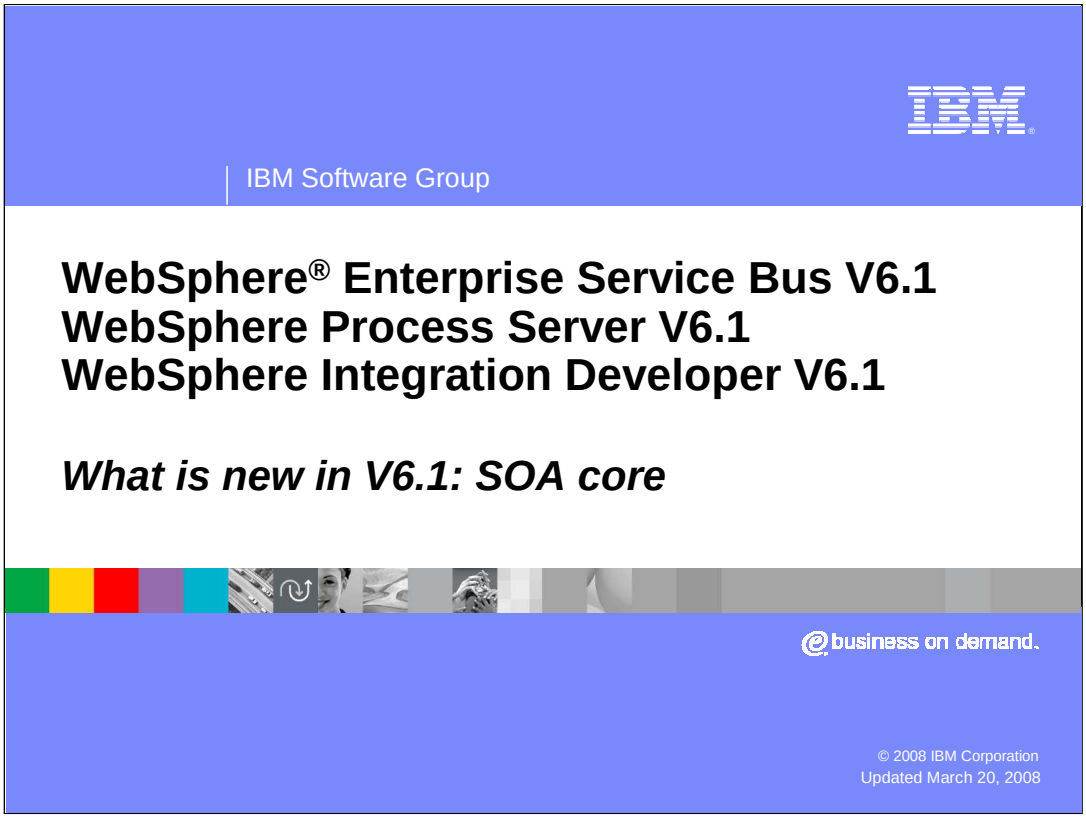
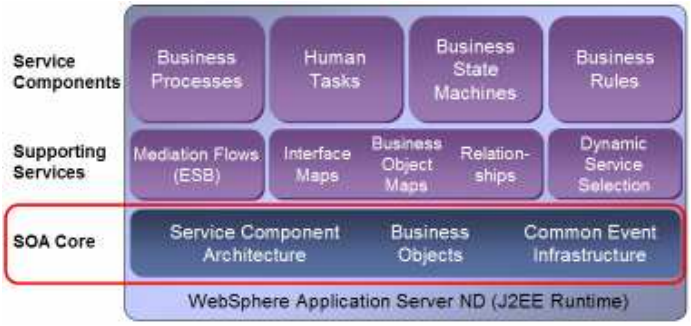


This presentation provides an introduction to the new function delivered in version 6.1 for the SOA core portion of WebSphere Enterprise Service Bus, WebSphere Process Server and WebSphere Integration Developer.

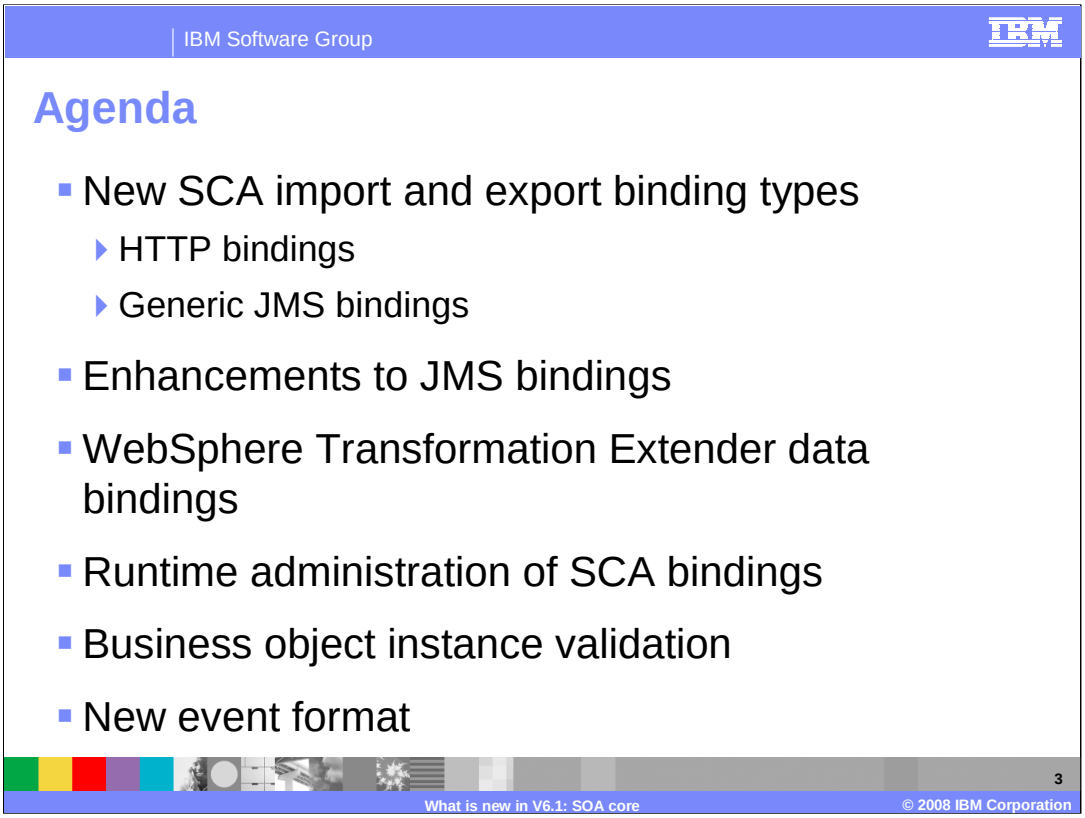
Goals



- To introduce enhancements to SOA Core made for V6.1
- Applies to:
  - WebSphere Enterprise Service Bus
  - WebSphere Process Server
  - WebSphere Integration Developer
- Prerequisites to understanding this presentation:
  - Knowledge of SOA Core in V6.0.2

The goal of this presentation is to introduce you to the enhancements that have been made to the service oriented architecture core for version 6.1. As you can see in the architecture chart, the SOA core is composed of the service component architecture, business objects and the common event infrastructure. These enhancements apply equally to WebSphere Process Server and WebSphere Enterprise Service Bus, including new and changed capabilities in WebSphere Integration Developer which support these enhancements from a development tool perspective.

In order to understand the material in this presentation you should already have a knowledge of the capabilities of SOA core in version 6.0.2 of the products.



