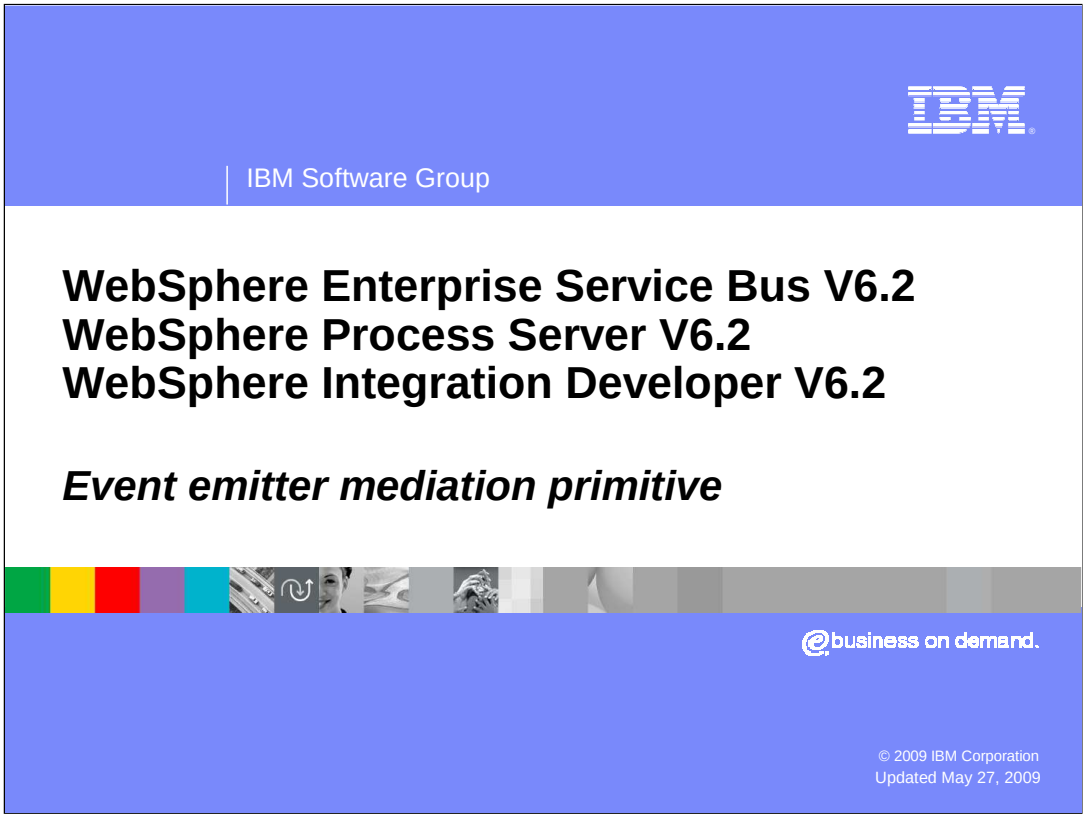


This presentation provides a detailed look at the event emitter primitive.

