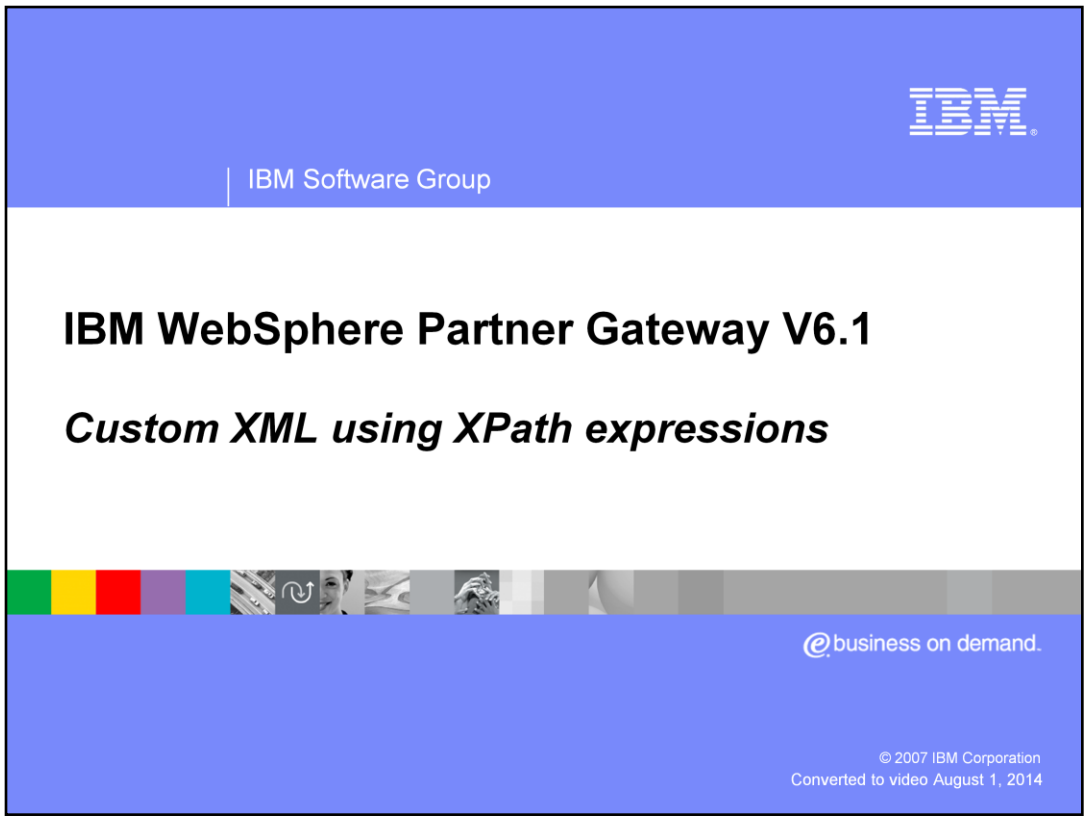
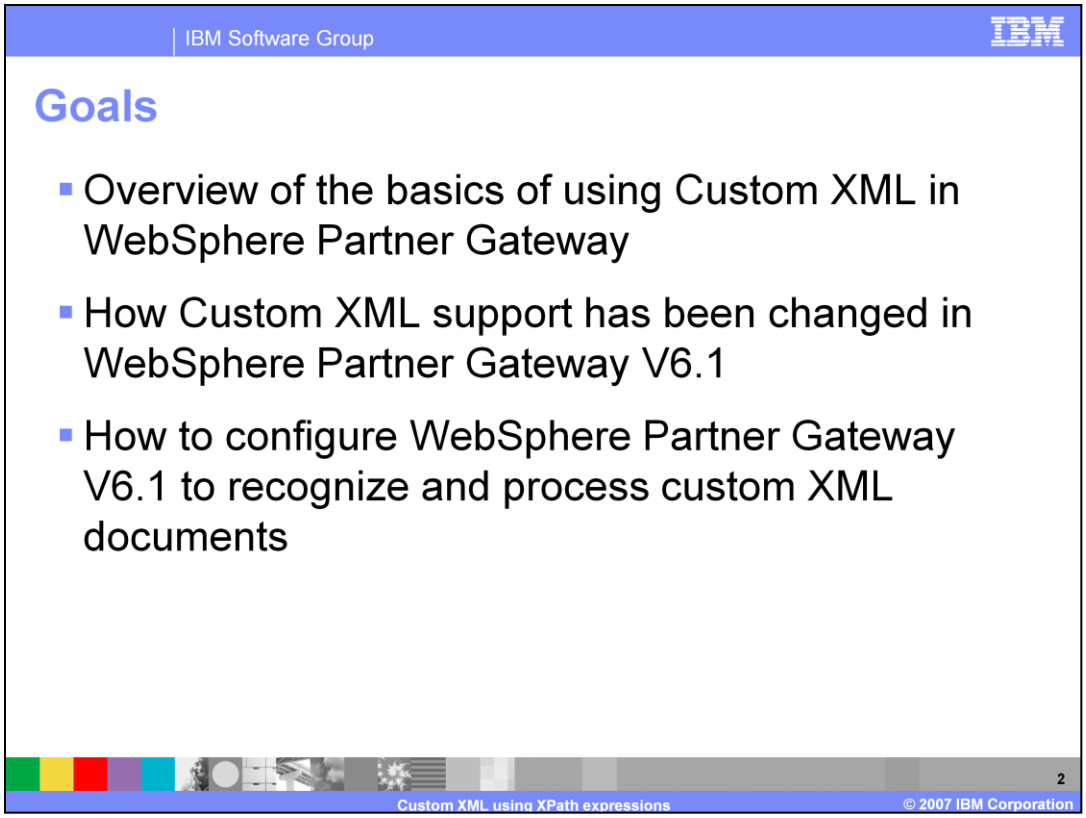
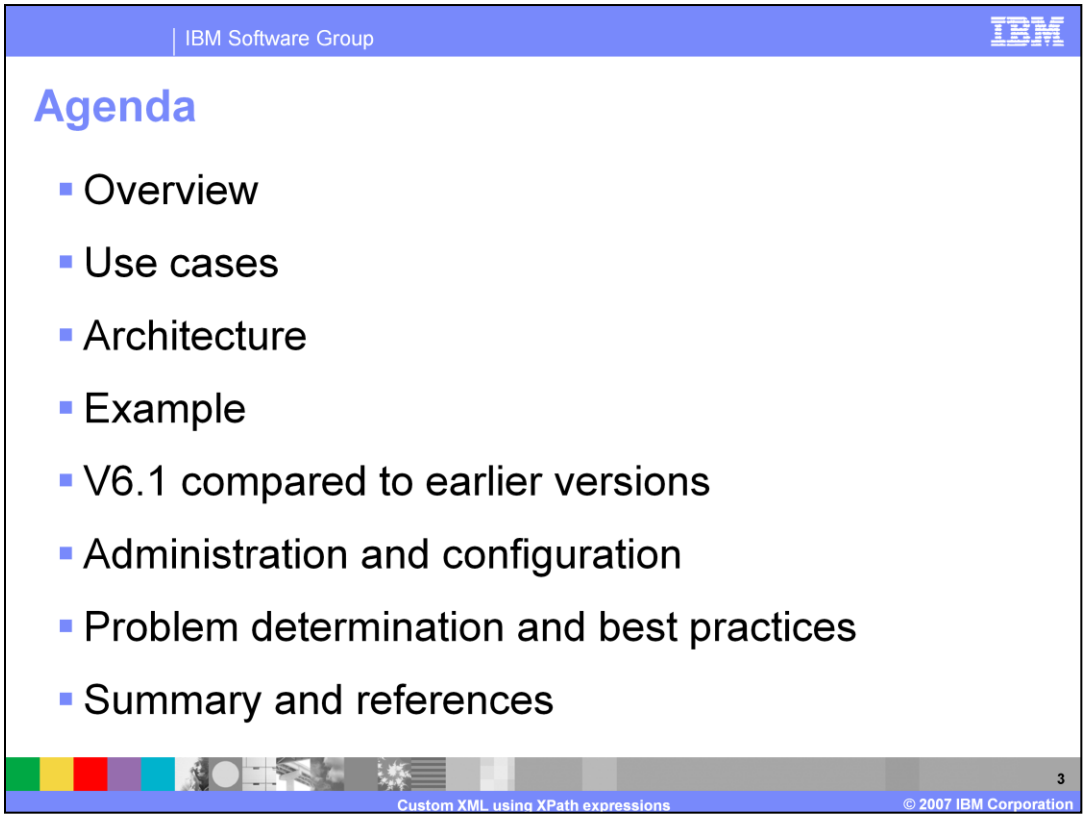


This presentation provides introduction to the configuration and usage of the new custom XML features available in WebSphere® Partner Gateway V6.1.

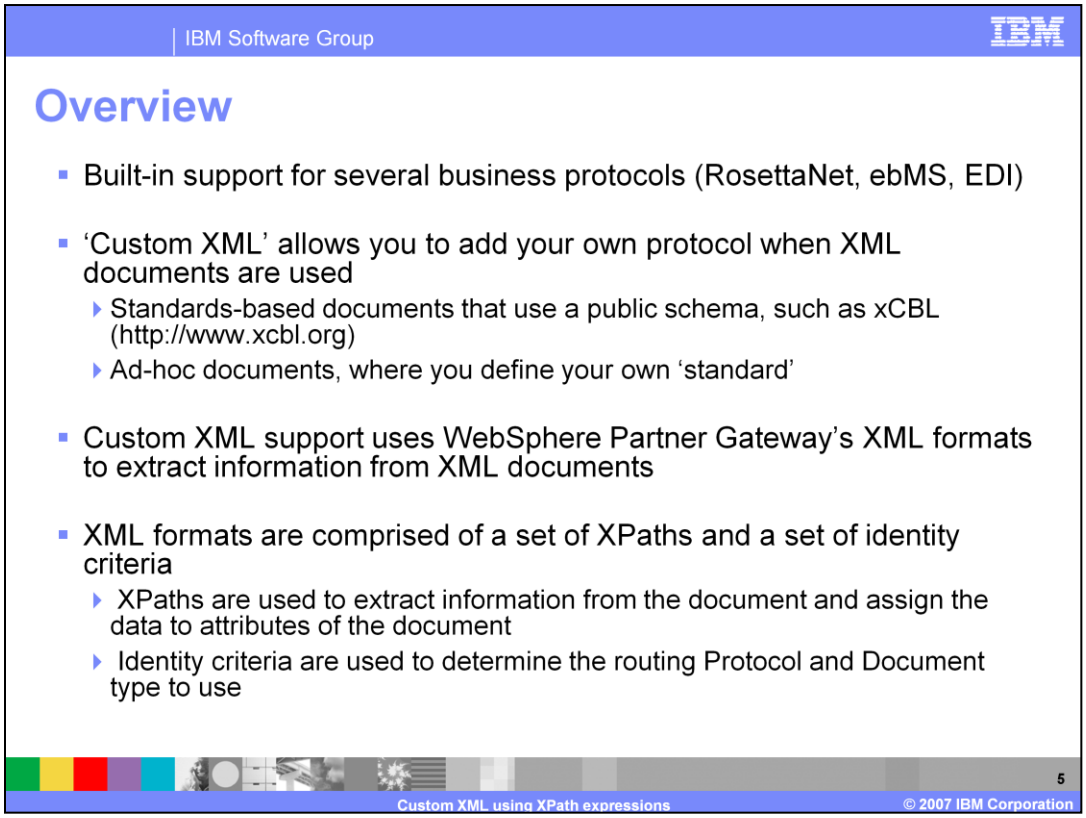


Besides just covering the new custom XML features in V6.1, the presentation will also go over some of the product internals that are needed to better understand how custom xml is processed. This includes discussion of the protocol parse fixed inbound workflow processing and the ways that synchronous flows are implemented.



In this presentation you will learn at a high-level about the new custom XML processing features in WebSphere Partner Gateway V6.1. You will see a set of V6.1 use-cases that could not have been implemented with WebSphere Partner Gateway V6.0. These use cases will help you to see the need for the new V6.1 processing features and they will give you an introduction to what those features are about. To better understand the custom XML processing features, you will then learn how they fit into the existing WebSphere Partner Gateway protocol processing framework.

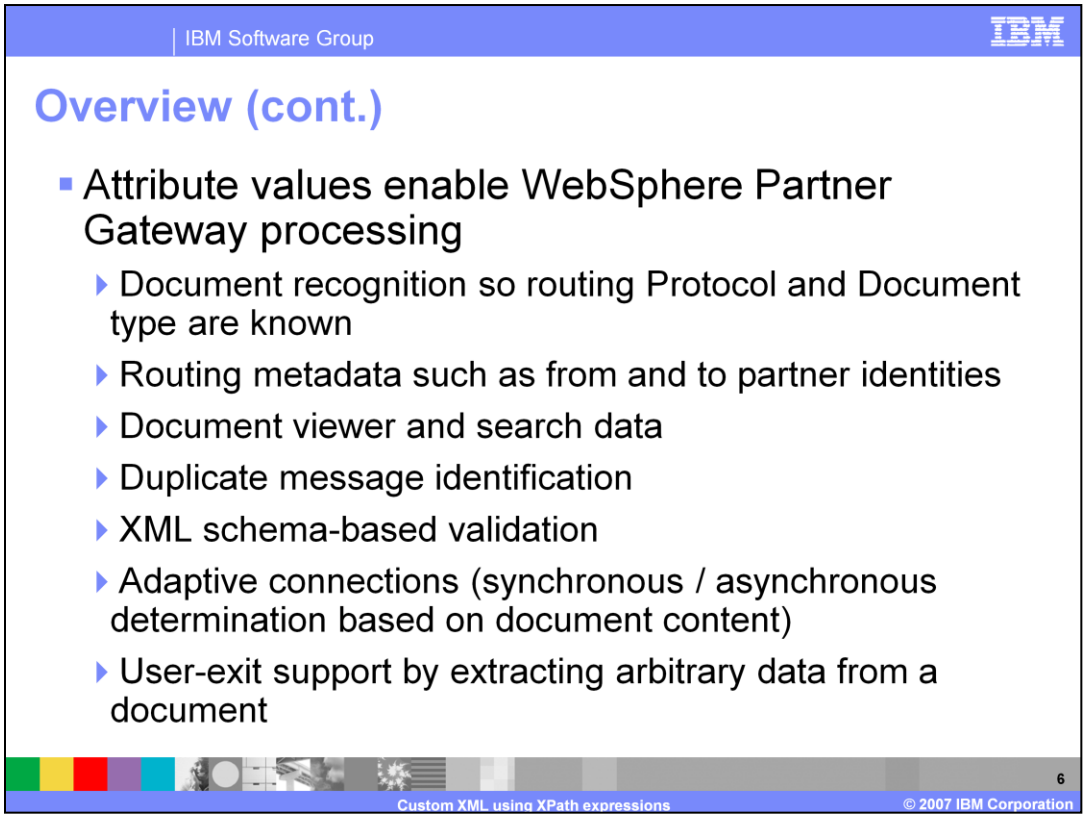
The next set of slides take you through a high-level view of the new custom XML processing features in WebSphere Partner Gateway V6.1.