Most of the enhancements relate to the service component architecture portion of the SOA core. There are new SCA binding types that have been introduced for supporting HTTP and HTTPS protocols and for supporting JMS in a generic, provider independent manner.

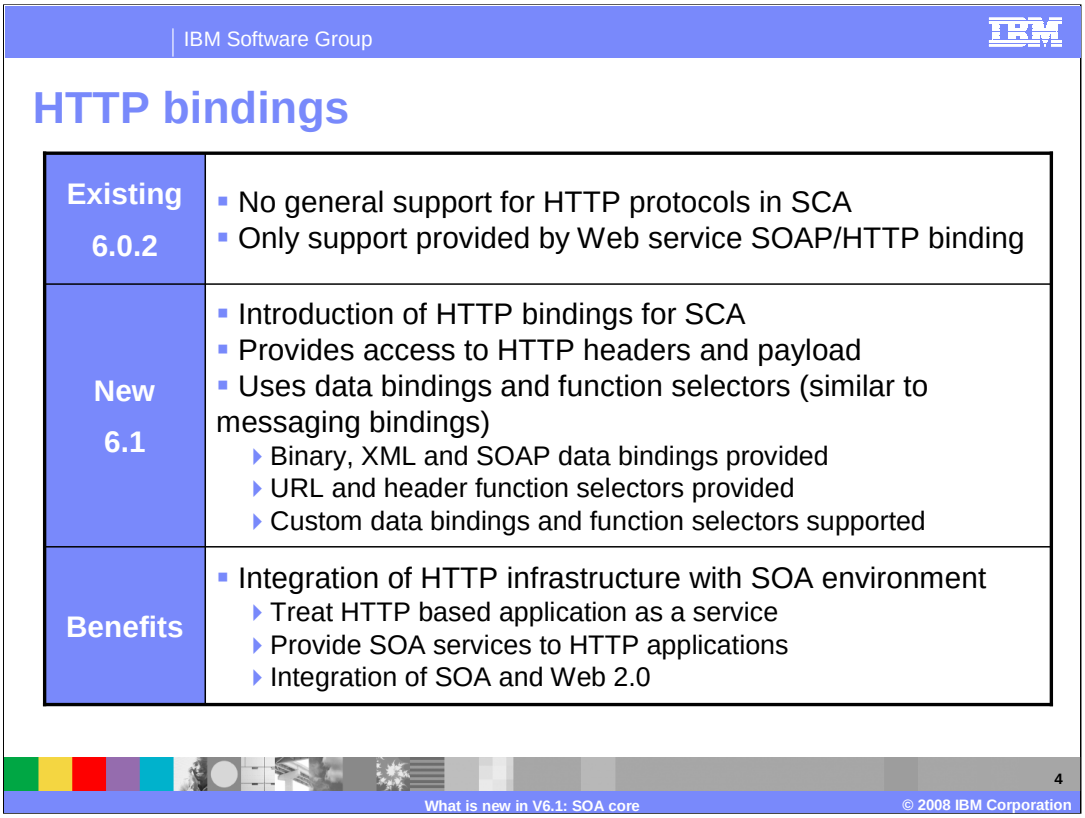
The existing JMS bindings used with the default JMS provider have a few incremental enhancements.

Most SCA binding types make use of data bindings to perform data conversions. The ability too integrate with WebSphere Transformation Extender through a data binding has been added to enable use of its excellent data transformation capabilities.

The ability to administer several of the SCA binding types at runtime has been added.

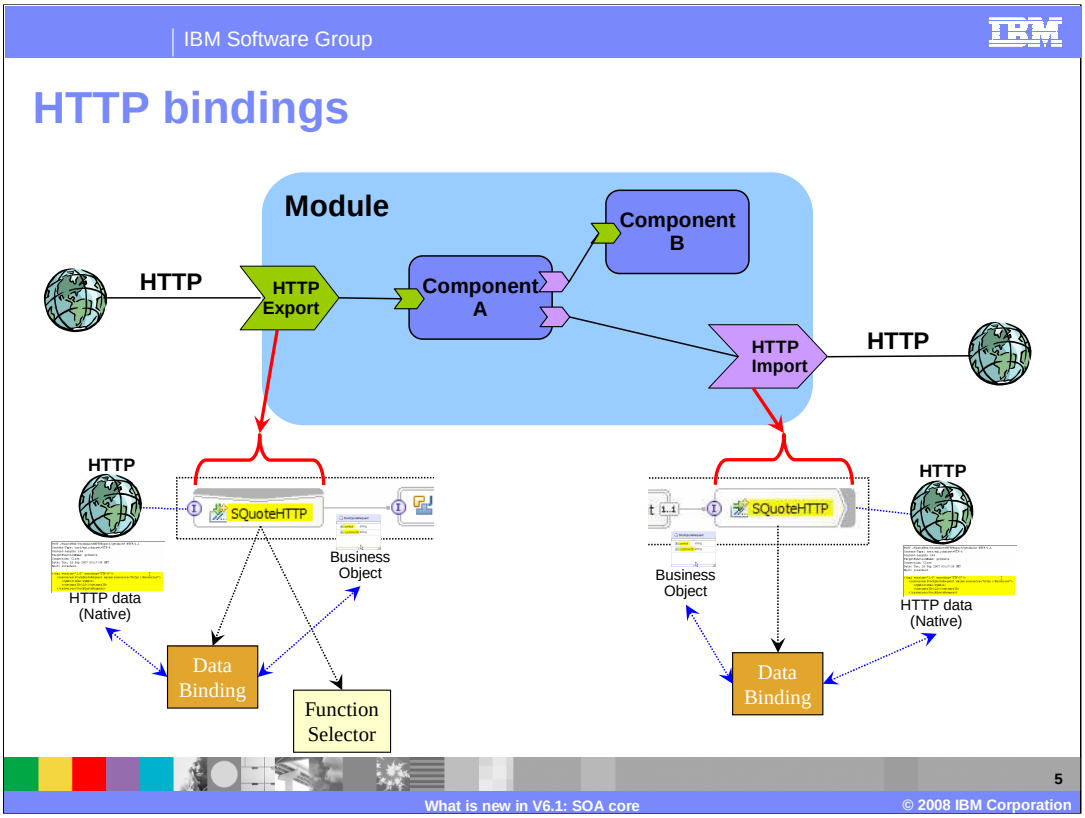
The enhancement for business objects is the introduction of a business object instance validation service.

Finally, the common event infrastructure has introduced a new event format which allows events to contain business data in XML.

