing

Figure 13 Defining an ALE Port of Type XML HTTP



for an explanation of how ALE handles errors and the meanings of error messages you might encounter. The audit message is designed to handle a limitation of the XML HTTP port regarding error handling, as that section explains.)

> ✓ ***Tip***
>
> *To set up the ALEAUD process, insert a distribution model filter (transaction BD64) for message ALEAUD that restricts the audit messages to MATMAS (or CHRMAS) messages. You create audit messages with ABAP program RBDSTATE.*

Changing the distributed business process from a tRFC connection to an HTTP connection requires just two tasks:

- Define an ALE port of type XML HTTP.
- Adapt the partner profile.

These tasks are covered in the next two sections.

Task #2: Define an ALE Port of Type XML HTTP

We need to create an ALE port of type XML HTTP on both the sender and receiver systems, because both systems will act as sender systems (one for sending the master data message and the other for sending the audit message).

To define an ALE port of type XML HTTP, follow these steps:

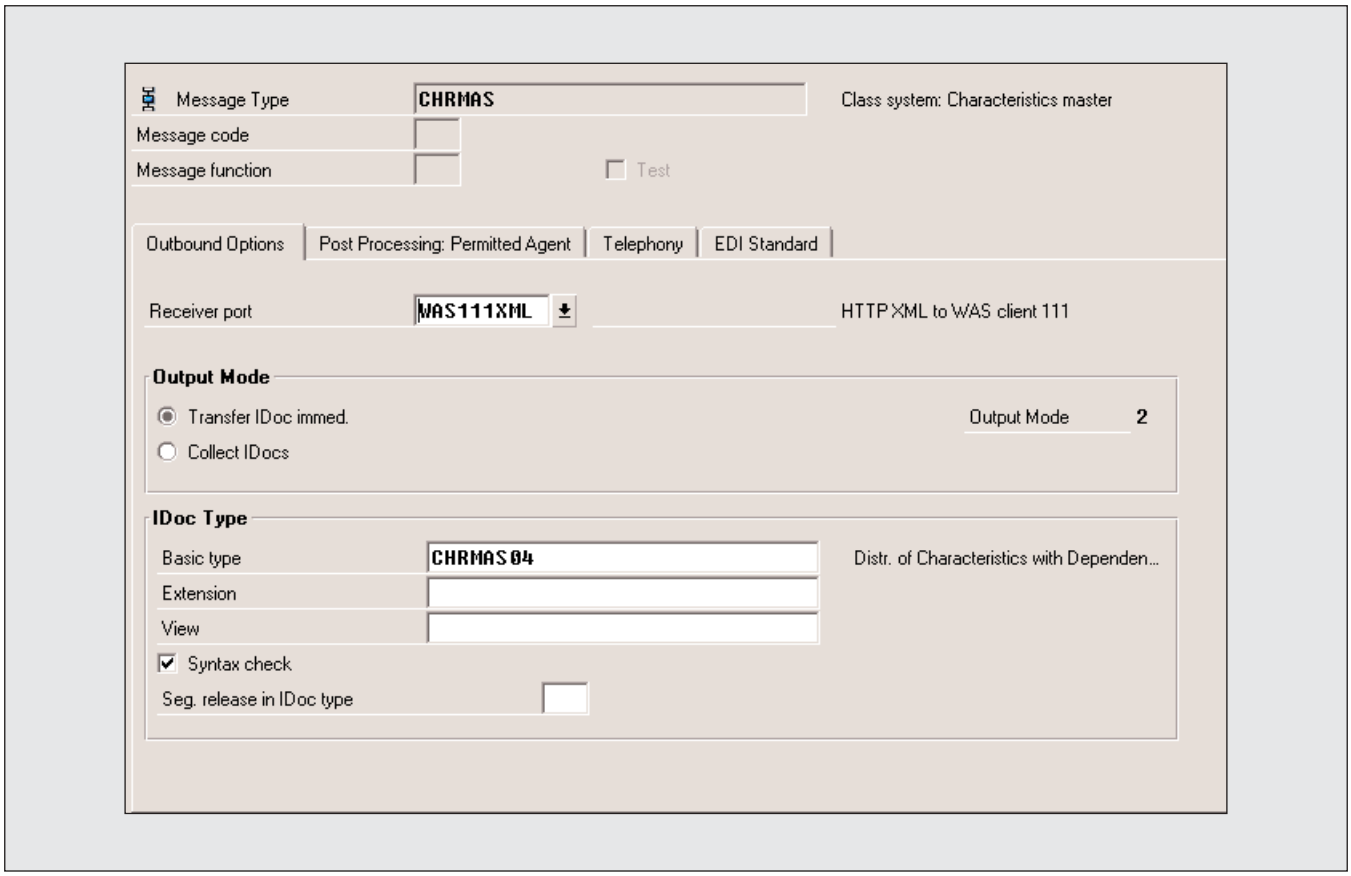
1. Call transaction *WE21* (Ports in IDoc processing).
2. Enter a name for the port and add a description.
3. Specify an XML HTTP port type for the RFC destination to the receiver system that you created in Step 3 (refer back to Figure 12), and in the *Content Type* field, leave the radio button selection at *Text/XML*.