WebSphere Partner Gateway V6.1 provides support for many standard business protocols like RosettaNet, ebMS, and others. The built-in support for these protocols can identify an incoming message, un-package it to separate the business message from its transport and protocol wrappers, and identify the business protocol that was used to form the message. Once the protocol is known, there is code for each built-in protocol to identify the specific document type and then to process it.

'Custom XML' is the term used in WebSphere Partner Gateway to describe the situation where you need to provide support for XML messages that are not formed using one of the built-in WebSphere Partner Gateway protocols. An example of this would be a case where a company has adopted the use of an industry accepted set of messages that are formed using a published XML schema. WebSphere Partner Gateway does not support all of the available public schemas of this type. So to handle these messages, you can use WebSphere Partner Gateway's custom XML functionality possibly along with some user-exit code if you need it for your requirements.

Custom XML in WebSphere Partner Gateway has always used XML formats and user-defined routing protocols and document types. This basic principle is not changed in V6.1. But the way that XML formats are defined and organized is different now. That is the focus of this presentation.

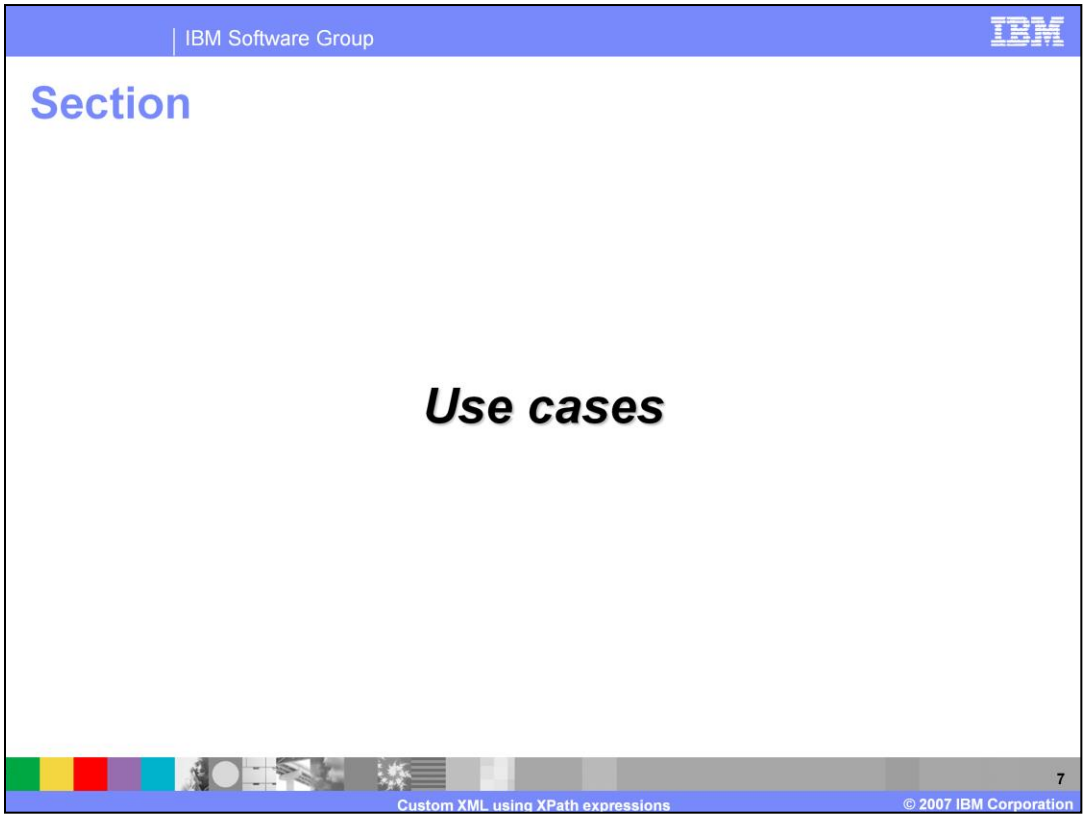


For all protocols, including custom XML, WebSphere Partner Gateway processes documents by extracting information from a document and its metadata. This extracted information is assigned to attributes of a BusinessDocument object that flows through each of the WebSphere Partner Gateway processing steps.

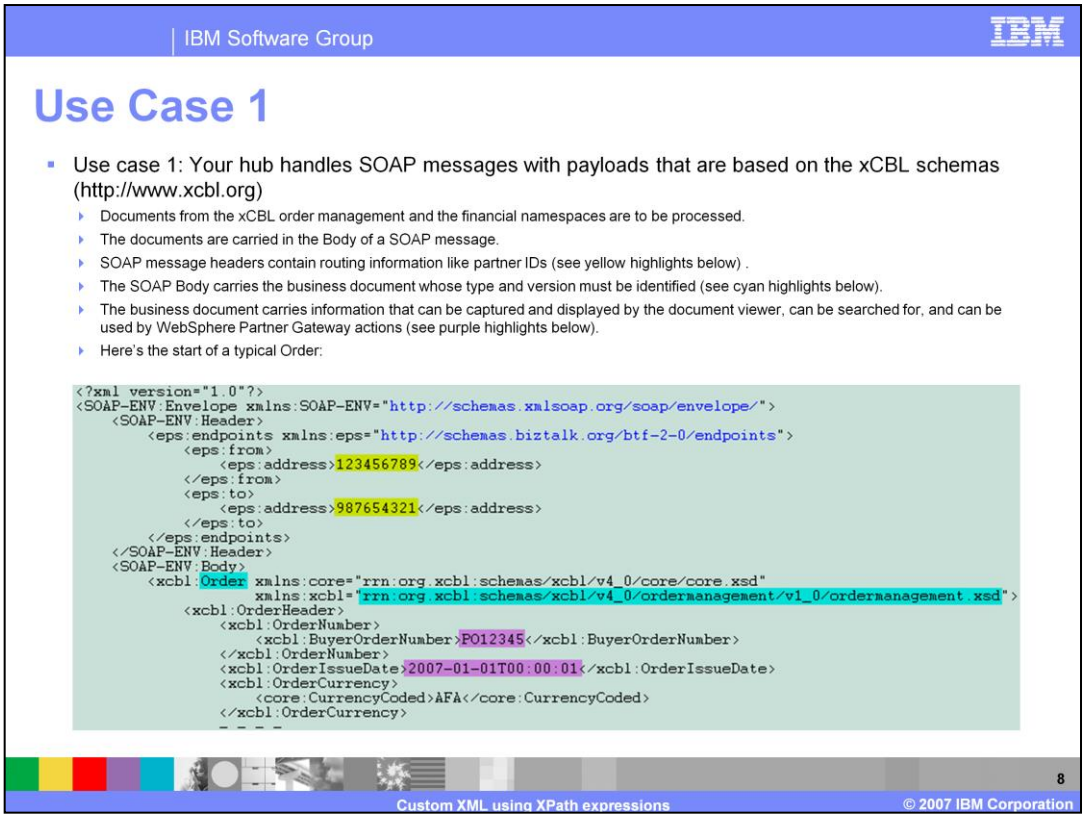
Each processing step gets the information it needs by accessing the attributes that were added to the document object.

This slide lists the information that can be extracted from a document using a version 6.1 XML format.

As you will see later, for each type of information on this chart an XML format uses an XPath version 1.0 expression to look into the XML message and extract the data. This data is then set into attributes of the internal BusinessDocument object for access by the system as the document is processed.



The next section covers some of the use cases of custom XML using the newly introduced features. More is possible with V6.1 compared with earlier versions because there are many new attributes provided in V6.1 XML formats. As you will see from the use cases, changes in the way that XML formats are defined and matched to documents provides a more powerful processing model in V6.1.



This is a typical scenario where there is a SOAP Envelope that carries a schema-based payload. The payload in this case is derived from schemas from the xCBL (XML common business library) site.

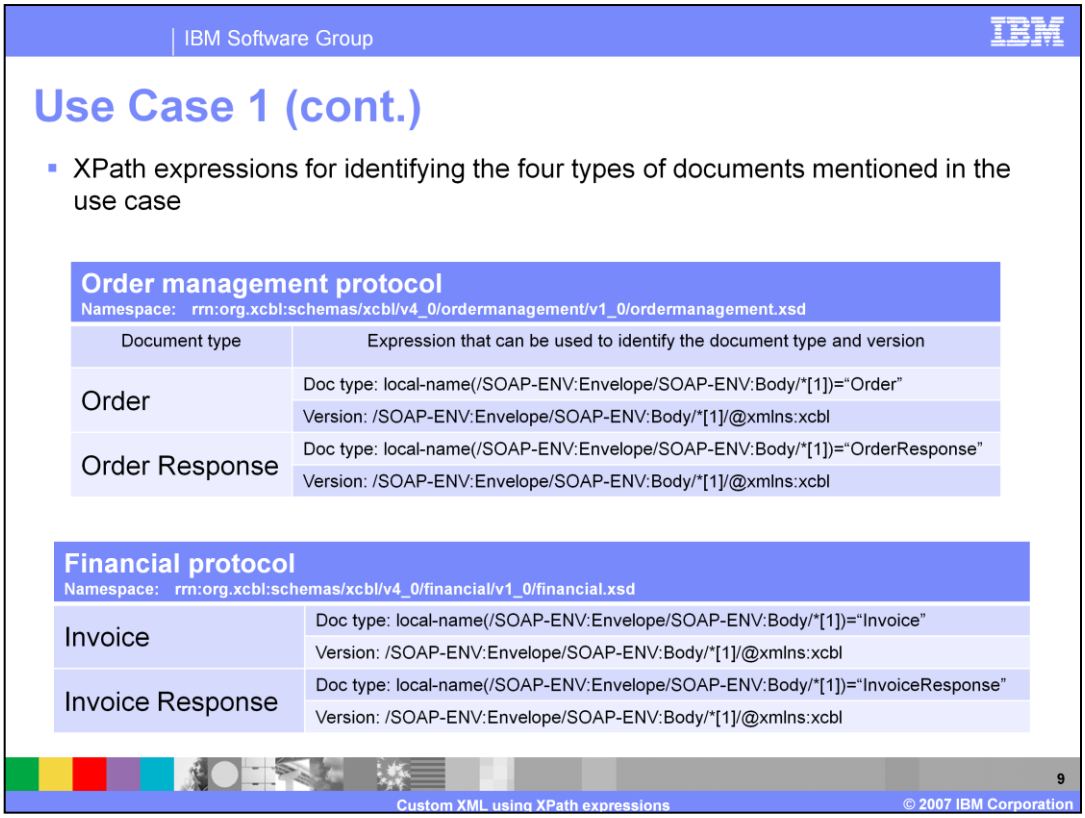
One thing to note here is that the use case involve SOAP messages, but they are not using the built-in WebSphere Partner Gateway capability for processing SOAP messages. Instead, the use cases here describe a scenario where you want to provide custom XML processing for SOAP messages that are received by your WebSphere Partner Gateway hub.

In use case 1, a set of documents needs to be handled. The business documents are based on a schema published by xcbl.org. The business documents are wrapped inside SOAP messages as the payload in the SOAP Body.

The SOAP headers are based on the Microsoft BizTalk standards. The headers carry metadata. In this example, the metadata that is carried is just the from-partner and to-partner IDs.

The business document itself has information that you want to make available to WebSphere Partner Gateway for purposes of identifying the message type and version. The cyan-highlighted fields can be used to determine that this is an Order that conforms to a particular version of the xcbl schema.

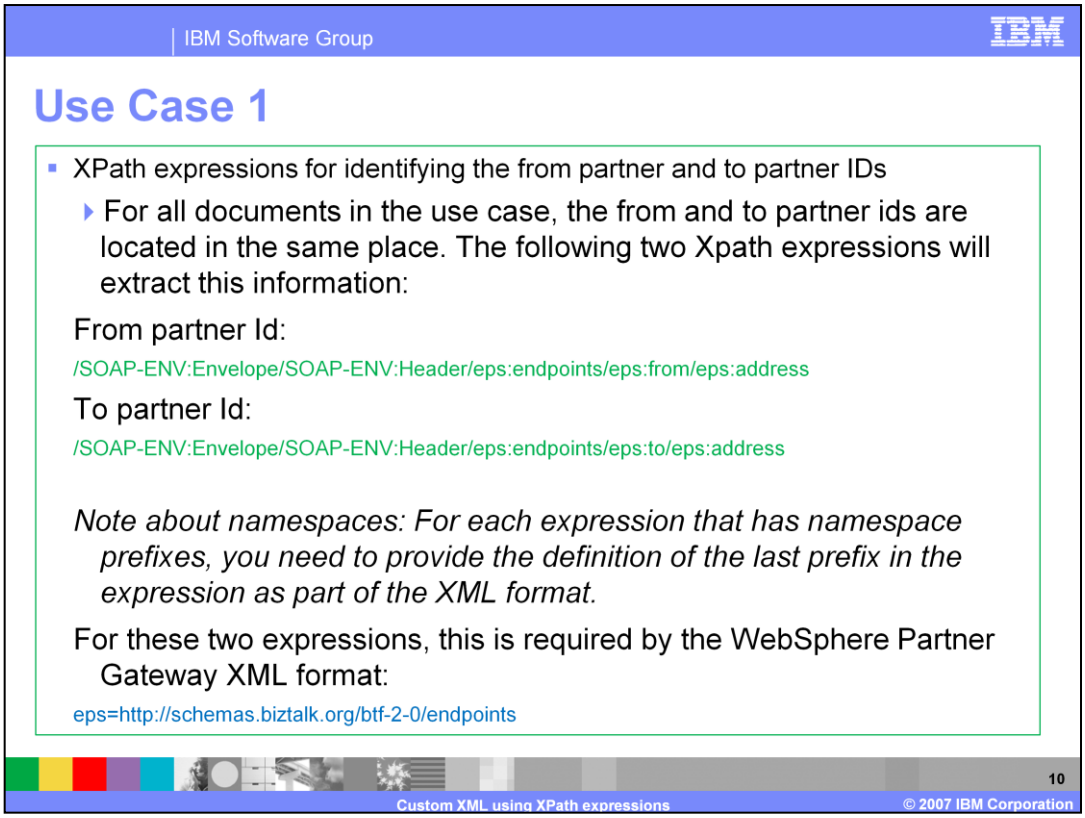
There is also information in the document that you want to make available to WebSphere Partner Gateway for processing purposes. The purple-highlighted fields are examples of such information.



To implement code that fulfills use case 1 and the remaining use cases you need to tell the system how to identify the document and its version. This is done using Xpath expressions that extract information from the document.

For document type and version matching, each XML format is assigned a document type identifier and version number. For this use case, the four document types are Order, OrderResponse, Invoice, and InvoiceResponse. The versions are the namespace for each category of message.

To write the Xpath expressions that extract information from the documents, you will need to know the basics of the Xpath expression language.