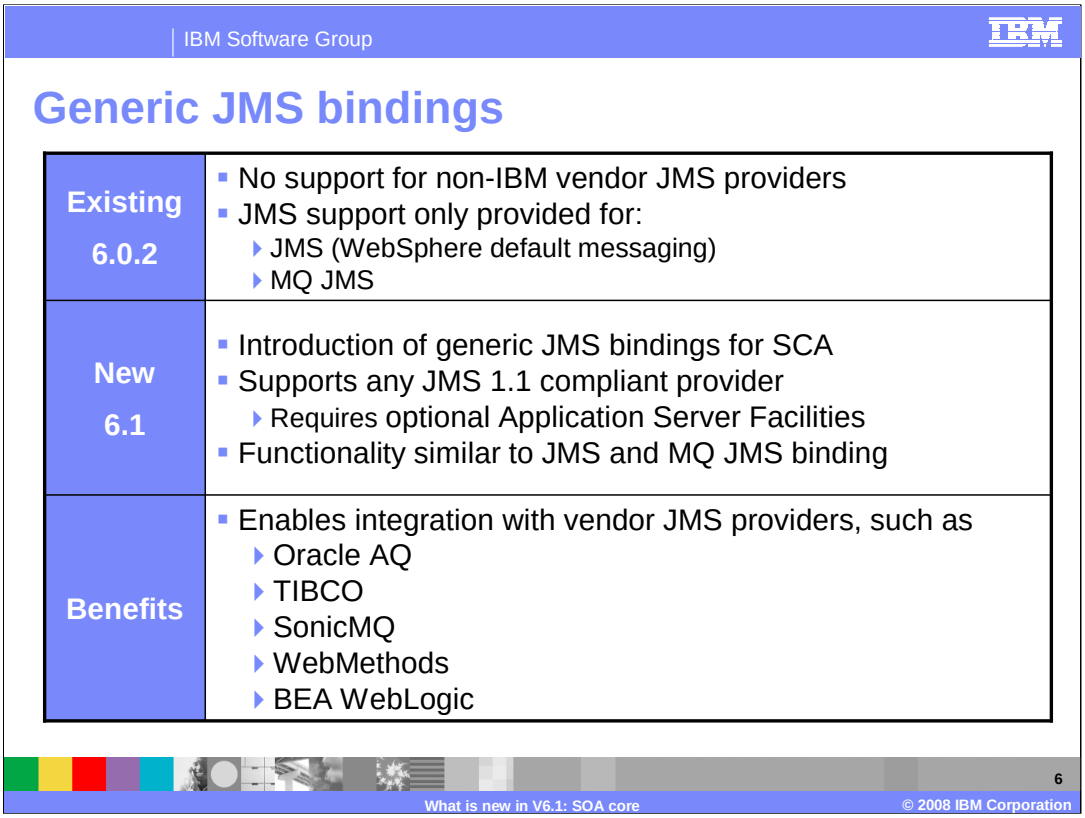


This enhancement provides generalized support for HTTP and HTTPS protocols in SCA bindings. In version 6.0.2 it was only through the Web services binding that HTTP was supported, requiring that the HTTP message contain a SOAP payload. With the introduction of the HTTP binding in version 6.1 the payload can be of any format and the header information is also available. The HTTP binding is similar to the various types of messaging bindings that make use of function selectors and data bindings. There are three data binding types provided for binary, XML and SOAP payloads and two function selector types which are based on a URL value or a value set in the header. Use of custom data bindings and function selectors is also supported.

Among the benefits of this enhancement is the ability to integrate your service oriented architecture infrastructure with your HTTP infrastructure, allowing you to treat HTTP based applications as services and to call services from your HTTP applications. Because Web 2.0 makes use of HTTP protocols, these bindings provide a solid mechanism to integrate your service oriented architecture infrastructure with emerging Web 2.0 applications.



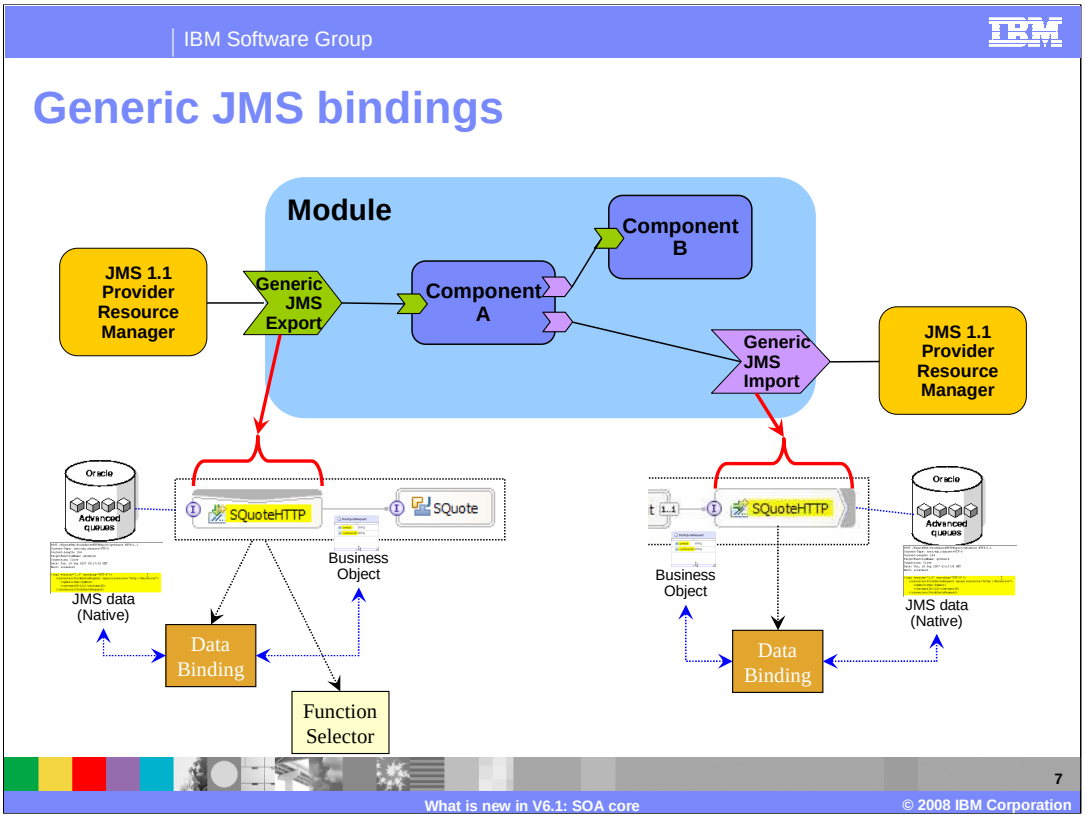
This is an illustration of an SCA module making use of an export and import with HTTP bindings. On the left side you can see the HTTP export which receives the inbound HTTP message and replies with an outbound HTTP message. The function selector configured for the export is responsible for determining which operation to call on the SCA component the export is wired to when an inbound message is received. On the right you can see the HTTP import which sends an outbound HTTP message and receives an inbound HTTP message in reply. Both the export and the import are configured with data bindings, which are responsible for conversion of the data. For inbound messages the conversion is from the native data format to the appropriate business object format. For outbound messages the conversion is from business object format to native data format.



In version 6.0.2 the support for JMS included two types of bindings. The first is the JMS binding used with the WebSphere default messaging JMS provider that comes with the server. The second is the MQ JMS binding that uses the JMS provider for WebSphere MQ. Version 6.0.2 provided no JMS bindings for support of other JMS provider implementations.