Goals

- Understand the event emitter primitive

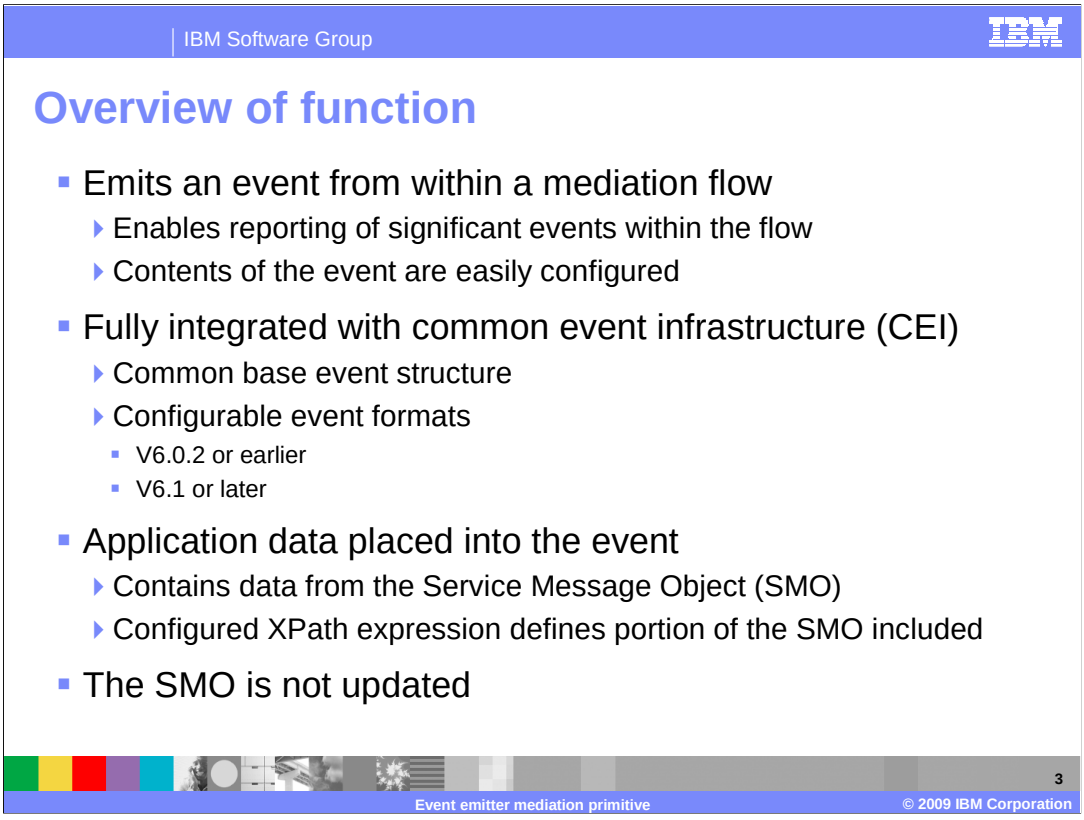
Event emitter

- Overview of function
- Use of terminals
- Definition of properties
- Event formats and contents of an event
- Error handling
- Best practices for usage
- Example usage

The goal of this presentation is to provide you with a full understanding of the event emitter primitive.

The presentation assumes that you are already familiar with the material presented in the presentations that cover common elements of all mediation primitives, such as properties, terminals, wiring and the use of promoted properties. The general knowledge of mediation primitives they provide is needed to understand the event emitter primitive specific material in this presentation.

In this presentation, an overview of the event emitter primitive is provided, along with information about the primitive's use of terminals and its properties. There is an explanation of the event formats and examples provided illustrating the contents of events. The error handling characteristics are then covered along with some information regarding best practices for the use of event emitters. Finally, an example mediation flow using an event emitter is provided.



The event emitter primitive provides a simple and easily configured mechanism that can be used to report significant events which have occurred within a mediation flow.

The event that is emitted is fully compatible with the common event infrastructure, which is part of the service oriented architecture core within WebSphere® Enterprise Service Bus and WebSphere Process Server. The overall structure of the event is known as a common base event. Events emitted by this primitive contain all the common base event elements that the common event infrastructure expects it to contain. The format for the application data included in the events changed between versions 6.0.2 and 6.1. A configuration option allows you to select if you want the events emitted in the version 6.0.2 format or the version 6.1 format.

When specifying the properties for the event emitter, you configure the portion of the service message object (SMO) that you want the event to contain. Examples of the resulting events, reflecting different portions of the SMO, are shown later in the presentation.

The event emitter primitive does not update the SMO.