Figure 13 shows the settings for our example XML HTTP port definition.

Task #3: Adapt the Partner Profile

In the last phase of our setup, we have to adapt the outbound parameters of the partner profile in order to

Figure 14 Partner Profile Adapted for the Example XML HTTP Distribution



transfer messages with HTTP/XML instead of RFC/IDoc.

On the system that sends master data messages:

1. Call transaction *WE20* (Partner Profile).
2. Navigate to the outbound parameters of the master data message type (MATMAS).
3. Change the receiver port (which is currently a tRFC port) to the XML HTTP port that addresses the receiver system.
4. Save your settings.

On the system that receives master data messages, use transaction *WE20* as just described to change the ALEAUD receiver port to XML HTTP.

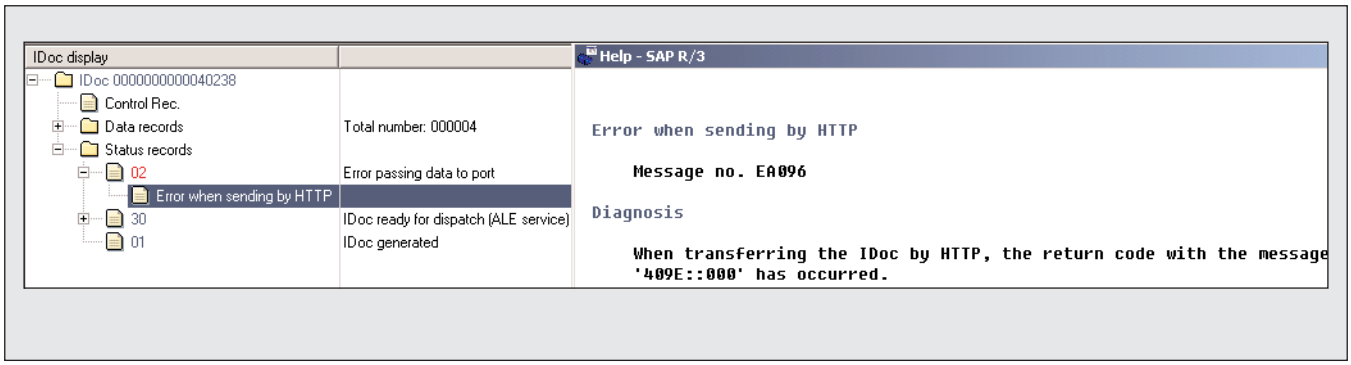
Figure 14 shows example settings for a partner profile where the message type is CHRMAS.

Step 5: Test the Distribution Scenario

All of the necessary settings have been made, and you can test the distributed business process based on HTTP/XML. Test it the same way you tested the template process based on tRFC/IDoc in the previous step.

Now that you know how to set up a fully functional HTTP/XML process on your own, it's important that you understand a particular issue regarding error processing with HTTP/XML, which we circumvented in the example using the ALEAUD message. The

Figure 15 IDoc Display with Status 02 and HTTP Error 409



next section explains this issue and the meanings of error messages you might encounter in an HTTP/XML distribution.