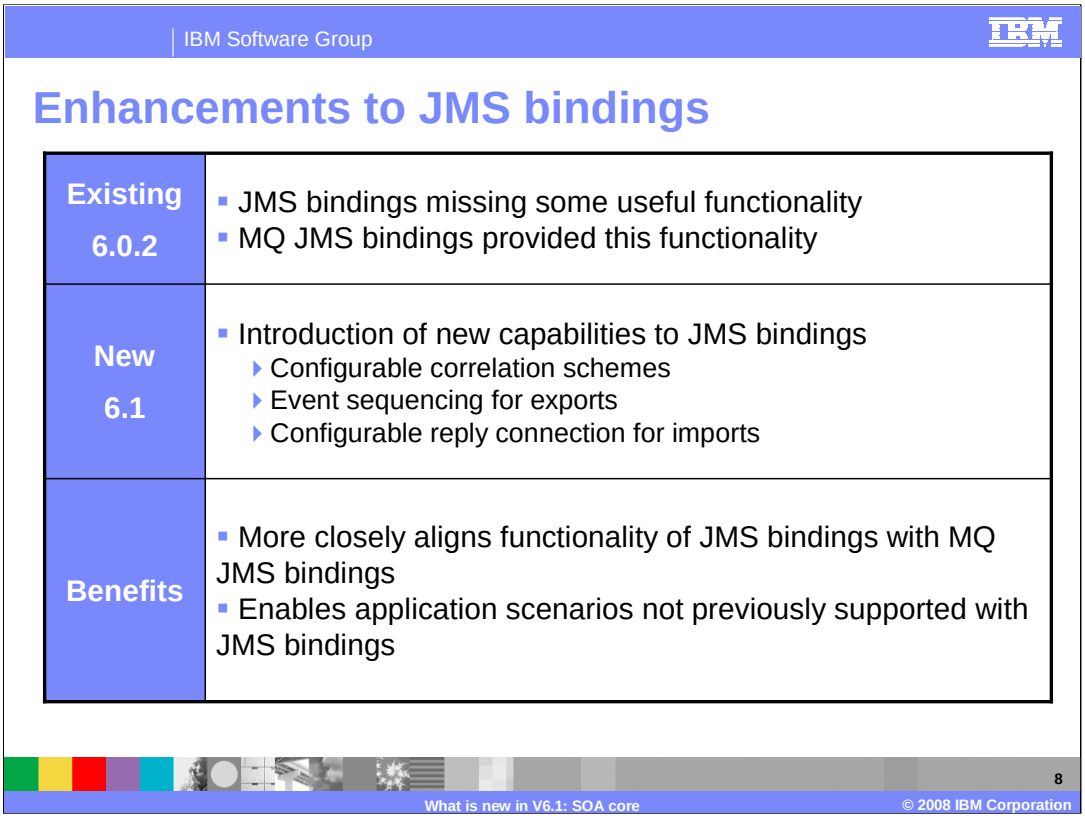
In version 6.1 the generic JMS binding was introduced. This allows you to integrate with any JMS provider that implements the JMS 1.1 specification, provided the implementation includes support for the optional application server facilities defined in the specification. The functionality provided is very similar to the already existing JMS binding types, making use of the same data binding and function selector implementations.

The benefit of this is enabling integration of your service oriented architecture with a variety of JMS providers. This can be particularly useful when you already have a messaging infrastructure investment based on one of these providers. Among the JMS providers that can be used with this support are Oracle AQ, TIBCO, SonicMQ, WebMethods and BEA WebLogic.



This is an illustration of an SCA module making use of an export and import with generic JMS bindings. The basic architecture of using data bindings and function selectors for the generic JMS binding is the same pattern used for other messaging bindings. You should also recognize it as the same pattern used for HTTP bindings shown on a previous slide.

On the left side you can see the generic JMS export which receives the inbound JMS message and replies with an outbound JMS message. The function selector configured for the export is responsible for determining which operation to call on the SCA component the export is wired to when an inbound message is received. On the right you can see the generic JMS import which sends an outbound JMS message and receives an inbound JMS message in reply. Both the export and the import are configured with data bindings which are responsible for conversion of the data. For inbound messages the conversion is from the native data format to the appropriate business object format. For outbound messages the conversion is from business object format to native data format.



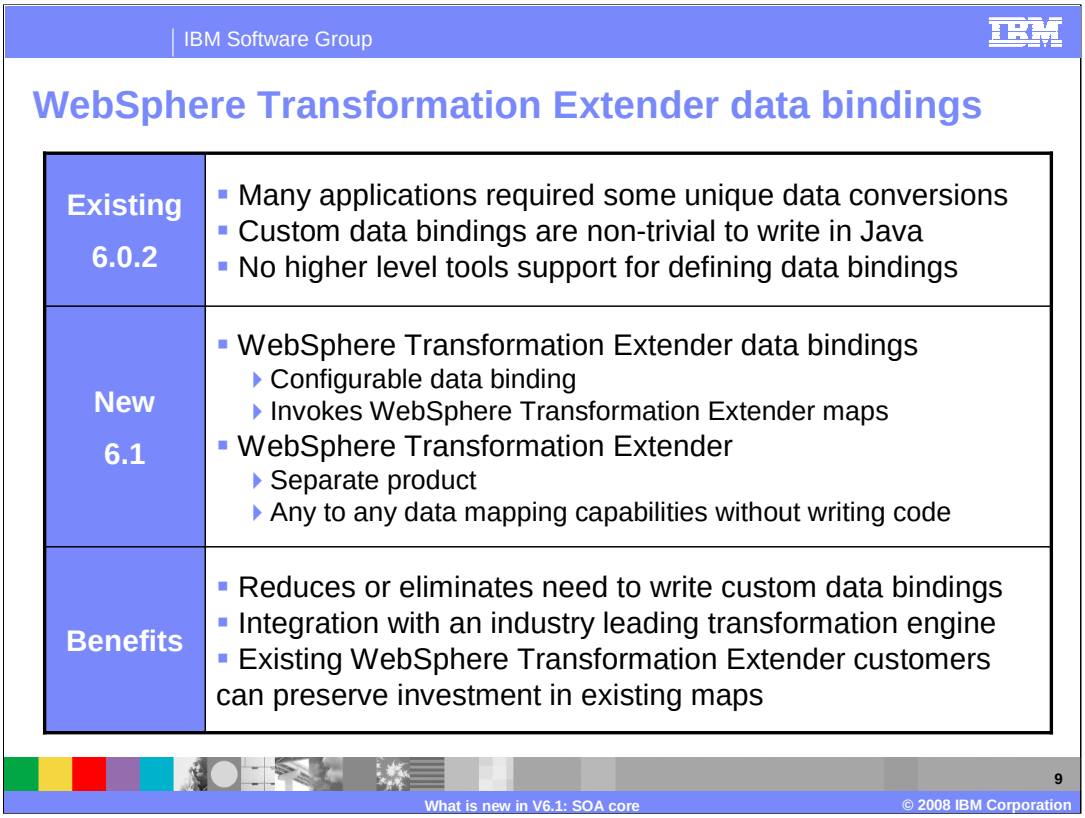
The enhancements described on this slide relate to the SCA JMS bindings that are used with the default messaging support built into the server. These enhancements are aimed at improving the JMS bindings with functionality that was already provided by the MQ JMS bindings. There are three new capabilities being added.

Configurable correlation schemes enable you to specify whether the request message identifier or the request correlation identifier is to be used for correlation of the request and response messages.

Event sequencing of exports allows you to require that messages received by an export are delivered in exactly the same order to the SCA component the export is wired to.

Configurable reply connections for imports enable you to specify an activation specification for the reply connection if the default connection does not satisfy your requirements.