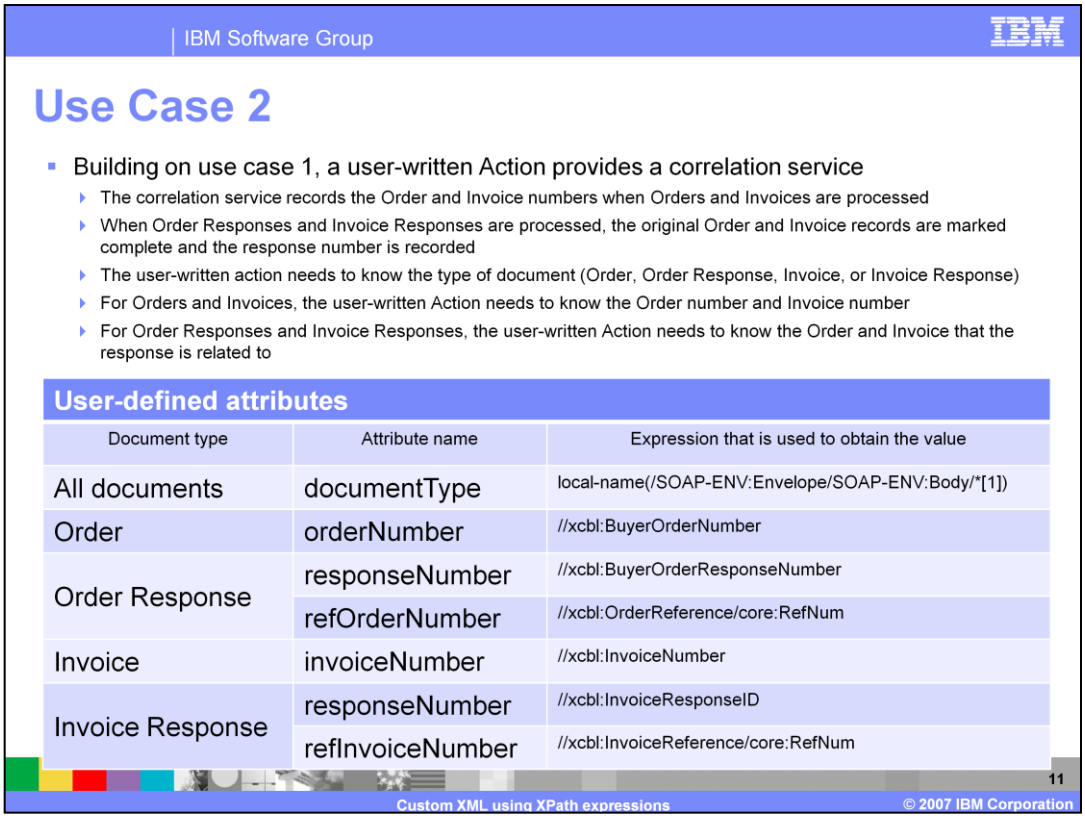
On the previous slide, you saw how the document identity can be determined by extracting information from it.

The from and to partner IDs also are needed to determine the partner connection that will be used to process the document.

Two simple XPath expressions can be used to obtain the IDs from the document.

The expressions shown on the slide use two different namespace prefixes, SOAP-ENV and eps. These namespace prefixes are defined in the document in different nodes, as you can see a few slides back on the first slide for use case 1.

Because of the way that the XPath evaluation code operates, it has to ‘know’ a context node that is in a path where all namespace prefixes are defined. You provide a ‘hint’ to this location as part of the XML format definition. The blue text in the slide shows such a hint for the namespace prefix eps. Later, you will see how this is entered into the XML format.



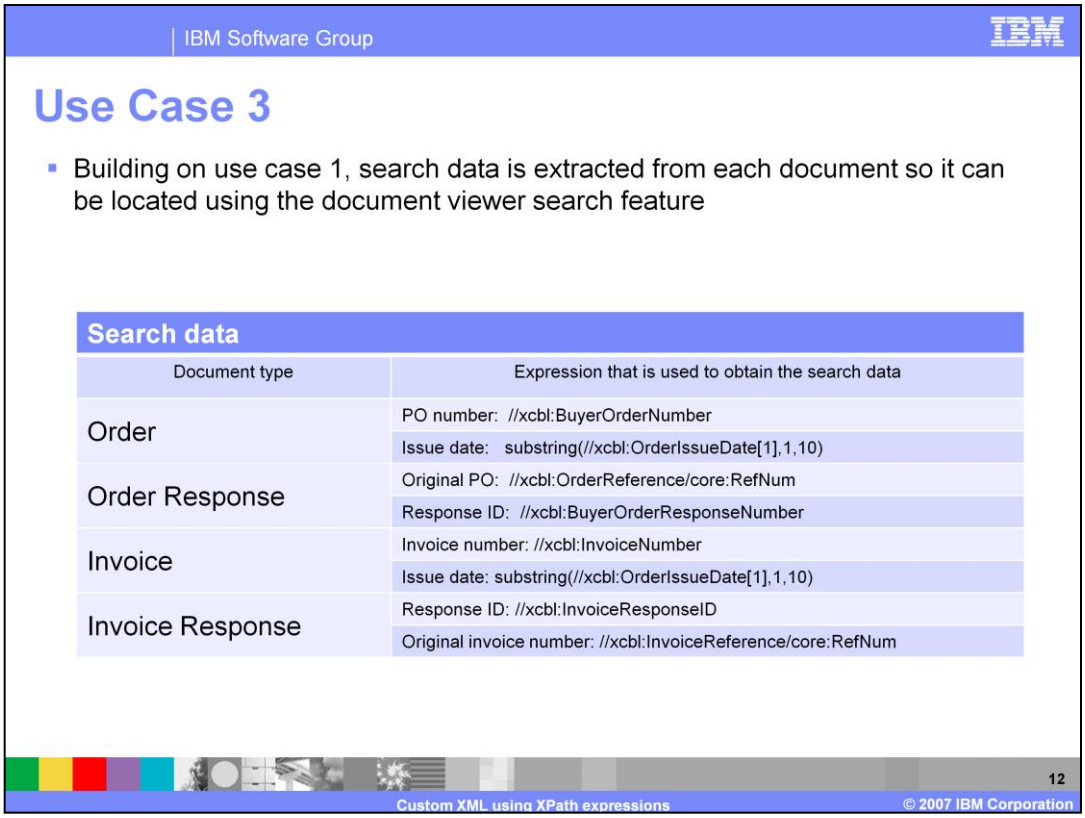
Use case one shows how a document and the sending and receiving partner IDs are identified. This is the minimum information needed to choose a partner connection, so with the information in use case one, the document can be routed.

Use case two assumes that you want to be able to correlate OrderResponse with Orders and InvoiceResponses with Invoices. The discussion of this case centers on the data needed by such a correlation service.

For this case, you will need to have a user-written Action that is a correlation Action. For input, this Action will need to record each Order or Invoice number, and it will need to know for each response the number of the associated Order or Invoice and the number of the response.

The new V6.1 XML formats allow you to obtain arbitrary information from documents and store this information in user-defined attributes. For each piece of information that is needed, you can add a user-defined attribute to the XML format. This attribute is then set with the result of evaluating an XPath expression.

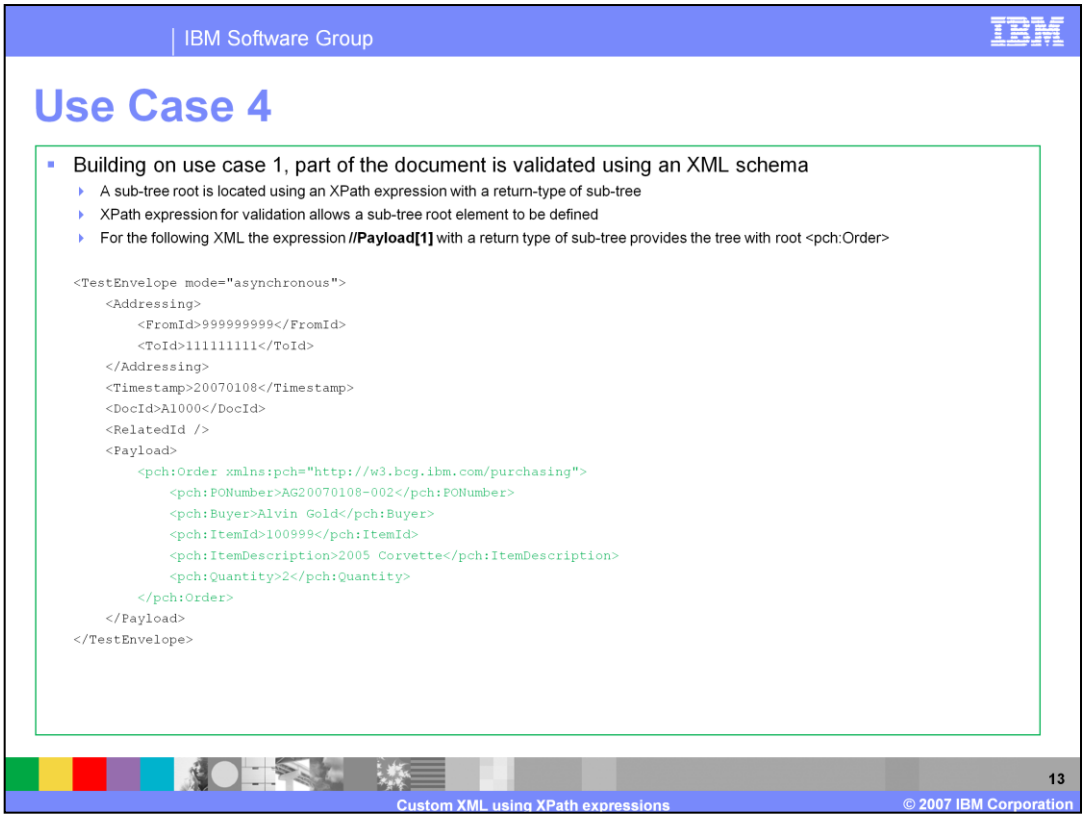
Your user-defined Action code that implements the correlation service can then access the information by retrieving the user-defined attributes from the document.



The third use case shows how you can extract information from the document so it can be saved and made available to the document viewer search function.

The XML format includes ten search fields that can be set with data extracted using XPath expressions. Then you can enter a search target into the document viewer and view only documents that set the search field with data that matches the target.

The use case shows two fields that could be of interest for each of the four document types considered.

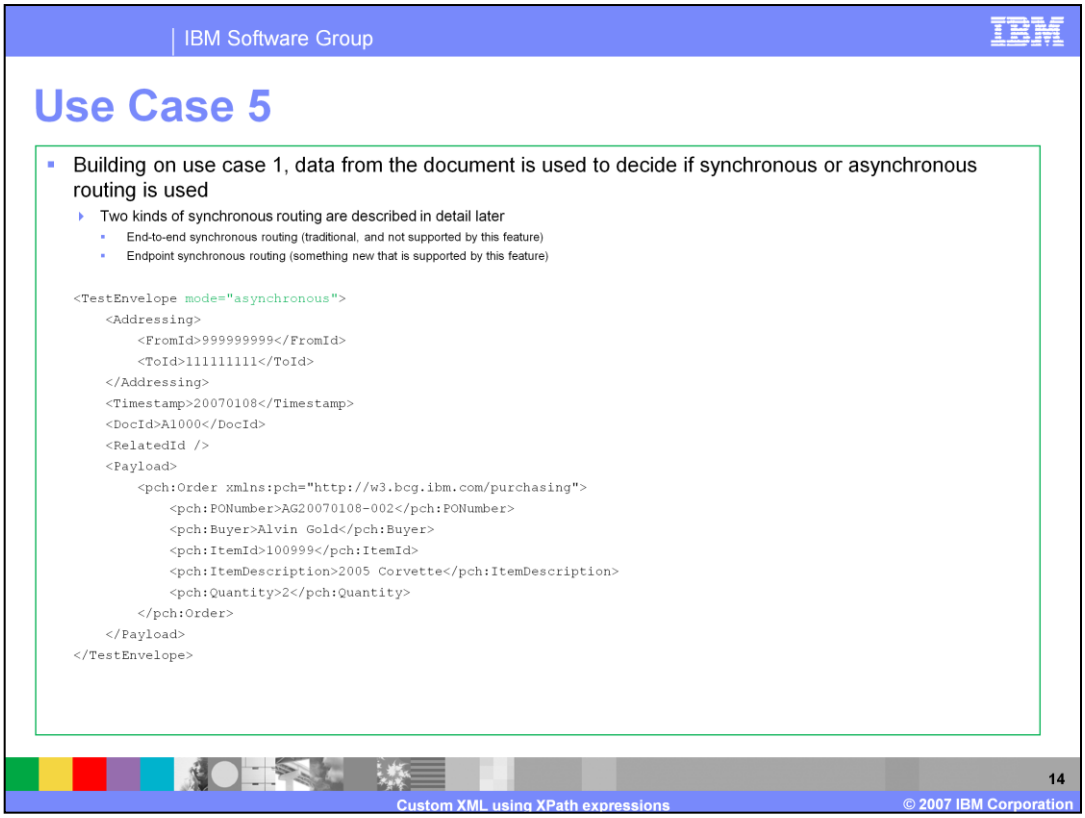


Use case four requires that a part, or sub-tree, of the document be extracted and validated.

This is how enveloped messages are typically formed. The business document portion of the message is carried as a payload of the complete message. There is an XML schema that can be used to validate the payload, but not the entire document.

This use case is fulfilled by the new XML format by allowing you to specify the root node of a sub-tree in the document that will be validated when a validation Action is chosen for the connection.

A different, simpler message than discussed earlier is shown here because it is short enough to show the entire message. The first child of the Payload node in this document is the root node of a business document. The portion shown in green above is the business document that is selected for validation. The expression //Payload[1] is one of many possible ways to identify this sub-tree.