Error Processing in an HTTP/XML Distribution Scenario

SAP ALE with tRFC has error-processing features that ensure each message is delivered once, and only once (guaranteed one-time delivery). XML message distributions with HTTP, however, have some limitations regarding guaranteed one-time delivery. To understand the limitations of HTTP/XML compared to tRFC, we will make a short excursion into tRFC and file port error processing.

When a communication error with tRFC occurs, the system retries the communication process a specified number of times at defined intervals. As soon as a retry is successful, the system regards the process as ended. If no retries are successful, the IDoc remains in the tRFC queue and the tRFC is added to a list of unsuccessful tRFCs that is maintained in transaction *SM58* (Transactional RFC). You can reprocess the tRFC with transaction *SM58* when the communication issue — e.g., an error in the network connection — is resolved. The IDoc is then transmitted and the tRFC is deleted from the list of unsuccessful tRFCs. This mechanism ensures guaranteed one-time delivery and provides a convenient method for reprocessing IDocs after communication errors.

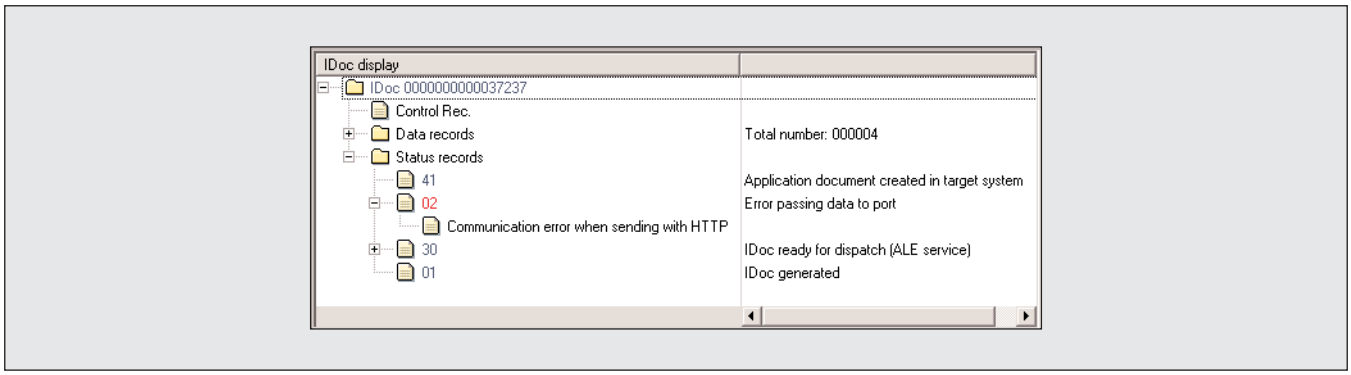
So how do you know if a tRFC is still pending? When you run program *RBDMOIND* (Status Conversion with Successful tRFC Execution), it indicates that a tRFC communication process for an IDoc has been completed by changing the IDoc status to 12 (*Dispatch OK*). Otherwise, the status of the IDoc remains at status 03 (*Data passed to port OK*). So, if you have run *RBDMOIND* and the IDoc status of a tRFC connection is 03, you can conclude that the communication process has not completed successfully and the tRFC of that process is still pending in the tRFC queue.

Error processing for distributions that use a file port type (File, XML File, etc.) works in a different, but similar, fashion. For instance, if an IDoc is supposed to be created as a file via a File port, and the file specified in the port type definition can't be opened, the IDoc will end with status 02 (*Error passing data to port*). After the problem is resolved — by specifying the proper directory for the file, for example — the IDoc can be reprocessed via ABAP program *RBDAGAIN*.

Communication errors with port type XML HTTP also result in an IDoc status of 02. IDoc status 02 with HTTP can indicate two types of errors: *Communication error when sending with HTTP* and *Error when sending by HTTP*. These errors are displayed in a subsegment of the status 02 record. See **Figure 15** for an example of status 02 with the error type *Error when sending by HTTP*.