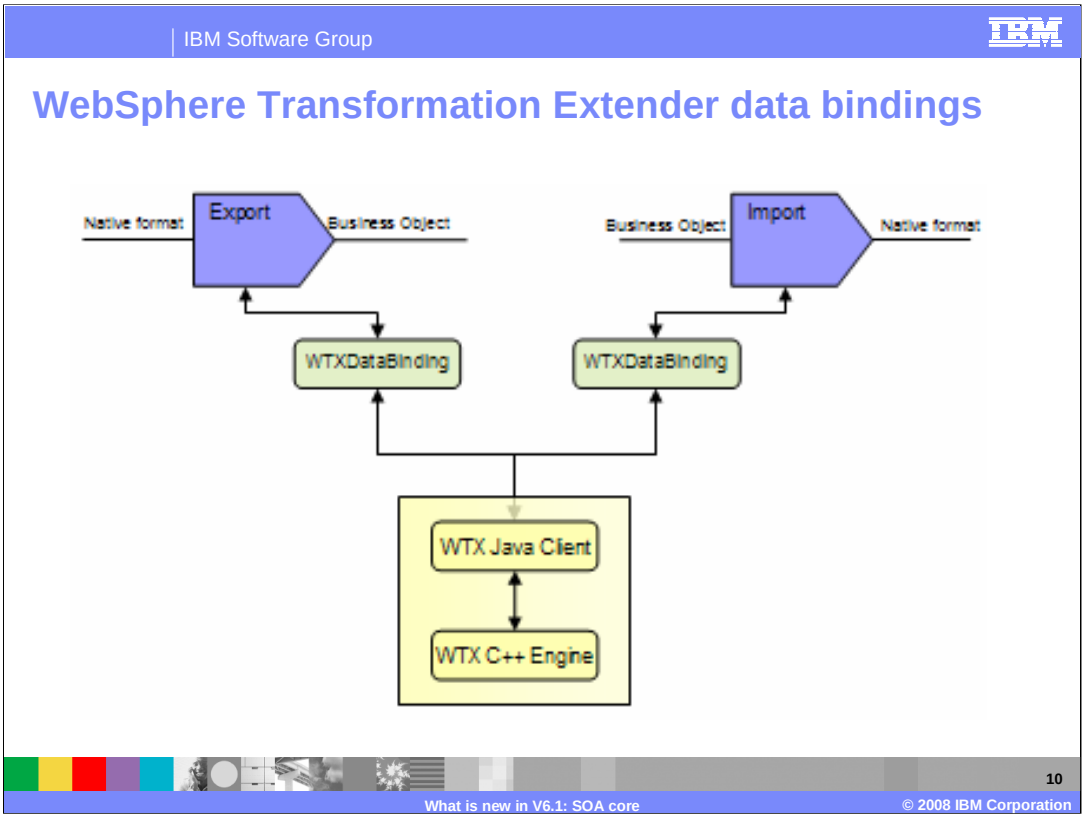
By providing these three enhancements, the JMS bindings are more closely aligned with the capabilities of the MQ JMS bindings. This enables some application scenarios that previously were not possible with the JMS bindings.



SCA binding types other than default SCA and Web services bindings require the use of data bindings to convert between native and business object formats. Each SCA binding type provides a few data binding implementations, but for most application scenarios these are not sufficient because application specific knowledge of both the business objects and native data is required. Therefore, it is often the case that custom data bindings need to be written. Unfortunately, writing custom data bindings in Java is not trivial and there are no high level tools provided in WebSphere Integration Developer for generating them.

In version 6.1 this situation is addressed by providing integration with WebSphere Transformation Extender through the use of a new data binding. WebSphere Transformation Extender is a separate product which specializes in data transformation, providing a high level map definition facility and a runtime transformation engine. You can now use WebSphere Transformation Extender to create maps which define your required data transformations and configure the data bindings to make use of these maps.

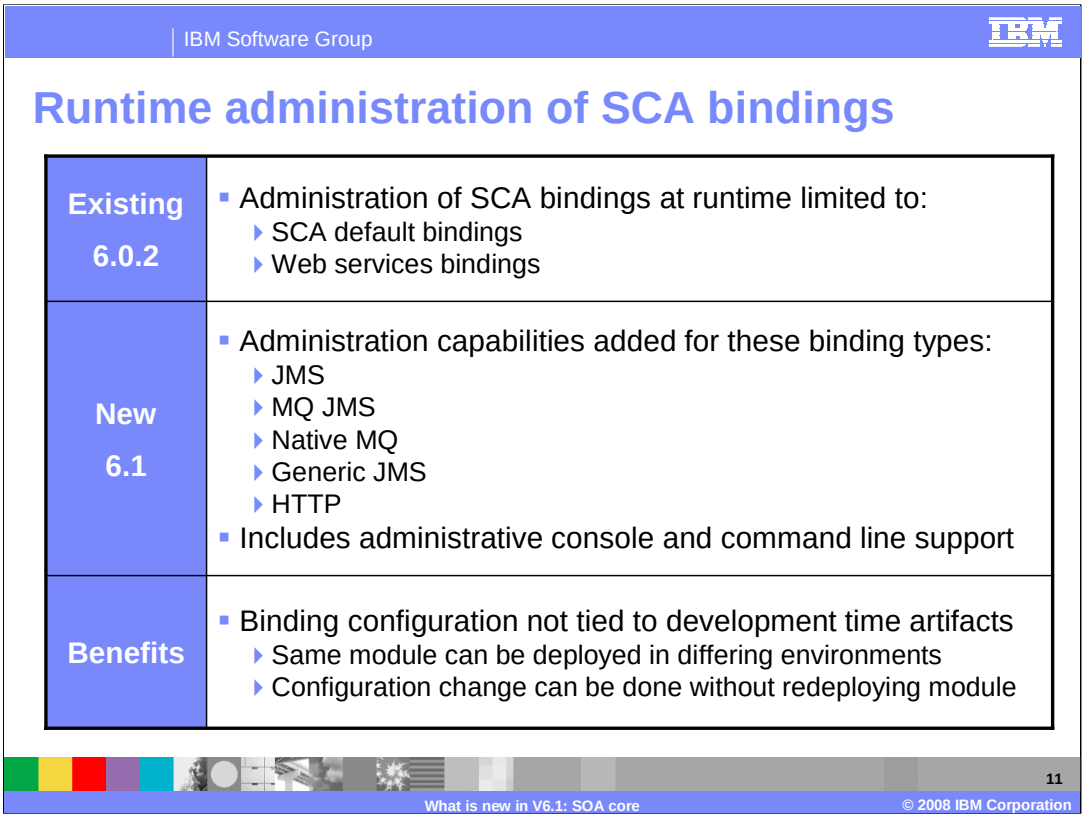
The benefit from this is the reduction and possible elimination of cases where you have to write custom data bindings in Java. You can define your data conversion maps using WebSphere Transformation Extender and then use them directly from your WebSphere Process Server or WebSphere Enterprise Service Bus server. In addition, if you have an existing investment in WebSphere Transformation Extender maps you can continue to use them with WebSphere Process Server or WebSphere Enterprise Service Bus.



This is an illustration of the interaction between WebSphere Process Server or WebSphere Enterprise Service Bus with the WebSphere Transformation Extender transformation engine.

Not actually shown on this picture are the maps which are defined using the WebSphere Transformation Extender tools. The maps must be defined to perform conversions to and from the native format and serialized XML.

As you can see in the diagram, the export and import are configured with a WebSphere Transformation Extender data binding. This data binding performs conversions to and from business objects and serialized XML. Through the use of some configuration information it determines the name of the map to be used and calls the WebSphere Transformation Extender Java client passing it the data and map name. The Java client then uses JNI to call the transformation engine which runs in the same process as the WebSphere Process Server or WebSphere Enterprise Service Bus. The transformation engine interprets the map and performs the transformation which is returned up the stack to the data binding.