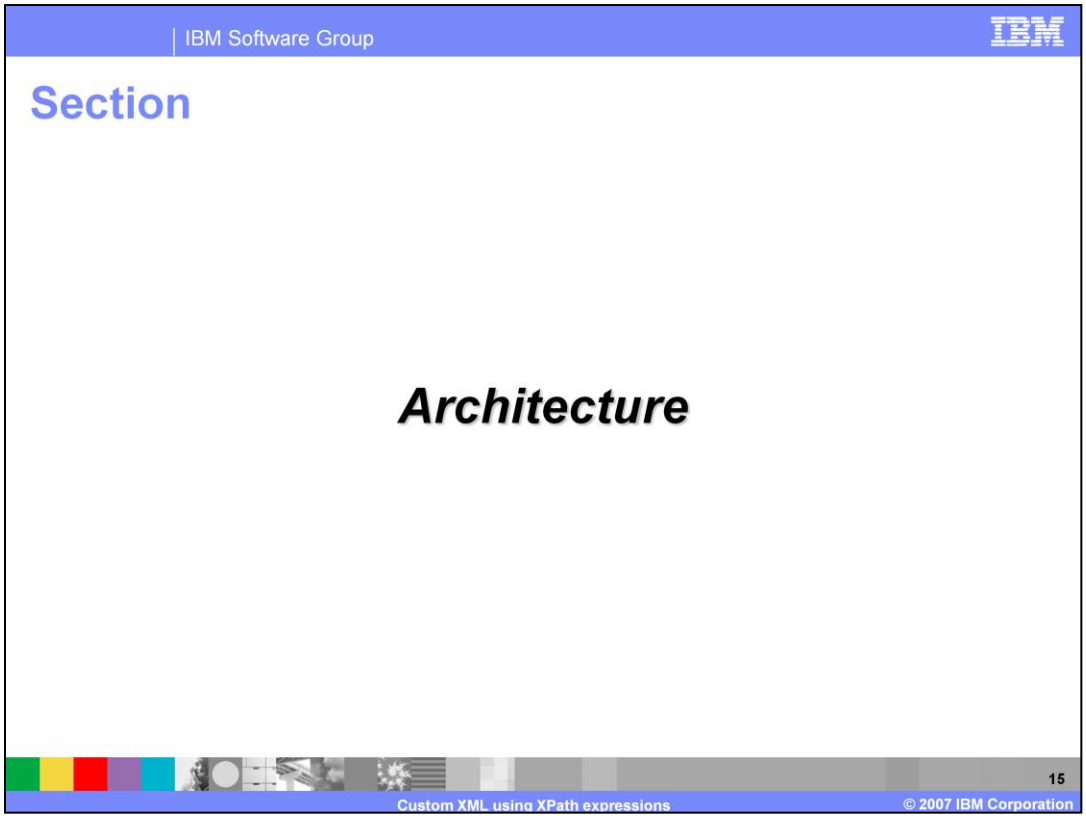


New XML formats provide a way to control synchronous interaction based on message content. The typical synchronous interaction pattern that is provided by earlier versions of WebSphere Partner Gateway is an end-to-end synchronous interaction that holds the original request connection open for the synchronous response from the recipient of the request. Control of this ‘end-to-end’ type of synchronous interaction is not supported by the XML format synchronous flag.

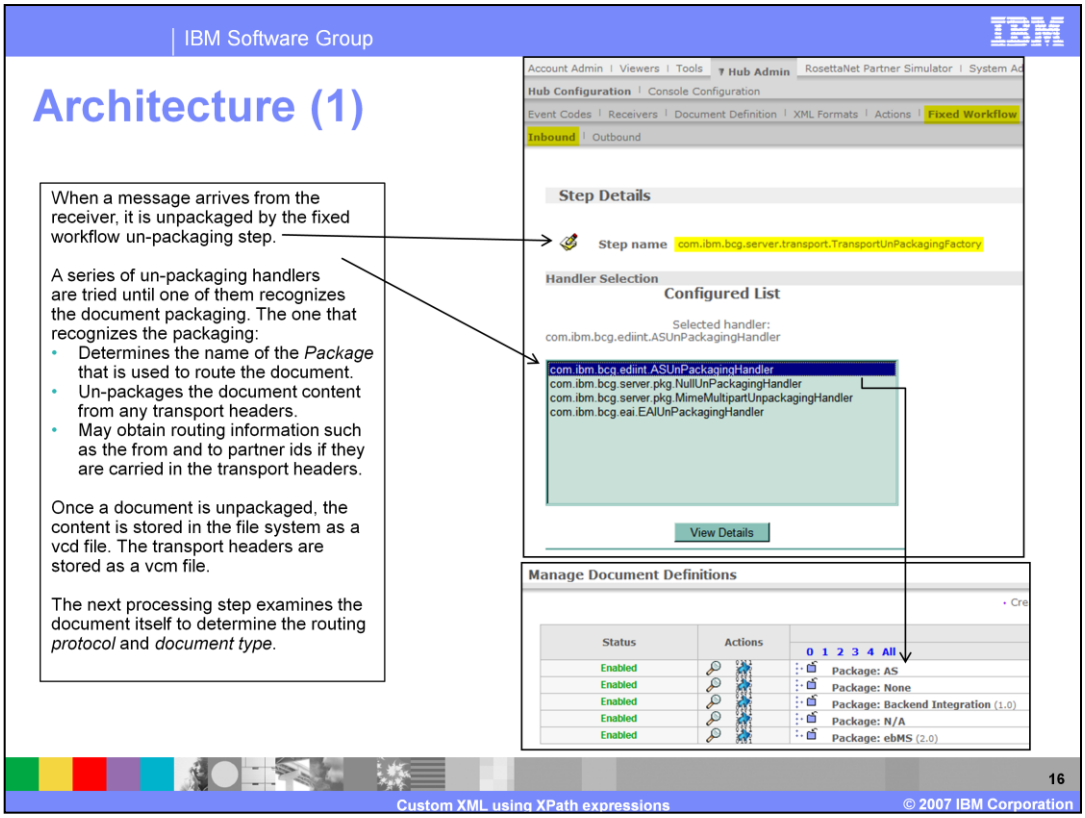
What is supported is a variation of synchronous routing or also called ‘endpoint synchronous’ routing. The original request is handled asynchronously in this case, but the XML format can test an XPath expression to set an attribute named ‘Synchronous flag’.

If ‘Synchronous flag’ is set to the string ‘true’ then if the destination is an Http destination, that destination connection will be held for a synchronous response. The synchronous response will be delivered to the sending partner’s destination as set on the connection. Notice that the sending partner’s destination in this case can be of any type, the response will be delivered to it regardless.

On the other hand, if the ‘Synchronous flag’ is not set, or is set to anything other than ‘true’, the message is delivered to all destination types in an asynchronous manner.



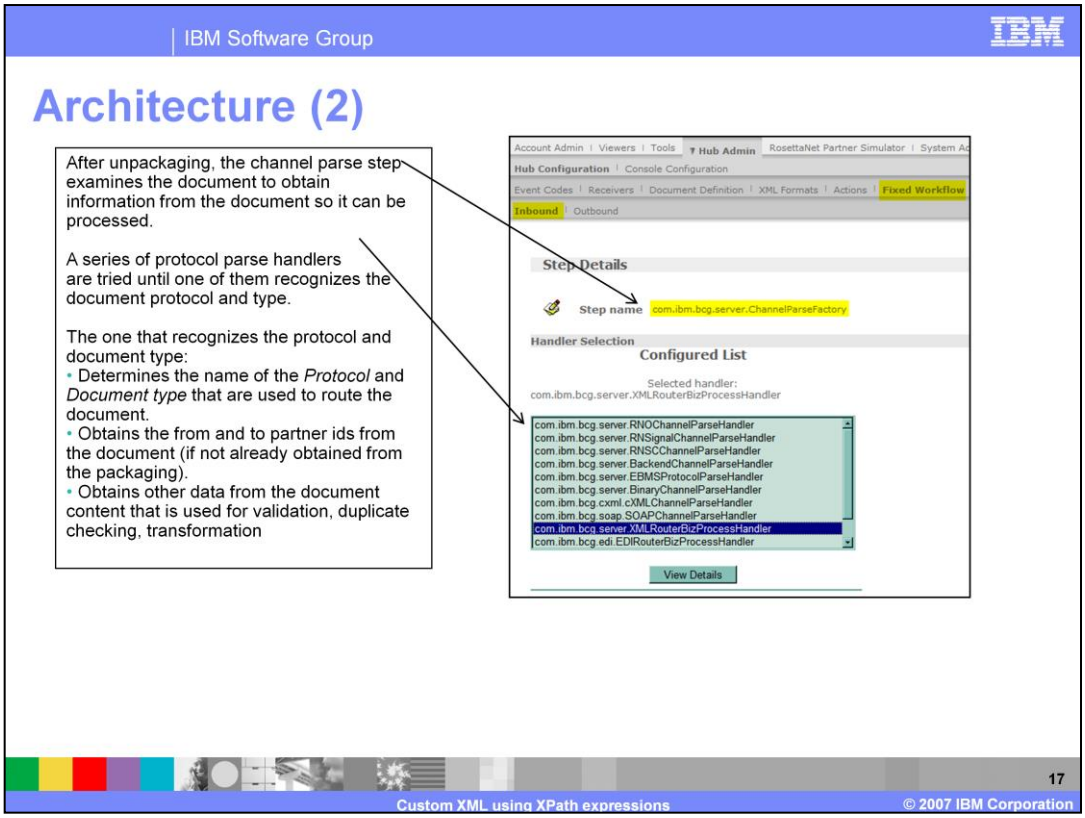
In the architecture section you will learn about how the new XML processing feature fits in to the existing framework for document routing and processing.



The WebSphere Partner Gateway framework operates by executing a fixed set of workflow steps for each document that is received. The next few slides walk you through these steps. Not all of the steps are directly involved in processing custom XML documents. But they are all covered so you can see the entire picture.

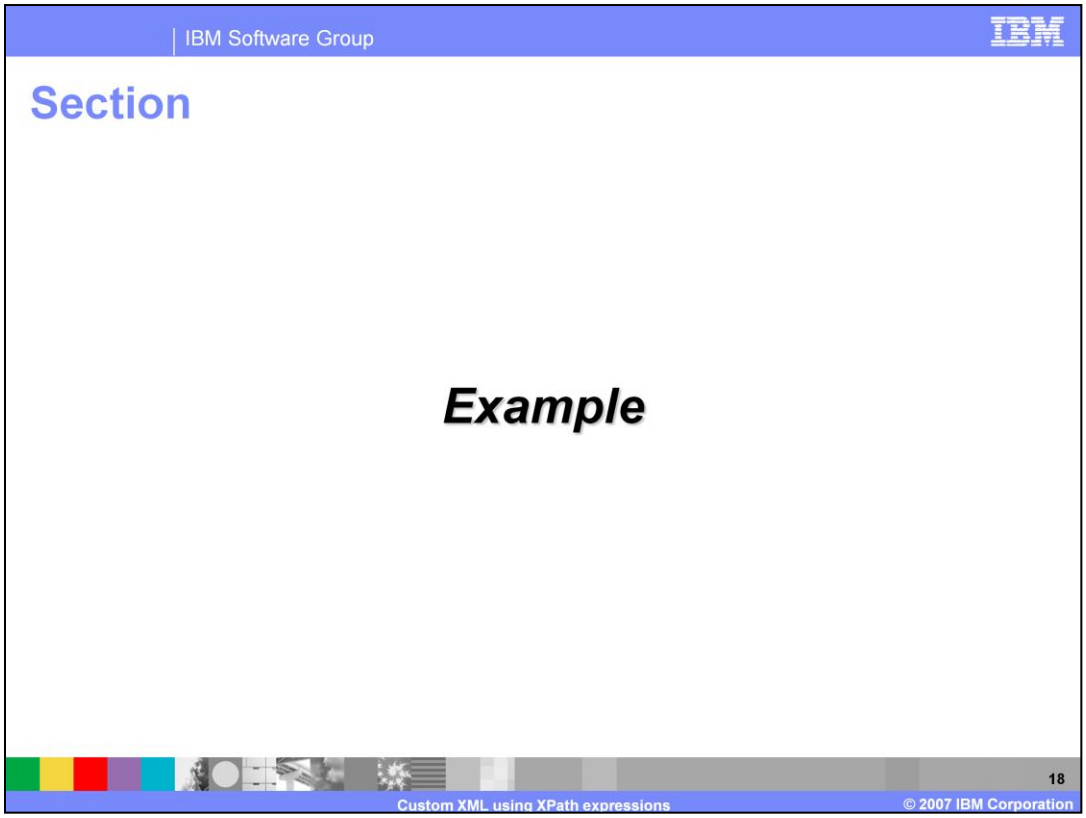
The first fixed workflow step is to un-package the document. Depending on how the document was sent it will be packaged in different ways. The term packaging refers to the types and names of transport headers that arrive with the document and the structure of the document itself.

Un-packaging executes a set of handlers in a specified order to see if one of them ‘recognizes’ the document packaging. Once one of them done, it sets the packaging for the document, isolates the document from its transport headers storing these in the file system, and the next processing step is performed.

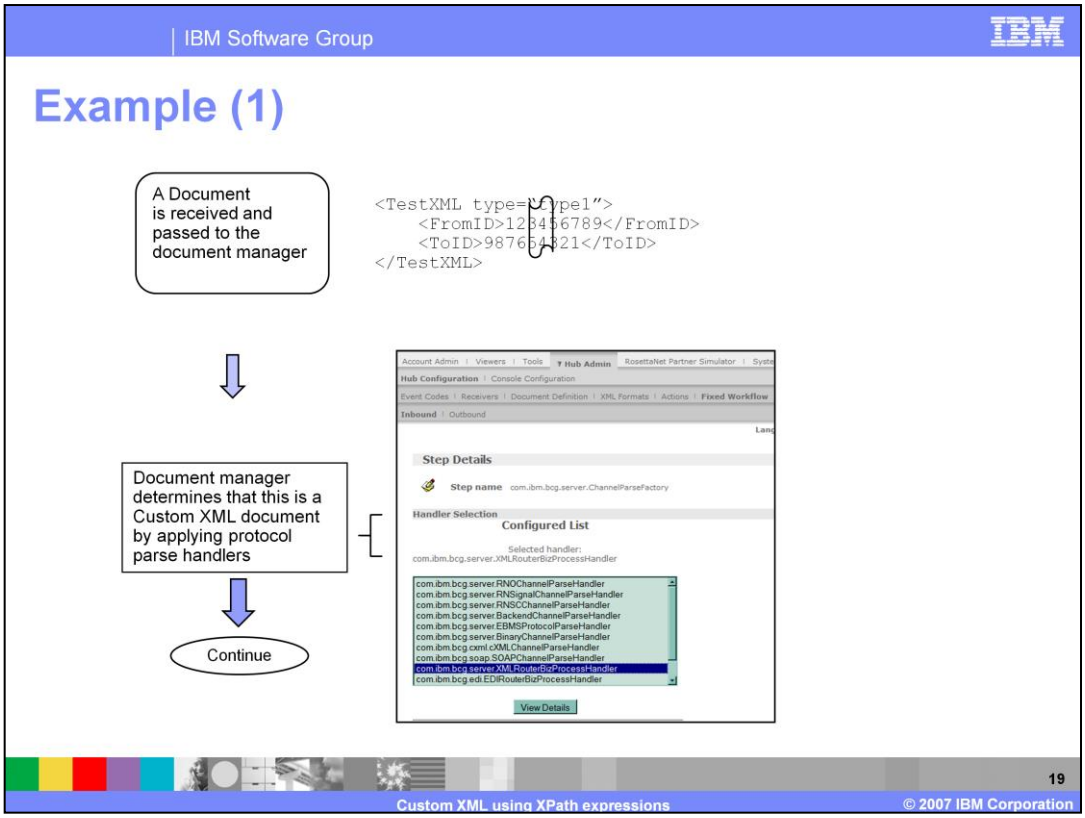


The next fixed workflow processing step examines and extracts information from the un-packaged document. This step is called the protocol parsing step.