Unlike messages sent via a File port, outbound

Figure 16 IDoc Display Showing That an IDoc with Status 02 Was Processed on the Receiver Side



messages sent as XML data via the XML HTTP port that have status 02 cannot be reprocessed via ABAP program *RBDAGAIN*. This is because, from the sender system point of view, you can't know with any certainty whether or not the message was processed by the receiver system after the occurrence of an outbound error. For example, think of an HTTP communication where the message data arrives at the receiver system but the HTTP response to the sender system is lost. In that scenario, even if the message is processed by the receiver system, the sender system will issue an error because it hasn't received a positive response from the receiver system. For this reason, SAP strictly excludes all IDocs from port type XML HTTP with status 02 from reprocessing.

With most status 02 errors of type *Error when sending by HTTP*, however, you can be fairly sure that the message hasn't been processed by the receiver system. For example, if the receiver system returns an error because there were problems encountered during HTTP processing (as in Figure 15), it could be the result of HTTP error 409 being issued, which indicates an error from the request handler. Because our request handler *CL_HTTP_IDOC_XML_REQUEST* implements all of the processing steps as a logical unit of work, if an error occurs, all database actions are rolled back and the system is left in a defined state. This means that if HTTP error 409 is issued, no data has been posted on the receiver system.

In any case, there is no reprocessing of IDocs from port type XML HTTP with status 02. If a communication error occurs, you must first find out if the message has been posted on the receiver side. If it hasn't, you must create the message again.

In the example distribution demonstrated in this article, we implemented an audit message to work around this issue — for every master data message created on the receiver system, an audit message is sent back that changes the IDoc status of the source message (in our example, the master data message). So if a communication error occurs, but the message is processed on the receiver side, the IDoc status will change; otherwise, it will remain at status 02. In **Figure 16** you can see that status 41 (*Application document created in target system*) has been added after a status 02 of type *Communication error when sending with HTTP*. This indicates that, despite the communication error, the message has been successfully processed on the receiver side.

Re-creating the message is easy in many cases — for example, when you need to create another ALE output message for business documents like sales orders, or when you are sending new master data messages with a program like transaction *BD10*, which is used for sending material master data. In other cases, it is difficult to re-create the message — for example, when you have master data messages created by the processing of change pointers. Unfortunately, I know of no good solution for that case. If you have to re-create such messages, you most likely will need to find the master data identification number and resend the master data with a transaction such as *BD10* for distributing material masters.

> ✓ ***Tip***
>
> *To arrange for a receiver or group of receivers to be notified by a workflow item when status 02 occurs, configure a workflow that will be triggered by process code EDIO.*
>
> *For more information on how to implement a workflow, in the SAP Library go to SAP Web Application Server → Basis Services / Communication Interfaces (BC-SRV) → IDoc Interface/Electronic Data Interchange (BC-SRV-EDI) → Processing IDocs → Exception Handling.*

Conclusion

Now that you know how to set up a distributed business scenario based on a standardized communication protocol (HTTP) and a standardized message format (XML), your horizons for distributed business processes have broadened. You can communicate with non-SAP systems in a world where HTTP and XML are both in widespread use.

HTTP/XML will get your data to places it can't go with tRFC. That doesn't mean you should use it all the time, however. I recommend using tRFC for communicating between SAP systems (and other tRFC-enabled systems) because of the additional communication features it puts at your disposal, like retrying erroneous communications and transaction control.

Keep in mind that as soon as your distributed process requires more complex integration features — in terms of communication, mapping, and so forth — you should consider using a more advanced application integration tool like SAP Business Connector or SAP Exchange Infrastructure (SAP XI).

Arthur Wirthensohn is a senior consultant at EDS Switzerland and a member of EDS's international Technical Leadership Network. He has worked both as a project manager and product manager in the ERP and IT integration business for many years, mainly in the retail, consumer products, manufacturing, and trade industries. For the past two years, Arthur was the product manager responsible for Application Services based on SAP systems. Currently, he works as a project manager and technical lead of the Enterprise Application Integration (EAI) department, which delivers SAP-related services such as ALE/EDI, SAP Business Workflow, Web Application Server, Data Migration, and ABAP Programming. Arthur can be reached at arthur.wirthensohn@eds.com.