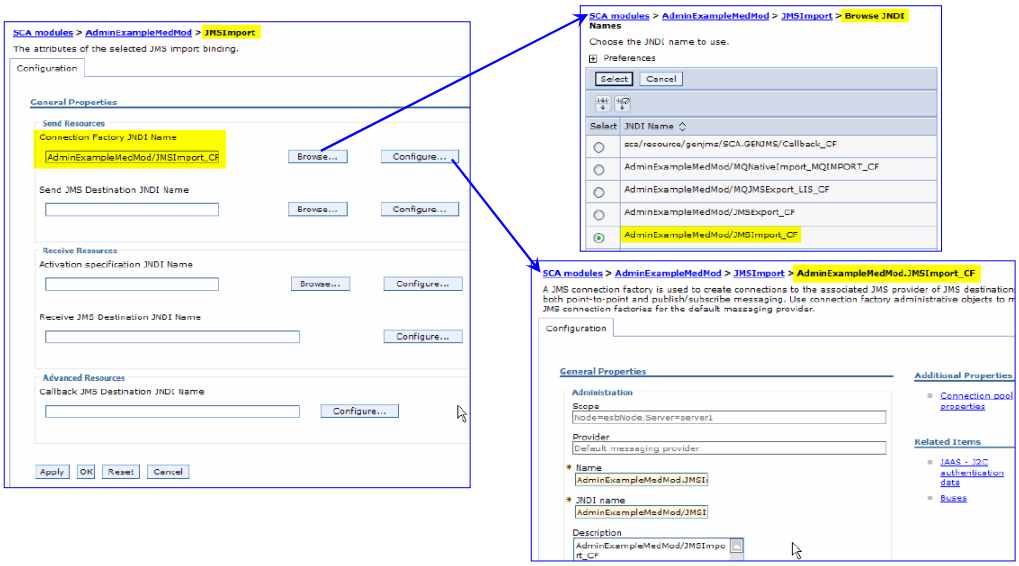


The configuration for SCA bindings is normally done at development time using the WebSphere Integration Developer, but there are often times when configuration information specific to a particular runtime environment is needed. In version 6.0.2 there are only two SCA binding types which can be administered at runtime, the default SCA bindings and Web services bindings. In version 6.1 the capability for runtime administration of bindings has been expanded to cover all of the messaging binding types and the new HTTP bindings. The administrative capabilities include specifying configuration information during application installation or dynamically for an installed application. Both of these capabilities are available through the administrative console and through command line support.

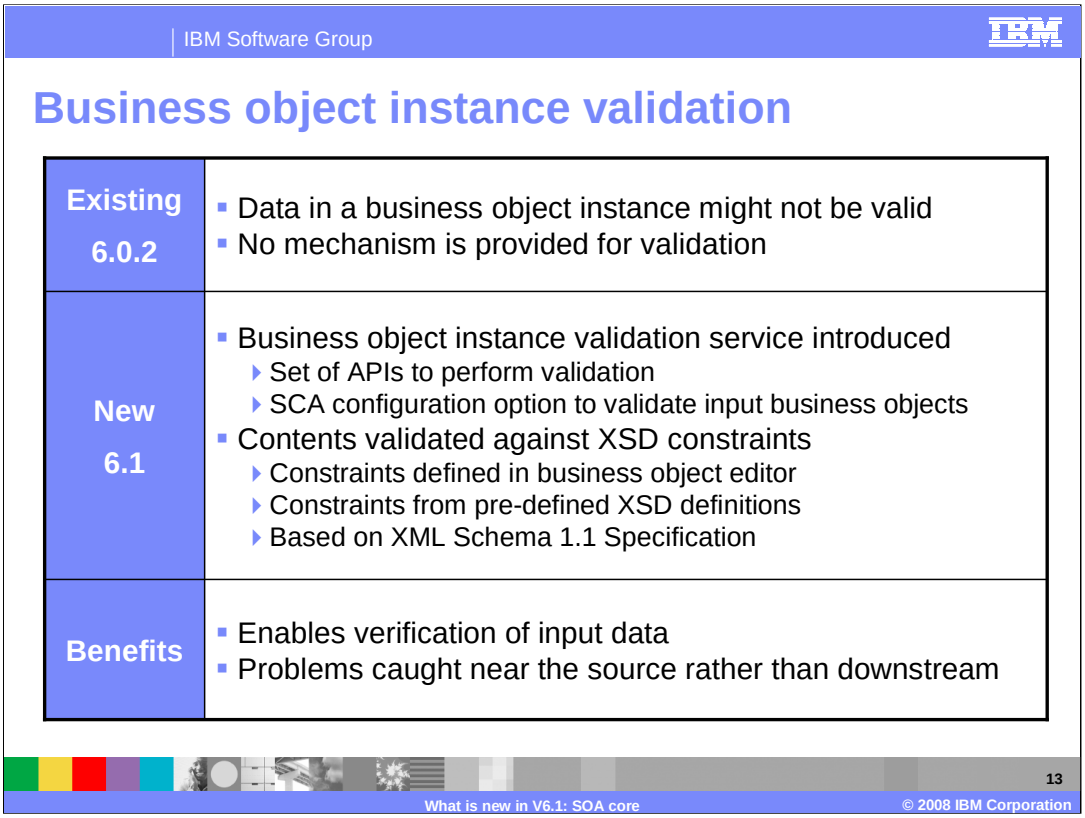
Using runtime administration allows a module exported from WebSphere Integration developer to be deployed into multiple environments where the configured resources have different names or configuration details. Without runtime administration, the module needs to have separate copies maintained in WebSphere Integration Developer with each exported separately.

Another benefit occurs when an environment changes and an existing module needs to be changed accordingly. Without runtime administration the updates have to be made in WebSphere Integration Developer and the module exported and reinstalled into the server. With runtime administration, the installed application can be updated dynamically without having to change and redeploy the module.

Runtime administration of SCA bindings



This is an example of the runtime administration panels in the administrative console, using JMS bindings as the example. The panel on the left lists the various resources that can be configured administratively, for example, the connection factory. These resources are identified using their JNDI names. The Browse… button can be used to get a list of JNDI names for resources of that type. Once a JNDI name is selected, the Configure… button can be used to access a configuration panel specific to that resource instance.



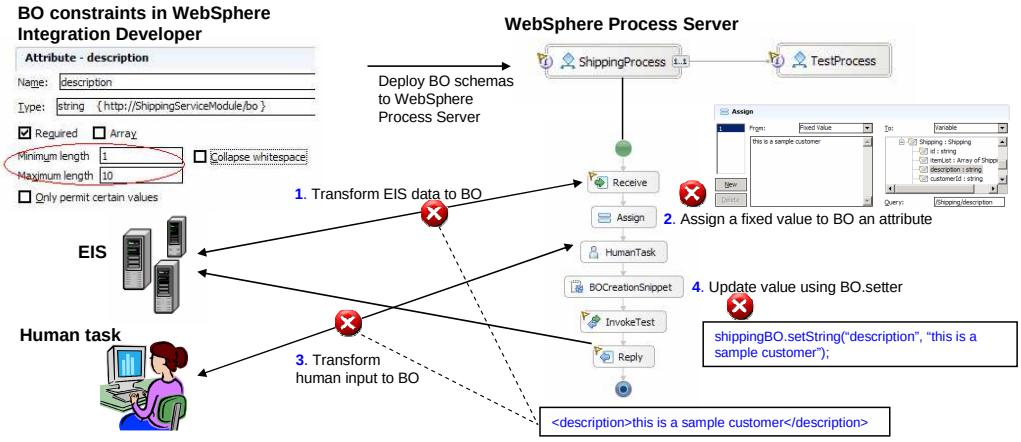
The next enhancement to look at provides new support for business objects. Within the server, an instance of a business object can contain data that does not conform to the XSD constraints defined for that business object type. In version 6.0.2 there is no mechanism provided to check a business object instance to ensure the data it contains is valid.

In version 6.1 a new business object validation service has been introduced. This service takes on two forms. The first is a set of APIs which can be called from your Java code and the second is an SCA configuration option to validate business object instances on SCA component boundaries. The validation is based on the XML Schema 1.1 specification. The validation checks the data contained in the business object instance against the constraints defined for it in the XSD. These constraints might have been specified using the business object editor in WebSphere Integration Developer or possibly were contained in pre-defined XSD definitions which were imported.

The benefit of this service is that data input into the system as a business object can be validated to ensure it conforms to the definition. This enables problems to be identified near their source and with explicit error indicators. Without this, the problems often do not arise until further downstream in the process, and can possibly produce errors which do not clearly indicate the source of the problem is the data in the business object instance.