Overview of function (continue)

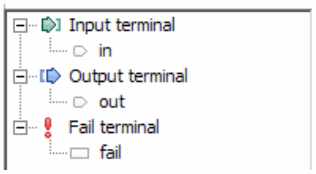
- Generated events are sent to the CEI server
  - Can be written to the event database
  - Can be forwarded using JMS topics or queues
  - Filters can be used to select specific events
- Can be used by monitoring applications
  - Common base event browser
  - WebSphere Business Monitor
  - User written event processing application

As stated on the previous slide, the event emitter primitive produces events which are fully compatible with the common event infrastructure. This implies some capabilities which are worth mentioning here. The events are sent to the CEI server, and therefore what happens with an event depends upon how the CEI server is configured. The possibilities are that the event is written to an event database, sent as a message on JMS queues, published to JMS topics or some combination of these. These capabilities make the event available to applications that query the database or receive the event through JMS. Through the use of configurable filters, the CEI server can selectively decide which events to forward on specific JMS queues or topics.

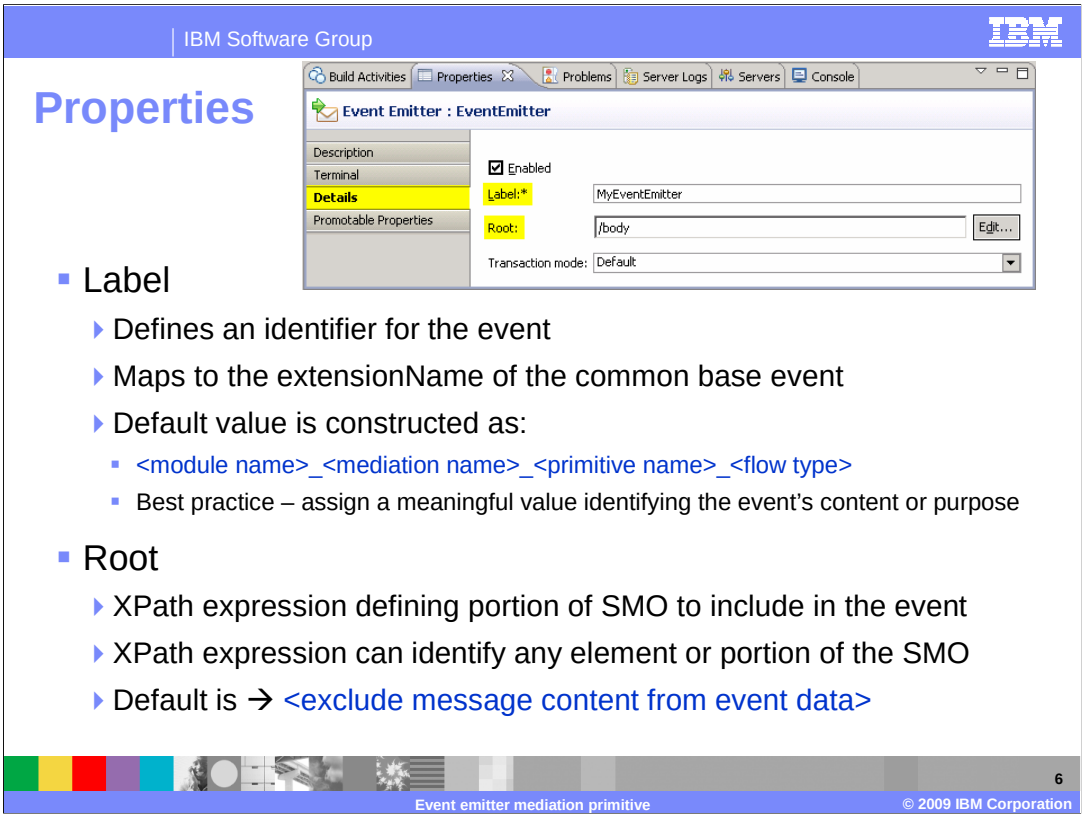
Based on your application requirements, the contents of events might need to be displayed or otherwise interpreted by monitoring applications. The common base event browser is provided as part of the WebSphere Enterprise Service Bus and WebSphere Process Server. It provides a mechanism to filter, sort and display events. The WebSphere Business Monitor has capabilities to provide analysis of events. For some requirements, you might need to write your own application to analyze and act upon events.