As with un-packaging, protocol parsing is performed by running a set of handlers in a specified order until one of them ‘recognizes’ that it should handle the document. The XMLRouterBizProcessHandler class identifies an XML document and locates the appropriate XML format for the document. It also evaluates the XPath expressions in the XML format in the context of the document, and sets the BusinessDocument attributes using data extracted from the document.

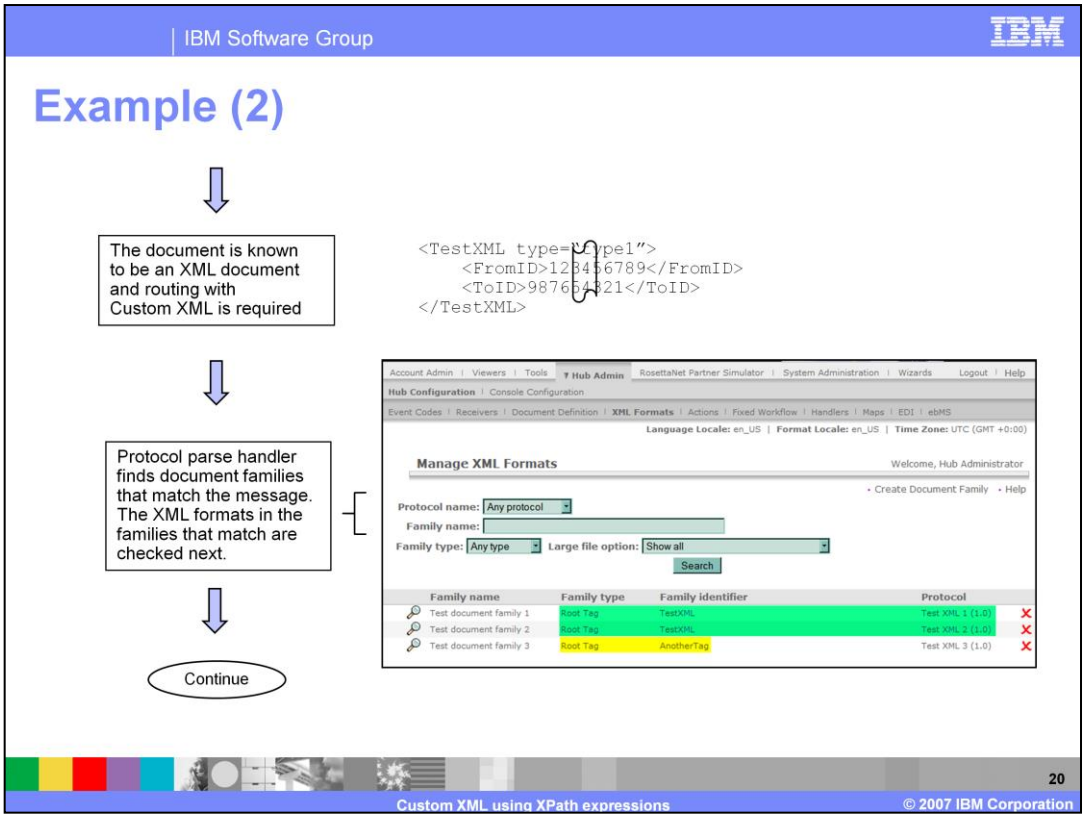


An example that uses a very simplistic XML document is presented in the next few slides. As part of the example, you will have your first look at the main configuration screens.



A simple document is shown on this slide. Some key things to notice now are that it has a root tag name of TestXML, type attribute set to say ‘type1’ , and elements to carry the from and to partner IDs.

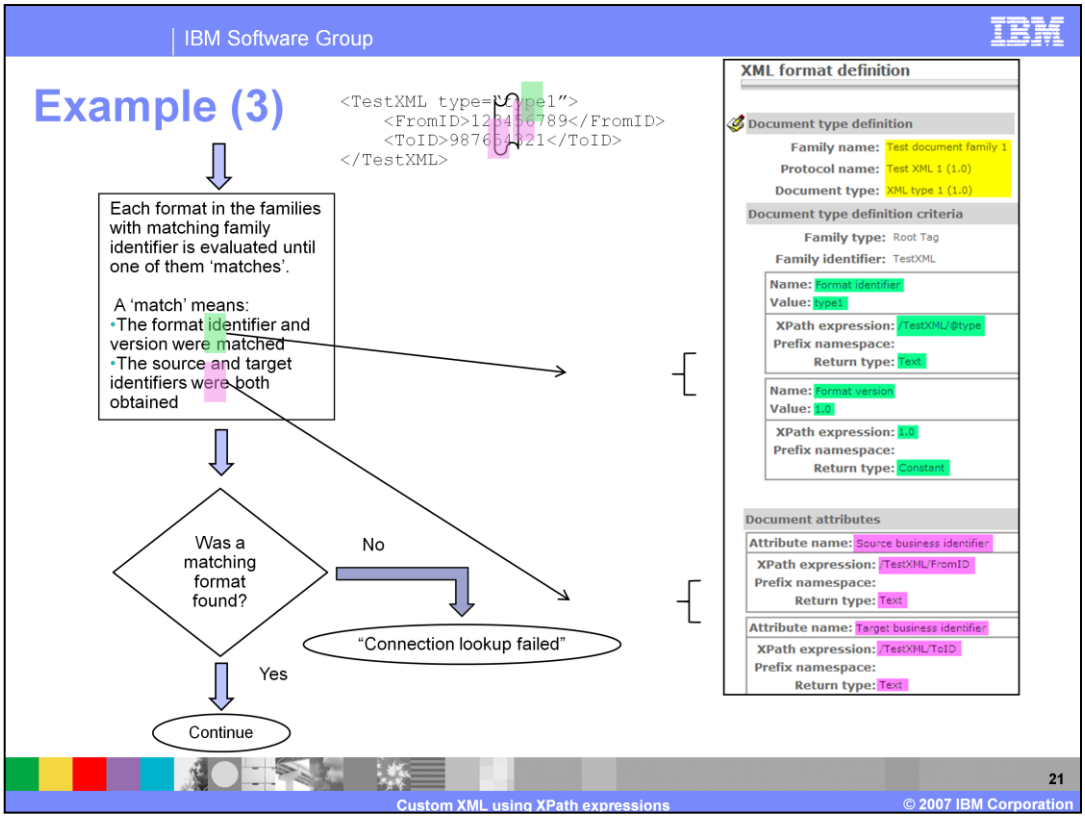
After the document is un-packaged it is passed to each of the protocol parse handlers in sequence until one of them recognizes the document. In this the XMLRouterBizProcessHandler will process the document because each of the earlier handlers in the list will not find what they look for to match the document.



The XMLRouterBizProcessHandler examines the document to find its root tag name, root tag namespace (if any), and DTD name (if any). After these are known the list of XML document families is searched for all families that match the document so far.

In this example, there are three document families in the system. Two of them apply to documents with a root tag name of TestXML, so the code selects them for further comparison with the document.

Note that the search order is first for matching DTD name, then for root tag namespace, and finally for root tag name.

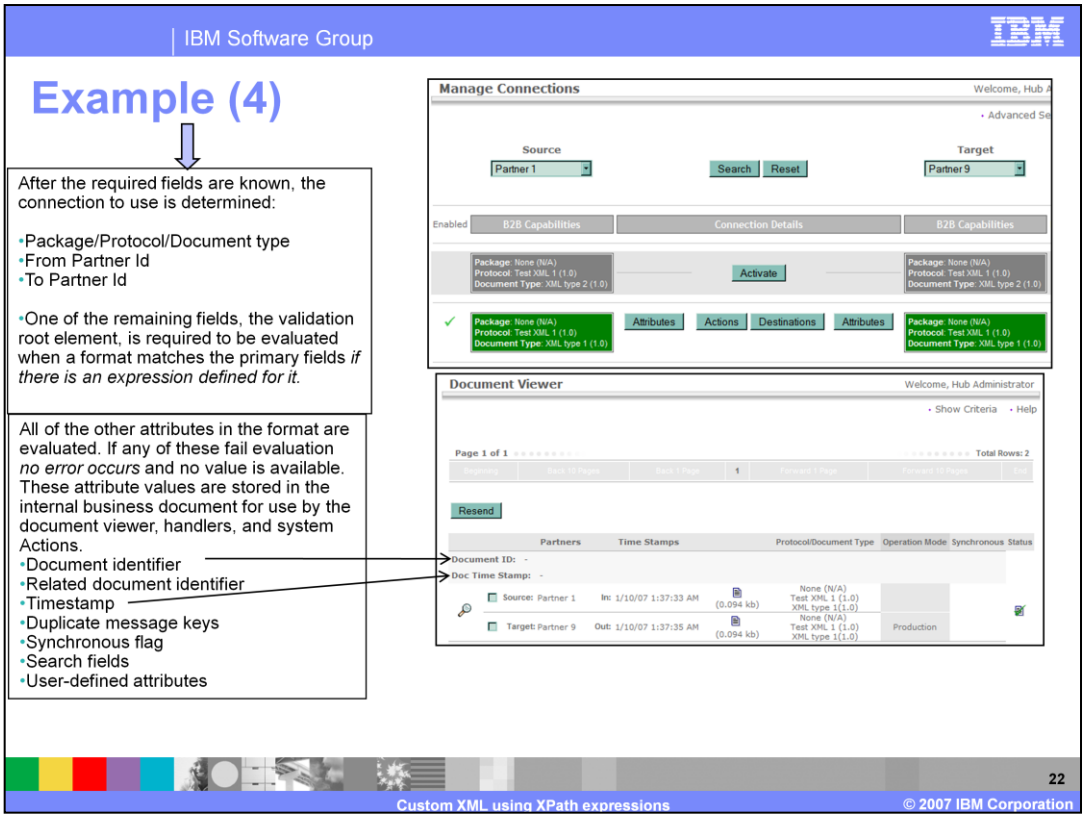


All of the XML formats in the qualified format families are searched one by one until a 'match' is found for the document.

In this example, there is a format defined as shown on the right. The information highlighted in yellow tells you that it is a member of 'Test Document Family 1' . If this format is used, the routing protocol named 'Test XML 1' version 1.0 and document type 'XML type 1' version 1.0 will be used to route the document.

To match this format, the data determined by the green highlighted XPath expressions has to match the corresponding Values. So when the expression /TestXML/@type is evaluated, if its value is the string 'type1' then the Format Identifier matches. The Format Version also has to match.

Two other values, the from and to partner Ids, are required before a complete match is declared. These are highlighted in pink on the right side of the chart. Notice that these do not need to match any target Value, they just have to have values so WebSphere Partner Gateway can find a connection.

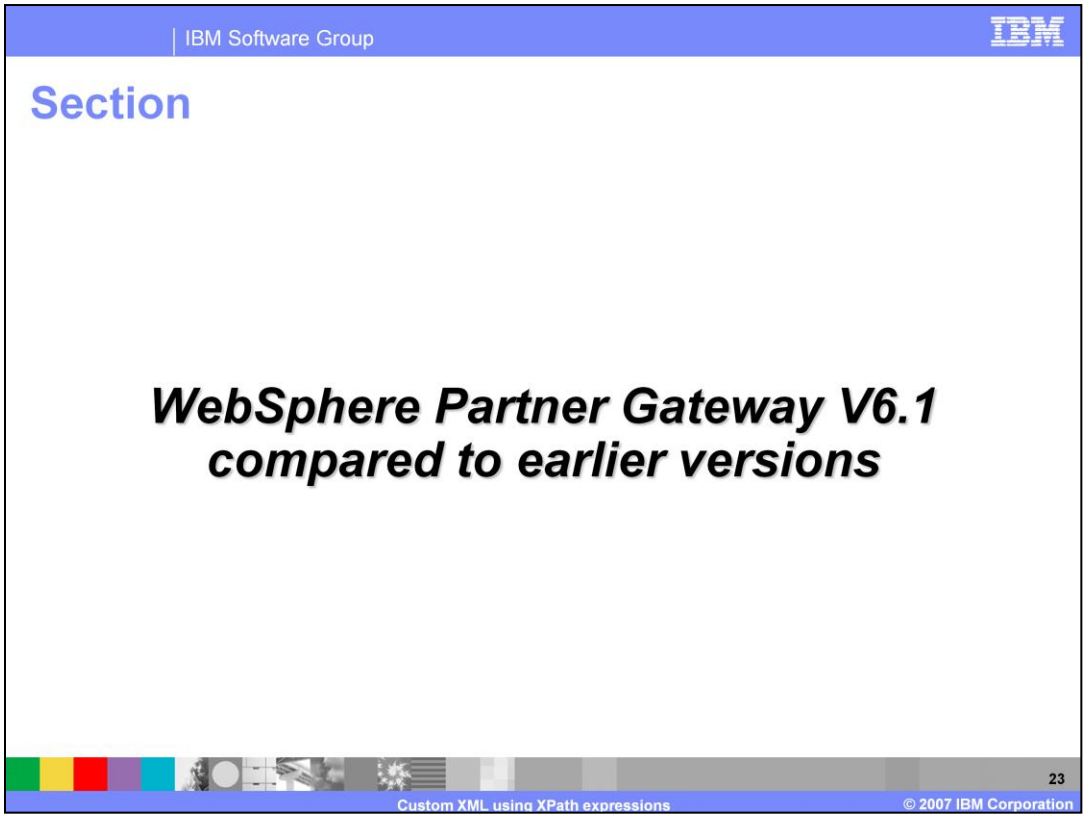


At this point in the example, the system has determined the routing package name from the un-packaging step, the routing protocol and document type because an XML format match was found, and the from and to partner Ids that were extracted by the XML format expressions.

The items known so far allow the system to choose a partner connection to use for routing the document. If there is not a connection configured with this information, a Check Channel Error code 210001 is logged.

There are several other attributes to be evaluated once a format is matched. One of these, the validation root element, is handled differently than all of the others. The validation root element is required to be evaluated if there is an expression for it. The reason for this is that validation is considered essential when it is specified. So a Protocol Parse Error code 240615 is logged when the validation root expression is invalid or cannot be located in a document that has a matching XML format.