Business object instance validation

- Illustration of the problem



This slide illustrates how a business object instance can potentially contain data that does not conform to the constraints of its XSD. If you look in the upper left corner you can see the business object editor. It defines constraints for an attribute named description which is of type string. This attribute is required and must contain a string of at least one character long and no longer than 10 characters. In WebSphere Process Server this business object is being used in a business process which is shown on the right. This illustrates four possible ways the business object instance might contain data which does not conform to this constraint. Cases 1 and 3 on this chart show data coming into the business process from the edges. Case 1 shows data entering through an export from some other enterprise information system, while case 3 shows data being entered by a person through a human task. Cases 2 and 4 show data originating within the business process itself. Case 2 illustrates an assign task while case 4 illustrates setting the value within a snippet.

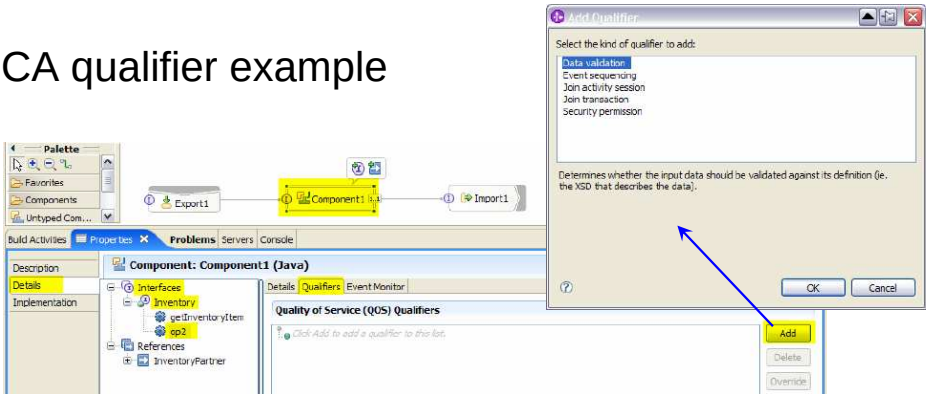
Note that these are just a few examples. Business object instance data which is not valid might occur with any SCA component type and applies equally to WebSphere Enterprise Service Bus as to WebSphere Process Server.

Business object instance validation

- API example

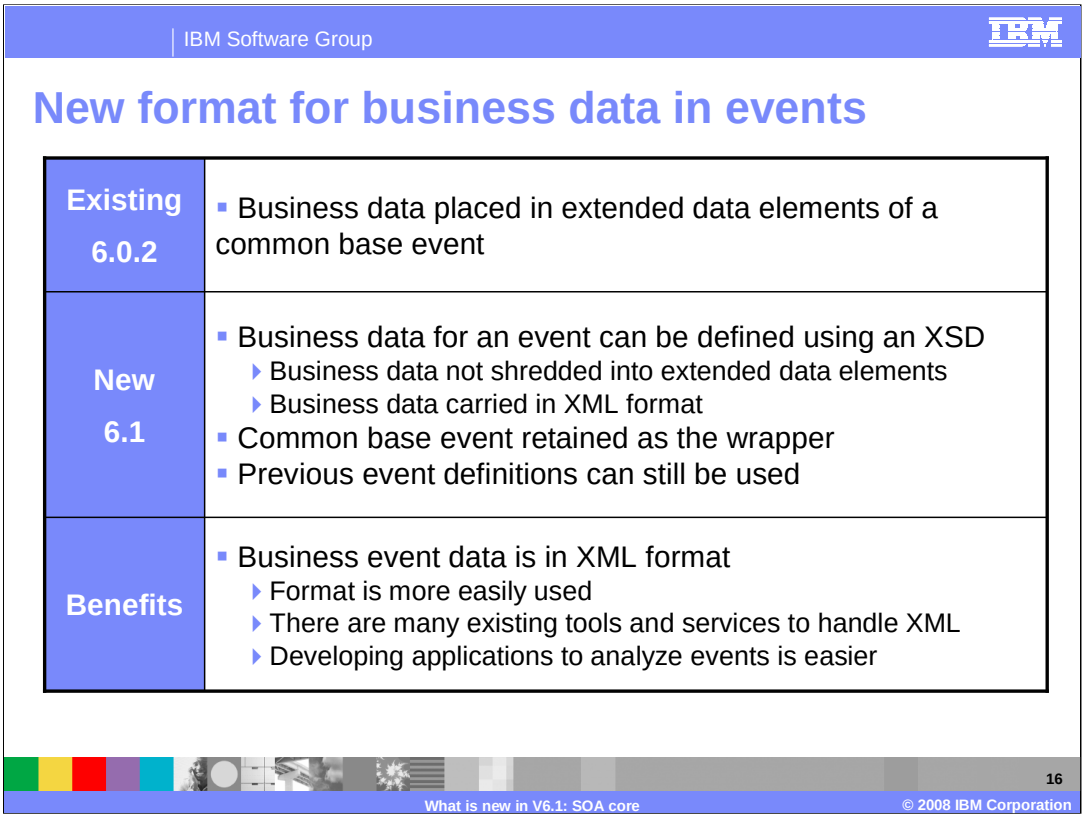
```
// Validate an InventoryItem BO instance
List diagnostics = new ArrayList();
BOInstanceValidator boValidator = (BOInstanceValidator)
    new ServiceManager().locateService("com/ibm/websphere/bo/BOInstanceValidator");
boolean result = boValidator.validate(inventoryItem, diagnostics);
```

- SCA qualifier example



The top portion of the slide shows a code sample of the API usage for the business object instance validation service.

The lower portion of the slide shows the SCA qualifier, called data validation, that can be set to cause business object instances to be validated upon entry to the SCA component. When data validation is performed, additional settings indicate if an error should be raised or only a warning issued when a business object instance is found to contain data which is not valid.

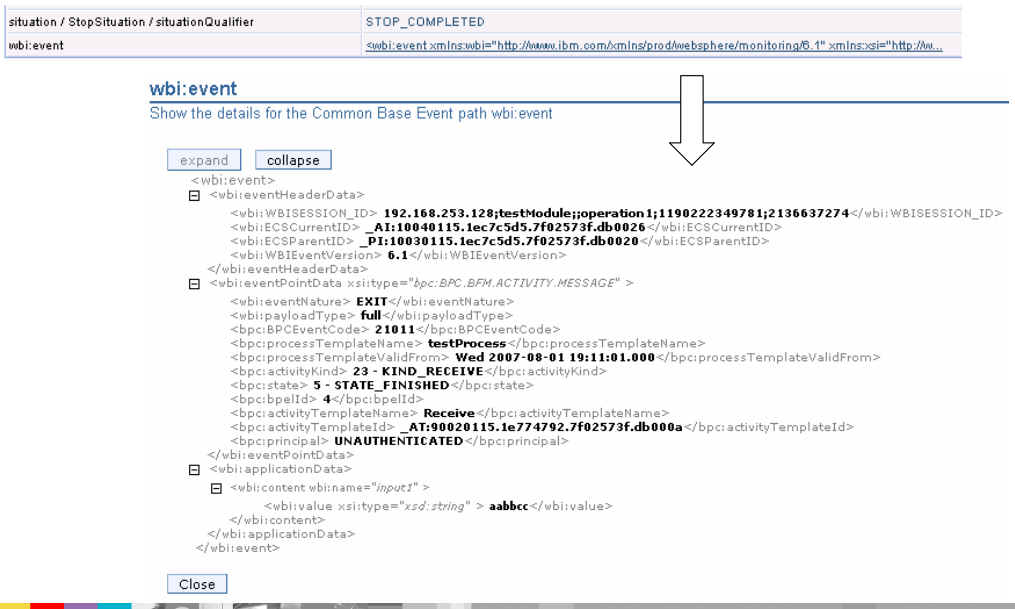


The common event infrastructure uses events defined by the common base event specification. The mechanism for carrying application data in a common base event is to place the data into extended data elements. When application data is specified as XML, the data is extracted element by element into these extended data elements.