Terminals

- Terminals:
  - Input terminal
  - One output terminal
  - Fail terminal
- All terminals must be for the same message type



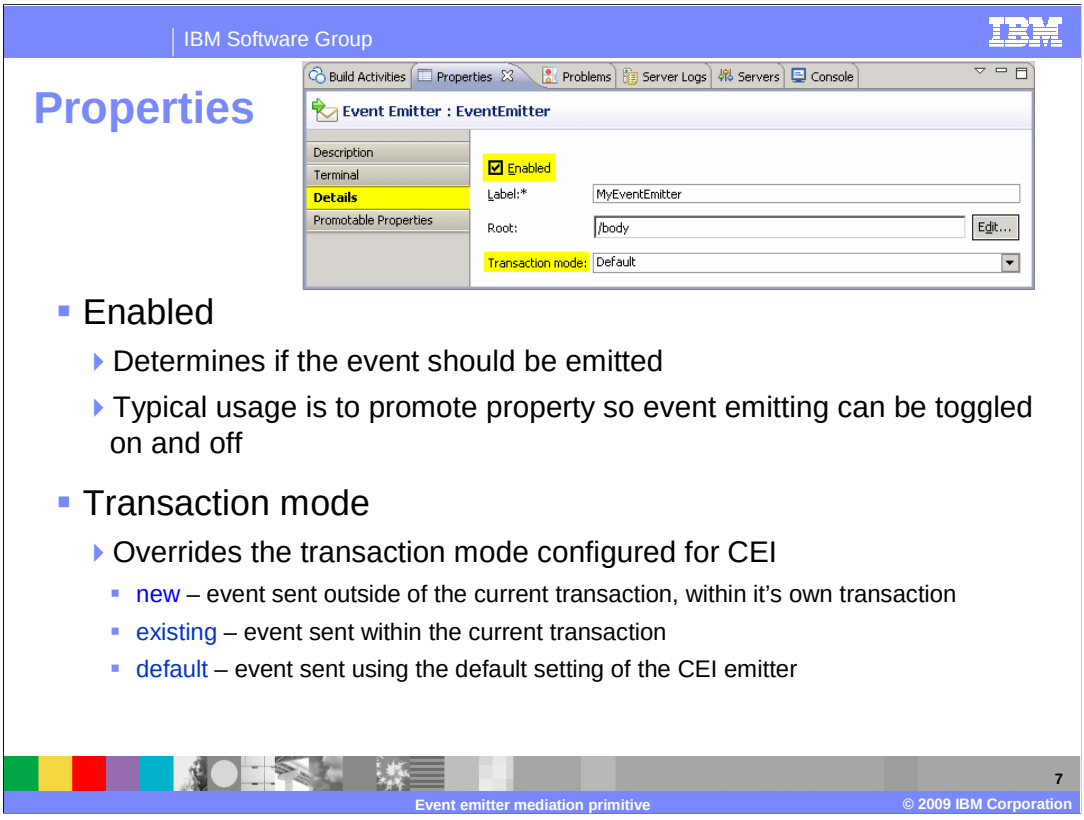
The event emitter primitive has one input terminal, one output terminal and a fail terminal. The output terminal must be for the same message type as the input terminal, because the event emitter primitive does not modify the message body. Shown here is an event emitter primitive with its terminals and the terminals as seen in the properties view.



In the upper right portion of the slide is a screen capture showing the Details tab of the Properties view for an event emitter primitive. The properties that affect the contents of the event are the label and root properties, which are highlighted in the screen capture.