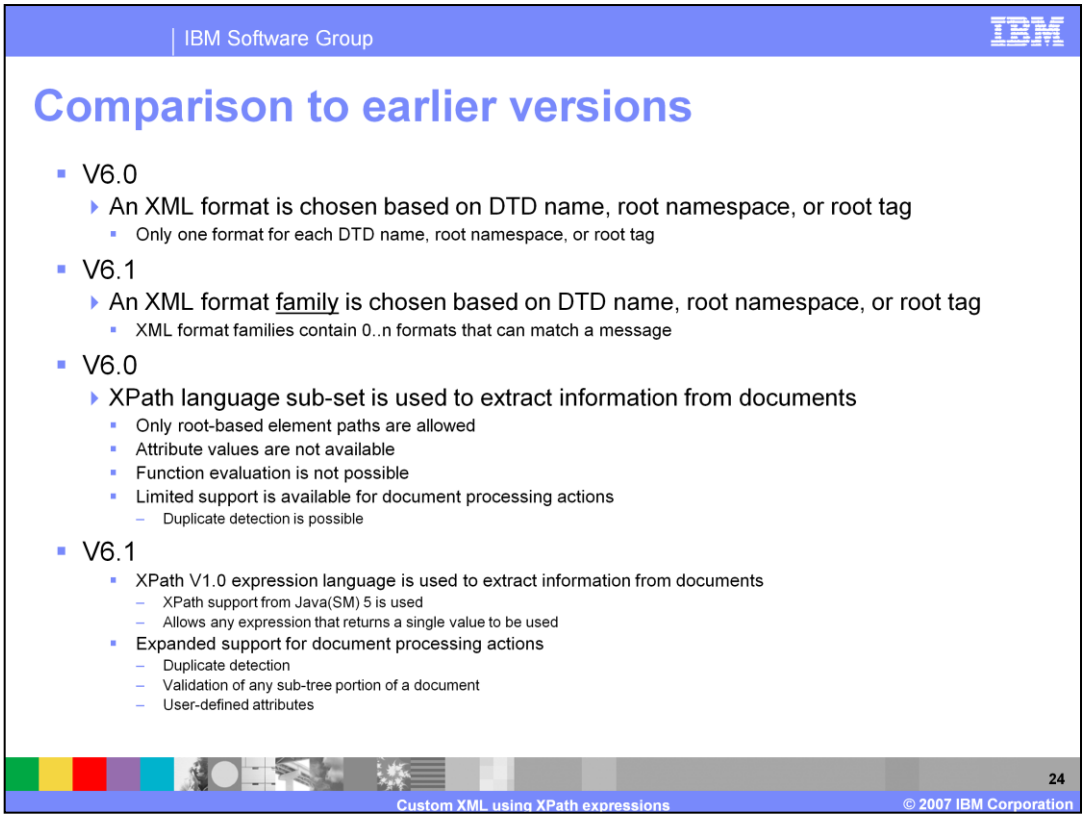
The remaining attributes are always optional to evaluate. So, if any of them have invalid expressions or expressions that do not return results from a document that otherwise matches an XML format, no errors are logged.



If you were familiar with the way that earlier versions of WebSphere Partner Gateway handled custom XML, it is helpful to see a comparison with the V6.1 functionality.

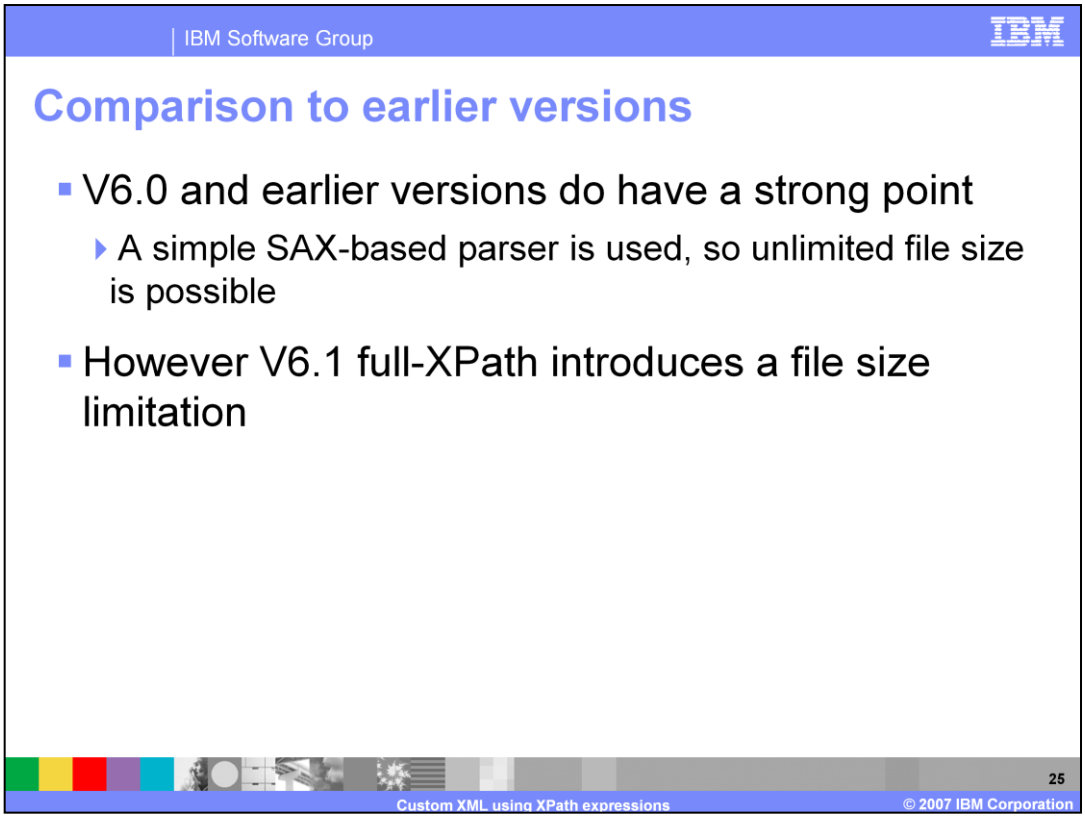
A main point brought out with this comparison is that in V6.1 it is still possible to use the old-style XPath evaluation code. This limits the use of some of the new V6.1 features like the validation root, and it restricts you to using a small subset of XPath expression language, but it allows you to handle arbitrarily large messages.

After discussing the large-file option, this section concludes with a review of some of the new attributes that are in V6.1.

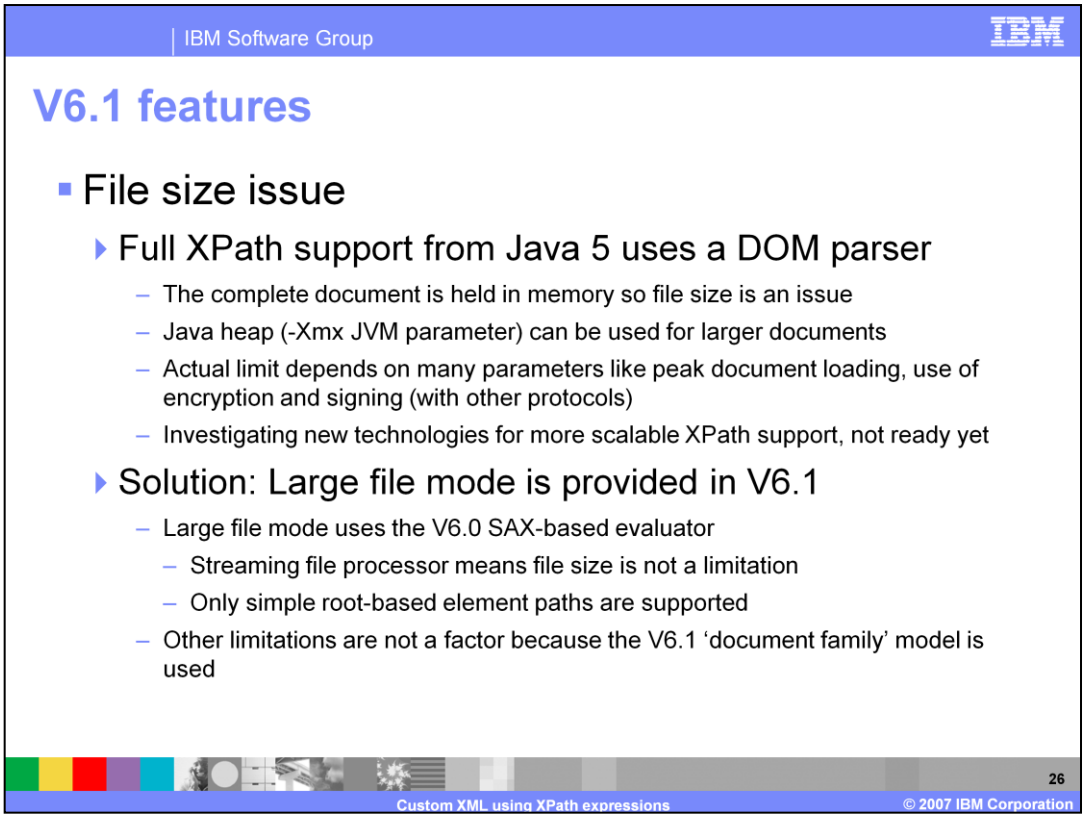


When the V6.1 custom XML features were added, the processing model was enhanced to allow ‘n’ formats to be associated with a given DTD name, root tag namespace, or root tag name. Previously only one format could be used for a given identifier. This provides support for enveloped messages, such as SOAP, for which all messages have a common root tag name and namespace.

Another enhancement in V6.1 is the use of the built-in XPath evaluation engine that is provided with Java 5. This allows very rich XPath version 1.0 compliant expressions to be used to extract data from documents. Before only simple root-based path expressions were possible.

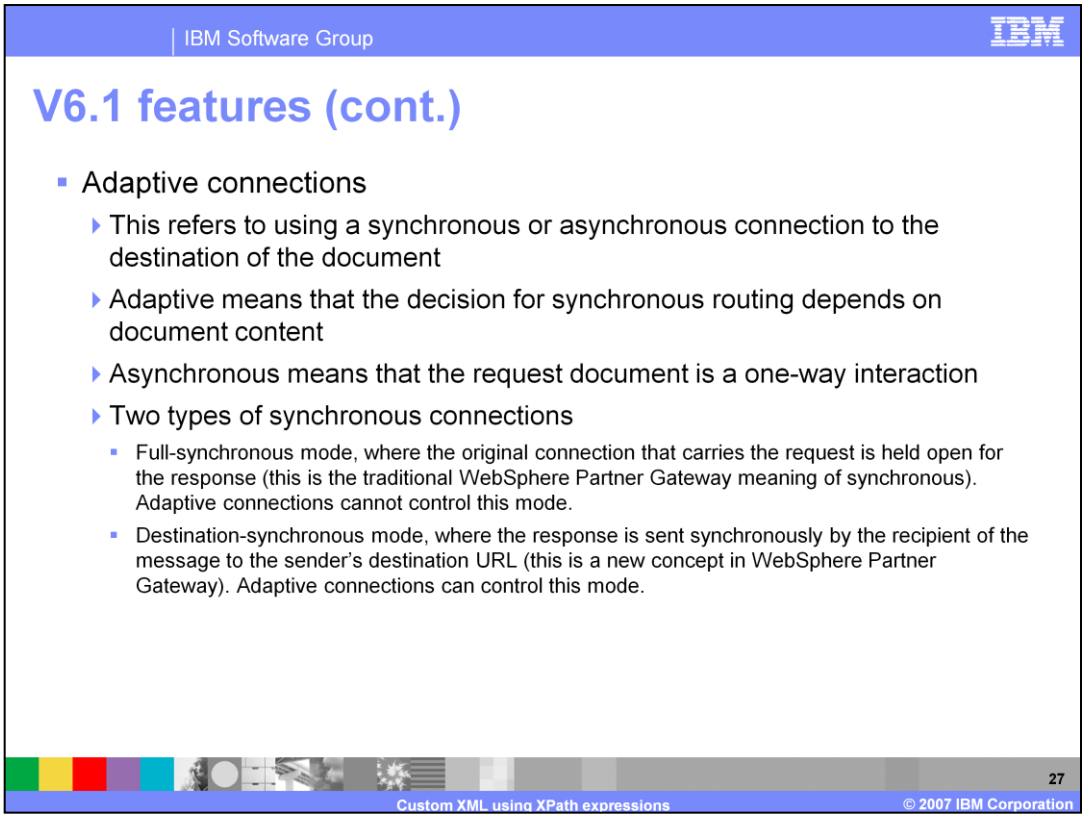


The power of XPath V1.0 compliance comes with a cost. Since XPath version 1.0 evaluation requires a DOM object to operate, the memory footprint of V6.1 processing can become a problem with large documents. The previous XPath engine was SAX based, and as such it used a streaming processing model that allowed unlimited document sizes to be processed without the danger of out-of-memory errors.



Since file size can be an issue with large documents, you should test with simulated production loads to understand what needs to be done about setting Java heap size appropriately and otherwise setting up the system for production.

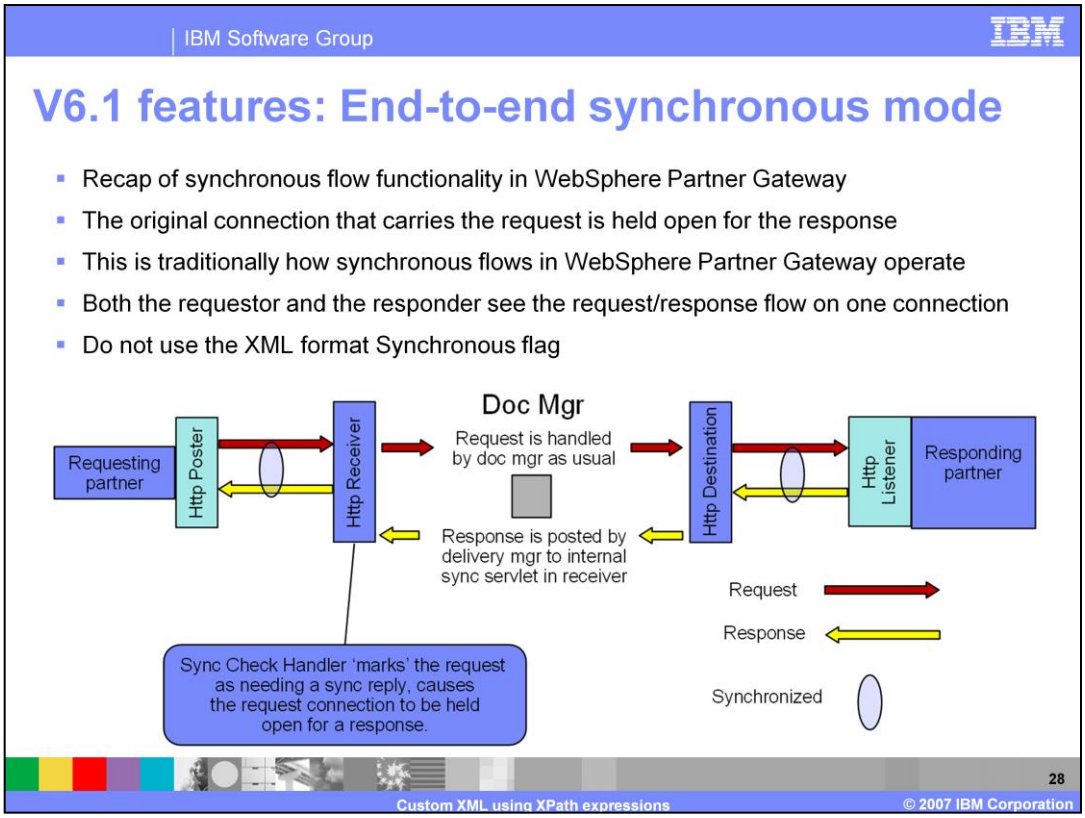
The immediate answer to huge documents is to use the large-file option that is provided in V6.1. This uses the original V6.0 and earlier streaming path evaluator. It limits you to using simple root-based path expressions and restricts some of the attributes that are available, for example sub-tree extraction is not possible so the validation root is not available. The large-file option is supported in the new configuration model with format families and format matching.