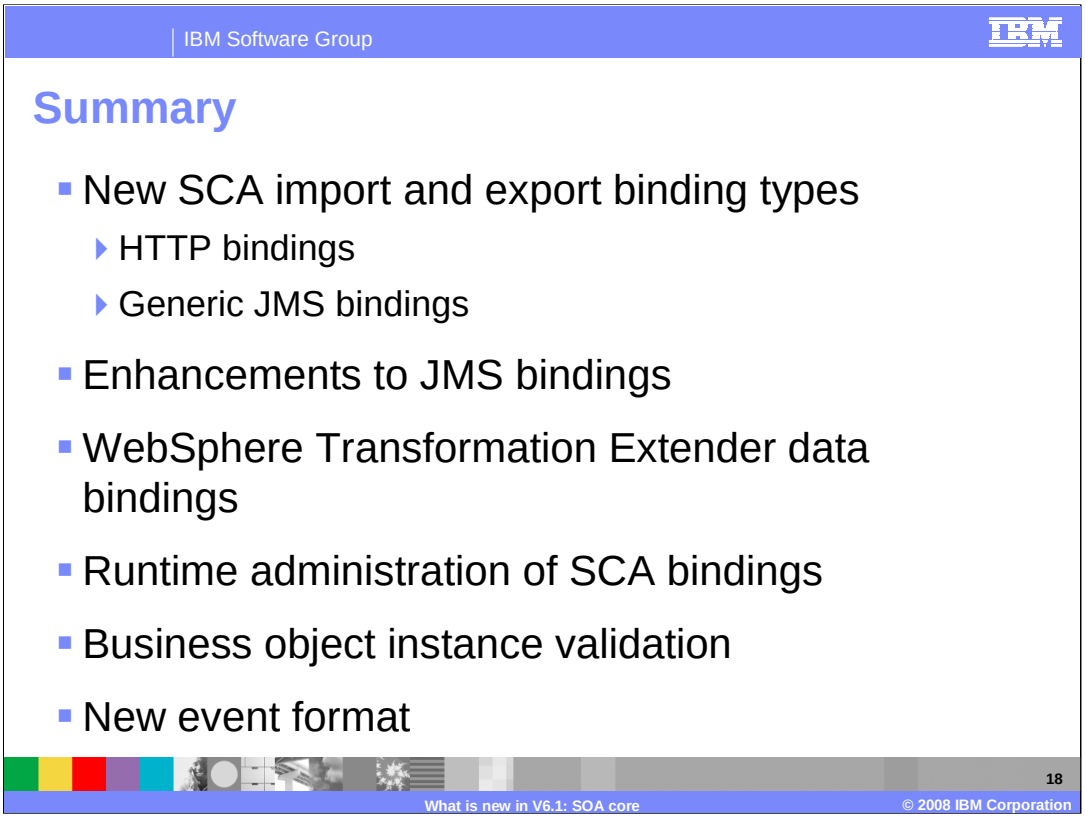
In 6.1 this mechanism has been extended to allow an event to have its application data defined using an XSD. The event directly contains the XML representation of the application data defined by the XSD rather than having the XML broken down into extended data elements. The overall common base event format is retained and it acts as a wrapper of the application data. In addition, which format is used for the common base event is configurable for each location where an event is raised.

Carrying the application data for the event as XSD defined XML provides a format that is easier to work with. Because there are many existing APIs and tools that deal with XML defined by XSDs it is much easier to develop applications that work with and analyze the events.

New format for business data in events



This is an example of the new style event being displayed from within the common base event browser. The wbi:event portion of the common base event contains the XML for the application data which is then displayed within the browser.



In this presentation you saw an overview of the enhancements made in version 6.1 for the SOA core layer of the WebSphere Process Server and WebSphere Enterprise Service Bus.

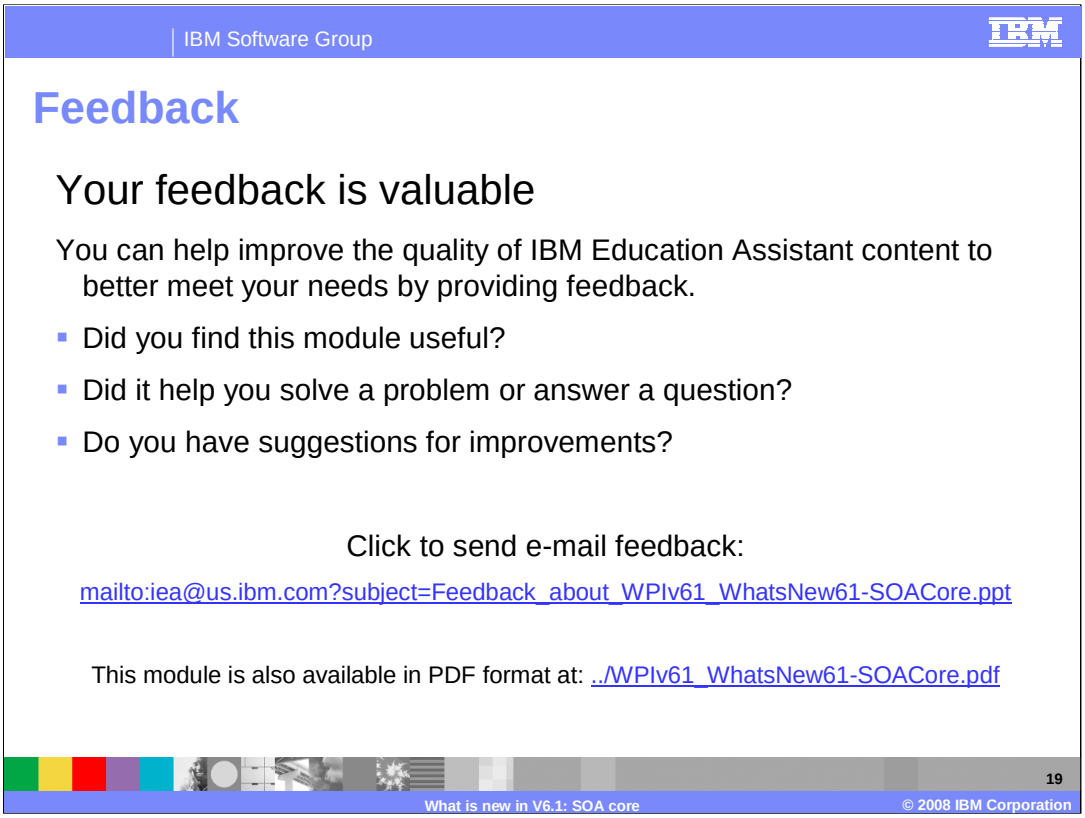
Most of the enhancements relate to the service component architecture portion of the SOA core, including two new SCA binding types for HTTP and generic JMS.

Additionally, enhancements to existing JMS bindings were described along with data bindings for WebSphere Transformation Extender integration.

From a strictly runtime perspective, the ability to administer several of the SCA binding types was described.

For business objects, the business object instance validation service has been added.

Finally, the common event infrastructure introduced a new event format which allows events to contain business data in XML.



You can help improve the quality of IBM Education Assistant content by providing feedback.

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

Java, 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.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.