In V6.1, the 'synchronous flag' attribute of an XML format provides you with a way to control whether a synchronous response from the destination will be handled by passing it the requestor's destination.

Note that this does not control the traditional synchronous connection that is an end-to-end synchronous connection through the hub. That level of control was not in the scope of the V6.1 XML format redesign.

The next slides explain the details of this feature.



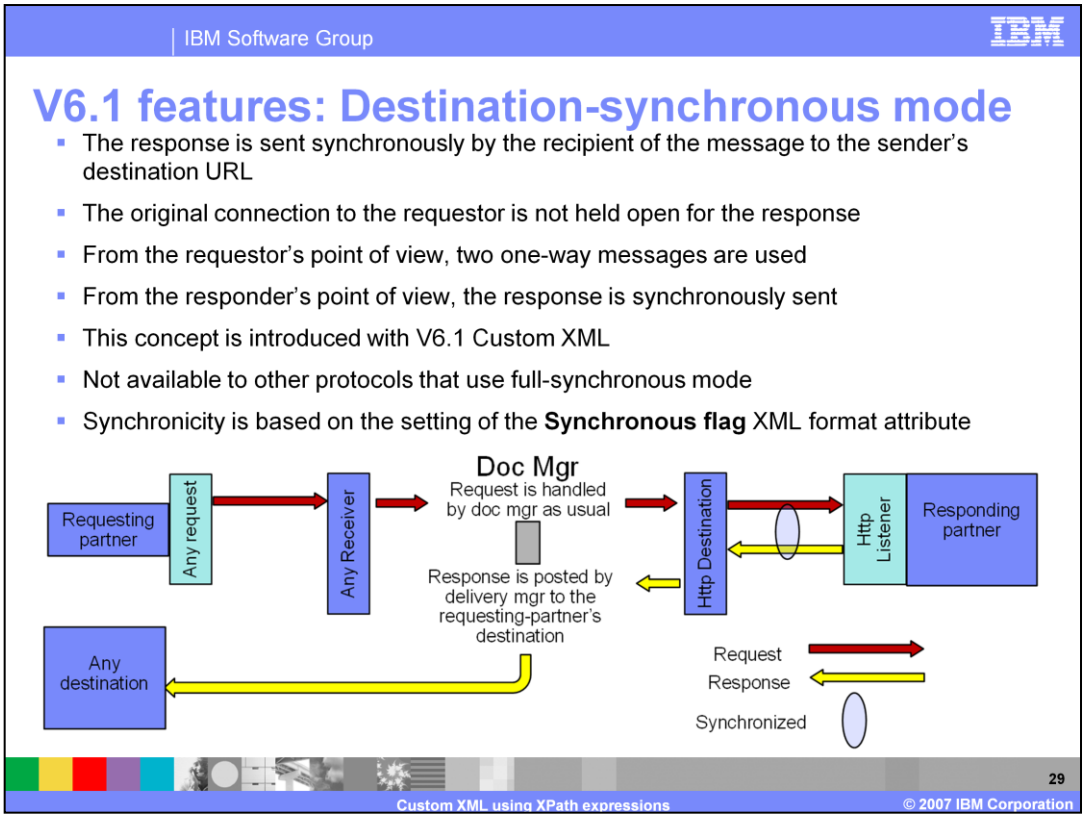
For an end-to-end synchronous flow through the hub, a transport capable of synchronous interaction is required. This example shows the use of the Http transport. The Http receiver for synchronous interactions has to have a sync-check handler installed. This means that for each request, the connection will be held open awaiting a response.

The sync-check handler sets a flag in the request document. When the flag is set, the delivery manager handles the request by sending it through the Http destination and then holds the connection awaiting a response from the endpoint it is sending to.

For the response to be handled synchronously, it has to have the from and to partner IDs switched relative to the request. Such a response that arrives back to the delivery manager on the original request connection is handled by passing it back to the receiver using an internal sync servlet.

The receiver is expecting the response, and it will look up the connection that is being held for it. It will send the response back to the requestor on the same connection that the request was received on.

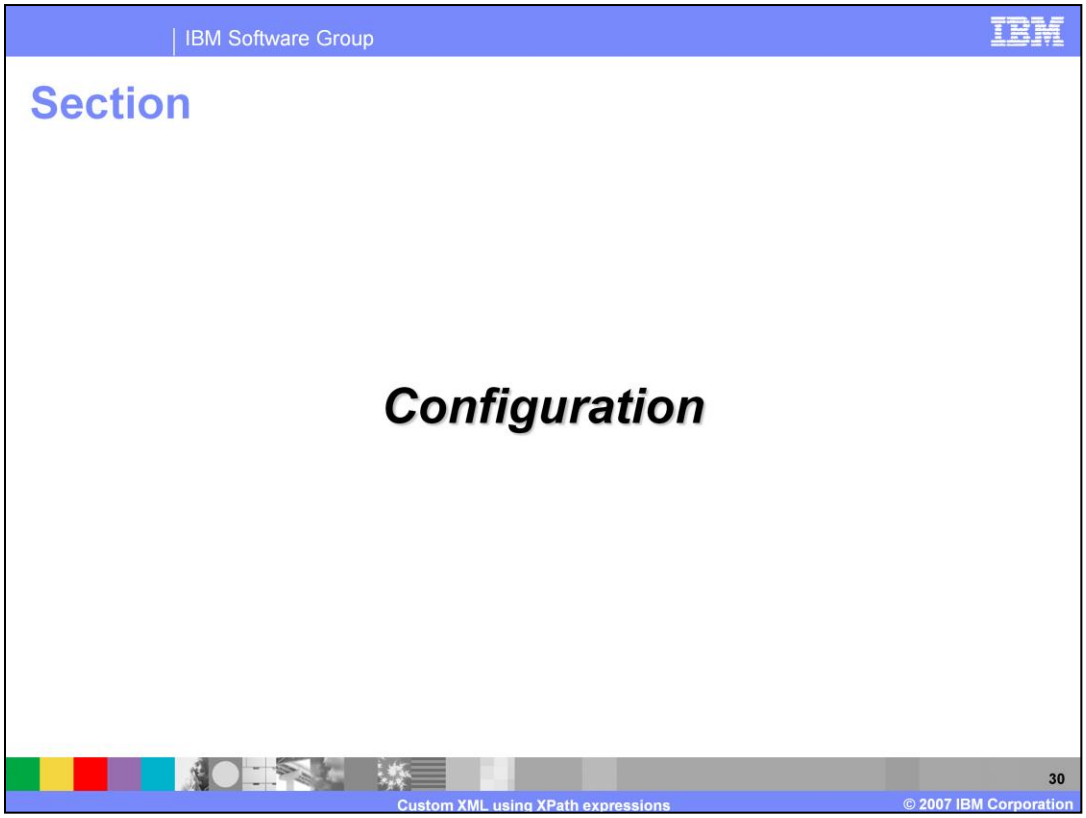
To be clear, this mode is not controllable by using the new synchronous flag, but another type of synchronous interaction can be controlled. That is described on the next slide.



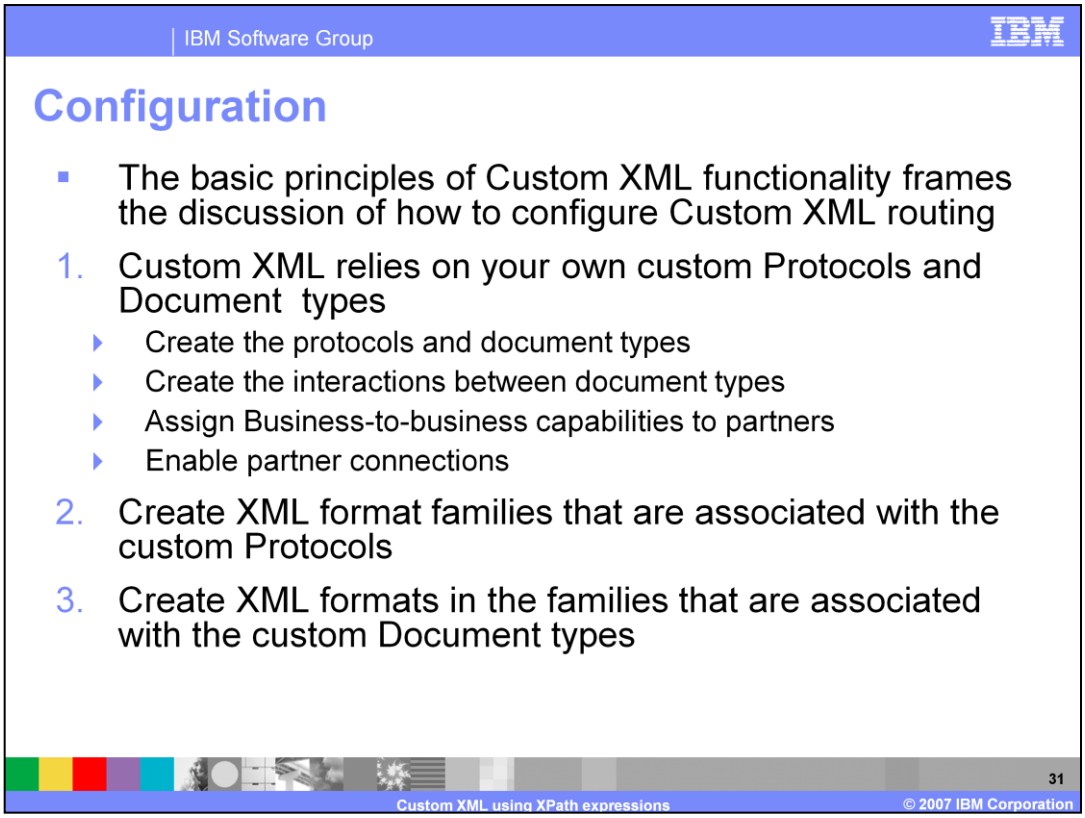
The synchronous flag attribute in XML formats is able to control the synchronous behavior of a destination when the destination uses a transport capable of synchronous interaction. For this example, an Http destination is used.

Any request that arrives that is routed using an XML format can be handled synchronously by the destination. All that is necessary is for the XML format 'synchronous flag' attribute to be set to the value 'true' and for the destination to use a synchronous transport. For this example, an XML request is received using any receiver and the 'synchronous flag' is set to 'true'. The request is delivered using a Http destination, and the delivery manager sees that the flag was set true, so it holds the connection awaiting a response.

When the response is received, the delivery manager will determine the requestor's destination as set on the connection that was used for the request. The response will be delivered to this destination, no matter what the transport for the destination is.

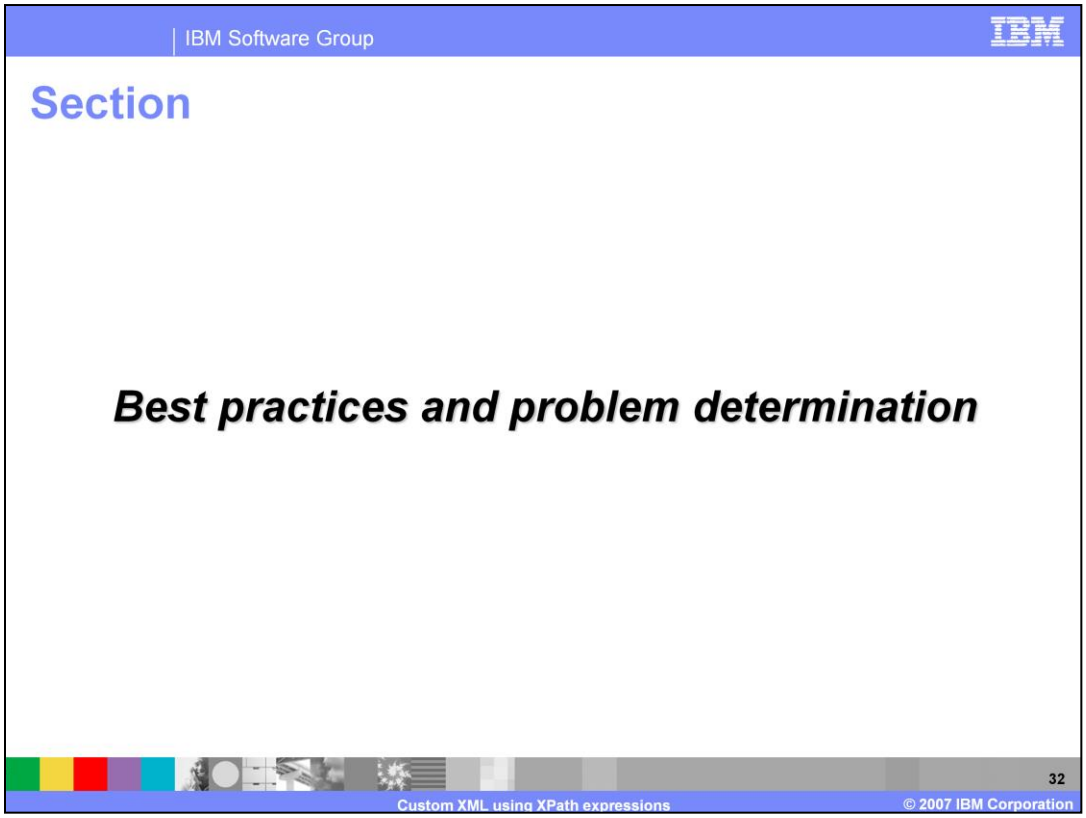


In the configuration section you will see how the configuration screens are used to create XML format families and XML formats.

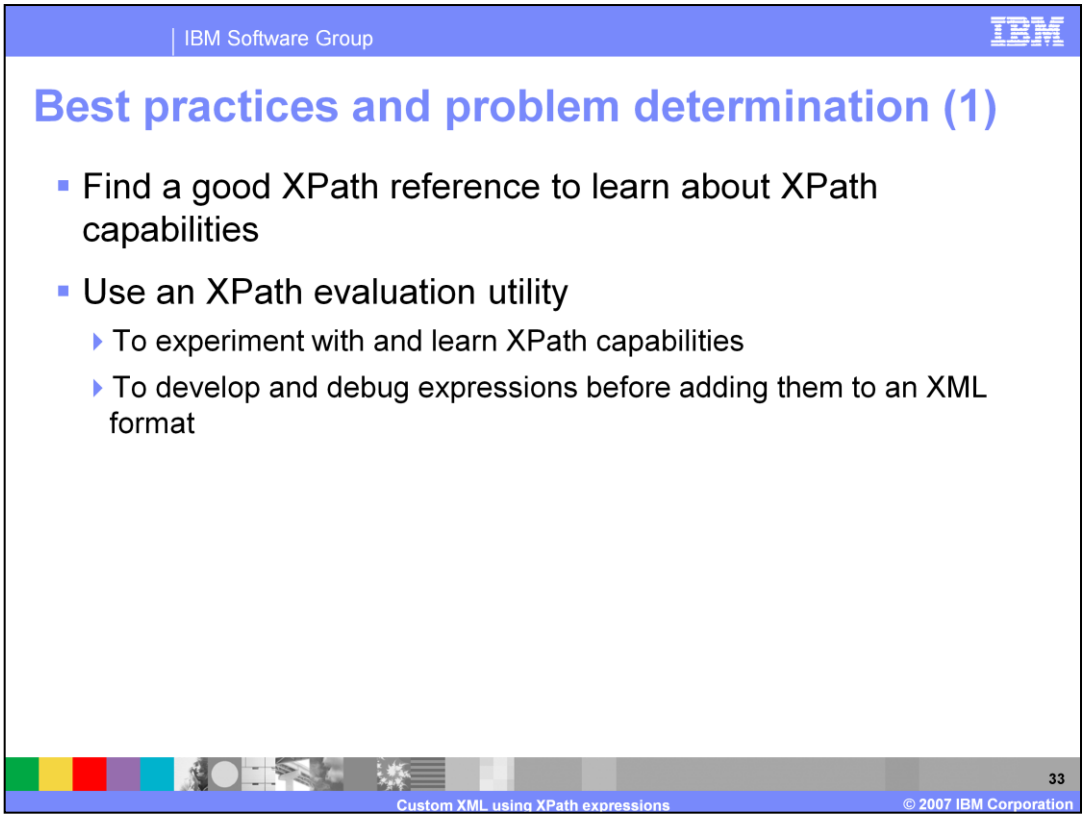


To use custom XML routing, you need to create protocols and document types that will be selected by the system to route documents that are received at runtime.

After you create the protocols and document types, you create XML format families that are associated with the protocols. Then for each XML format family, you create an XML format for each document type in the protocol that is associated with the family. The XML formats each have a set of XPath expressions that are evaluated in the context of documents until a format is found that matches a document. Remember that once a format is matched to a document, the document type for routing is the type associated with the format, and the routing protocol is the protocol associated with the format's family.



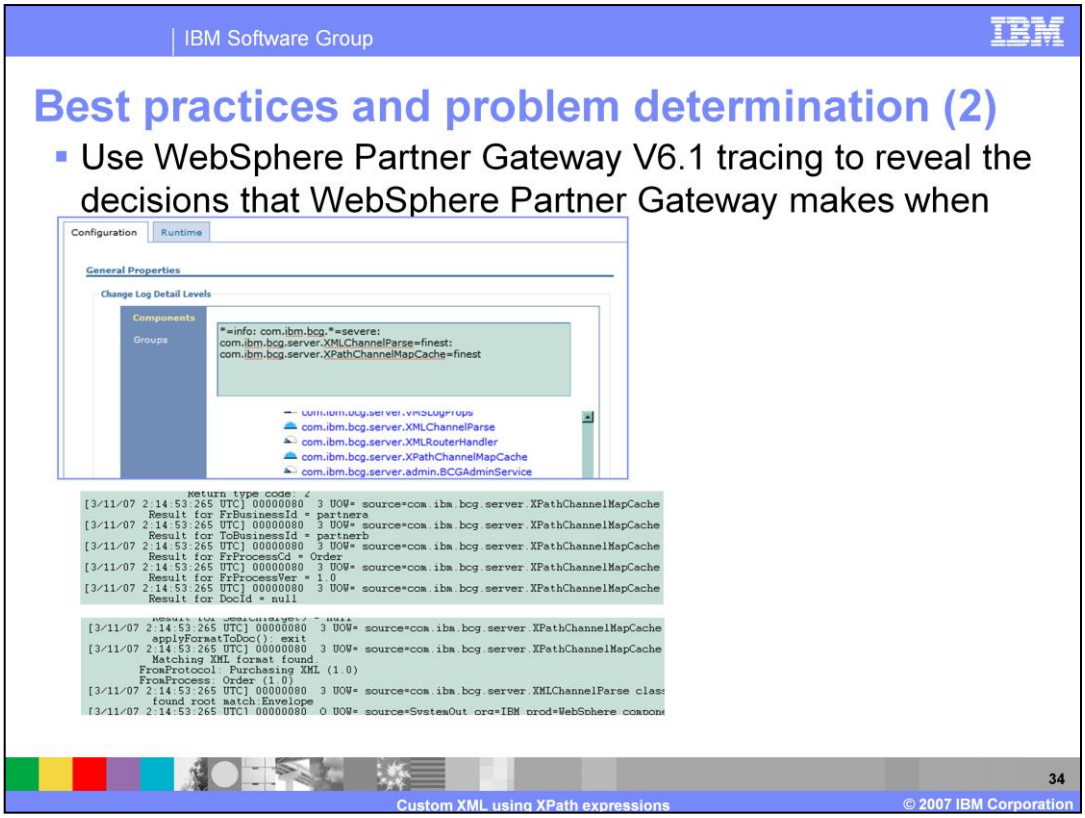
The next section covers the best practices and problem determination.



XPath version 1.0 is a rich expression language. The more you learn about it, the more you can take advantage of its capabilities. Typically people learn XPath as a way to write XML style sheets, and they don't learn it as a stand-alone tool. This means that many of the learning resources for XPath are found in the context of using XPath for style sheets. Nonetheless, there are numerous resources on the Web and in print that you can use to become proficient.

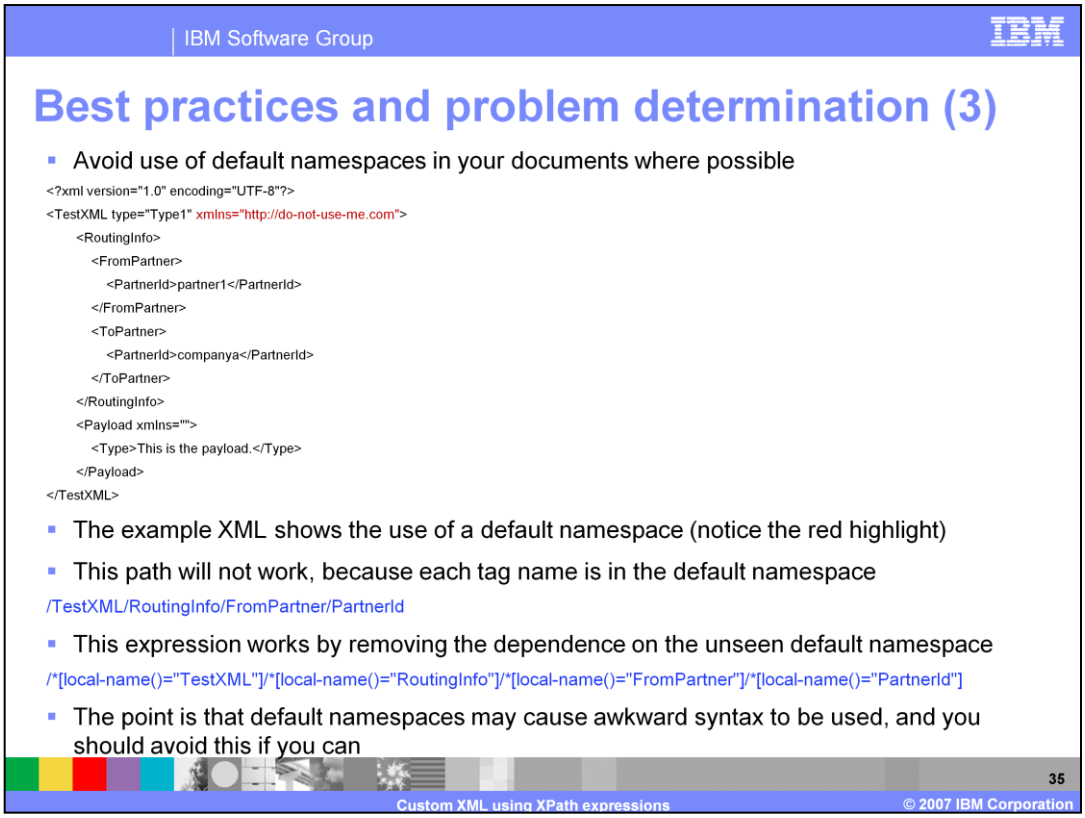
It can be difficult to develop the expression that does what you expect it to do. One thing you do not want to do is to develop expressions by trying them in XML formats and routing documents to test them. This is time consuming, ineffective, and frustrating

Instead, you should obtain and use a utility program that can take a sample XML file as input and allows you to evaluate expressions against the XML document. Rational Application Developer provides this capability in the context of developing XPath expressions for style sheets.



If you turn on finest tracing for the classes com.ibm.bcg.server.XPathChannelMapCache and com.ibm.bcg.server.XMLChannelParse you will get tracing that shows the search and selection of an XML format for a document.

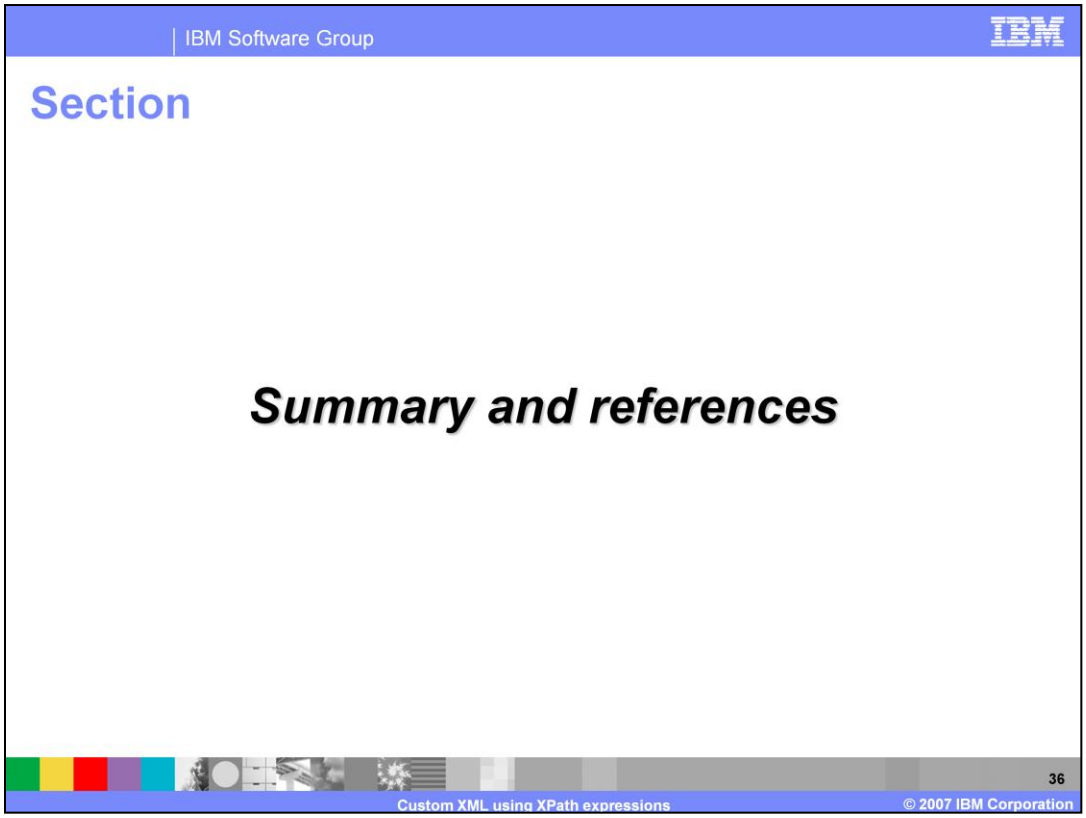
The trace example in the slide shows how this is specified in the WebSphere Administrative Console. The tracing for a match of a format to a document is also shown in the slide.



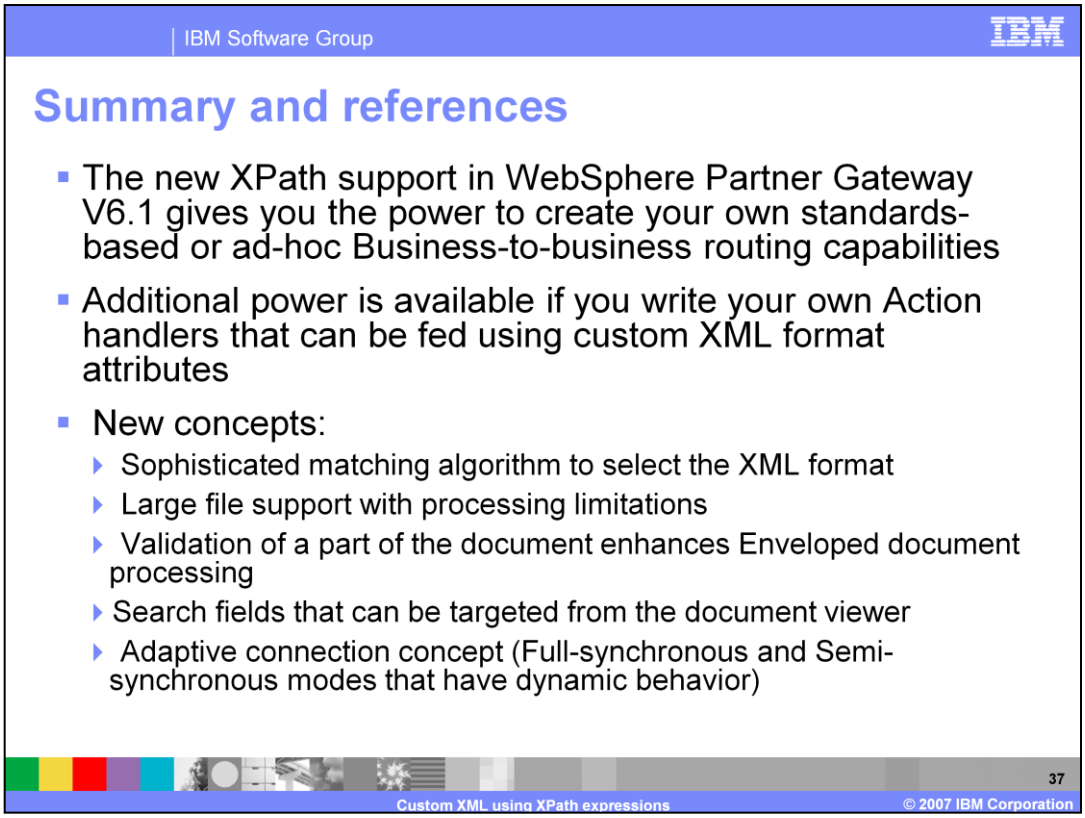
Default namespaces and XPath version 1.0 are not friends. In fact, in order to use default namespaces in your XPath expressions, you need to resort to a trick that uses the local-name XPath function.

The trick is shown in the slide for an example document.

If possible, you should use explicit namespaces instead of default namespaces in your XML messages.



The next section covers the summary and references.



In summary the enhanced XPath support in V6.1 gives you the power and flexibility to create your own standards based on ad-hoc Business-to-business routing capability. You can view the show-me demos on adding new XML formats and configuring to use custom XML in WebSphere Partner Gateway V6.1.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere

Expression, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, JVM, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.