The Label property provides an identifier for the event. The value for this property is placed into the extensionName field of the common base event. If you do not assign a specific value to this property, WebSphere Integration Developer provides a default value. It is constructed from the mediation module name, the mediation flow name, the event emitter primitive name and the flow type, all separated using underbars. It is likely that this default won't be meaningful to whomever is consuming the events, so as a best practice, you should assign some meaningful value to this property.

The Root property contains an XPath expression and is used to define the portion of the service message object that is included in the event. It can define a leaf element, a data object within the SMO or the entire SMO. The Edit… button can be used to access the XPath expression builder dialog that enables you to easily construct the XPath expression. If you do not specify this property, the default is to exclude all SMO content from the event.

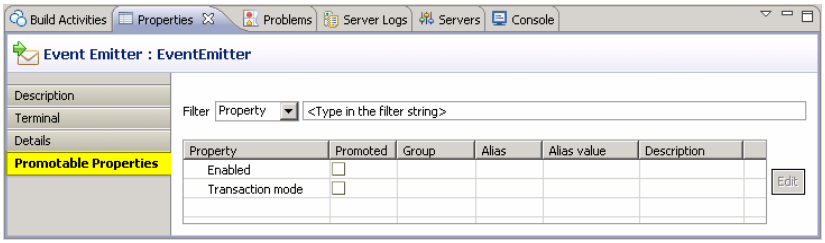


The properties that do not affect the actual contents of the event are the enabled and transaction mode properties, which are highlighted in the screen capture.

The Enabled property is a toggle which tells the primitive whether it should actually emit the event. The typical usage of this property is to promote it, thus enabling the emitting of events to be turned on and off administratively. This allows mediation flows to contain event emitter primitives that are disabled most of the time, but can be administratively enabled when needed for problem determination or other reasons.

The Transaction mode property is used to define transaction handling for the writing of the event. A setting of new means that the event is emitted within the context of its own transaction. A setting of existing means that the event participates in the current transaction. Finally, a setting of default indicates to use whatever setting has been configured for CEI.

Promotable properties



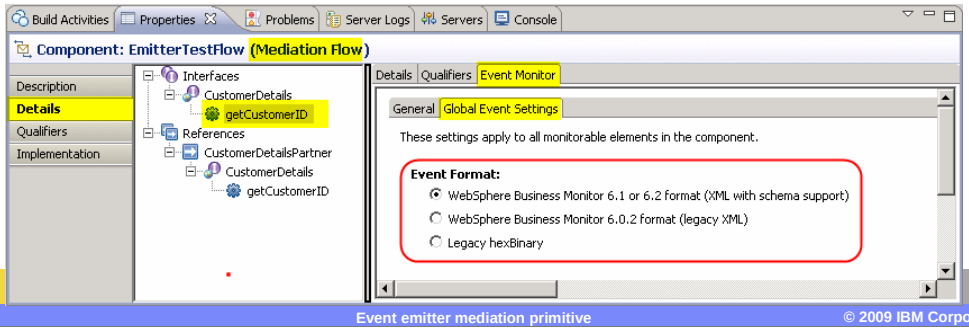
- Promotable
  - Enabled
  - Transaction mode
- Not promotable
  - Label
  - Root

This slide shows the Promotable Properties panel for the event emitter primitive. As was mentioned on the previous slide, the enabled property is promotable, allowing administrative control over whether the event is actually emitted. The transaction mode property is also promotable.

The label and root properties, which influence the contents of the event, are not promotable. A change to either of them might cause the emitted event to be incompatible with the expectations of an external event monitoring application.

Event formats

- V6.1 and later event format – the default configuration
  - Event data is based on XML schema definition (XSD) wbi:event
  - XML for the event wrapped in a common base event
- V6.0.2 and earlier event format – optional configuration
  - Application data is placed into extendedDataElement of the common base event
  - Event definitions can be exported for use by monitoring applications
- Format is specified in the mediation flow component
  - Specified at operation level, but applies to all event emitters for the mediation flow component
- This presentation focuses on the V6.1 and later event formats



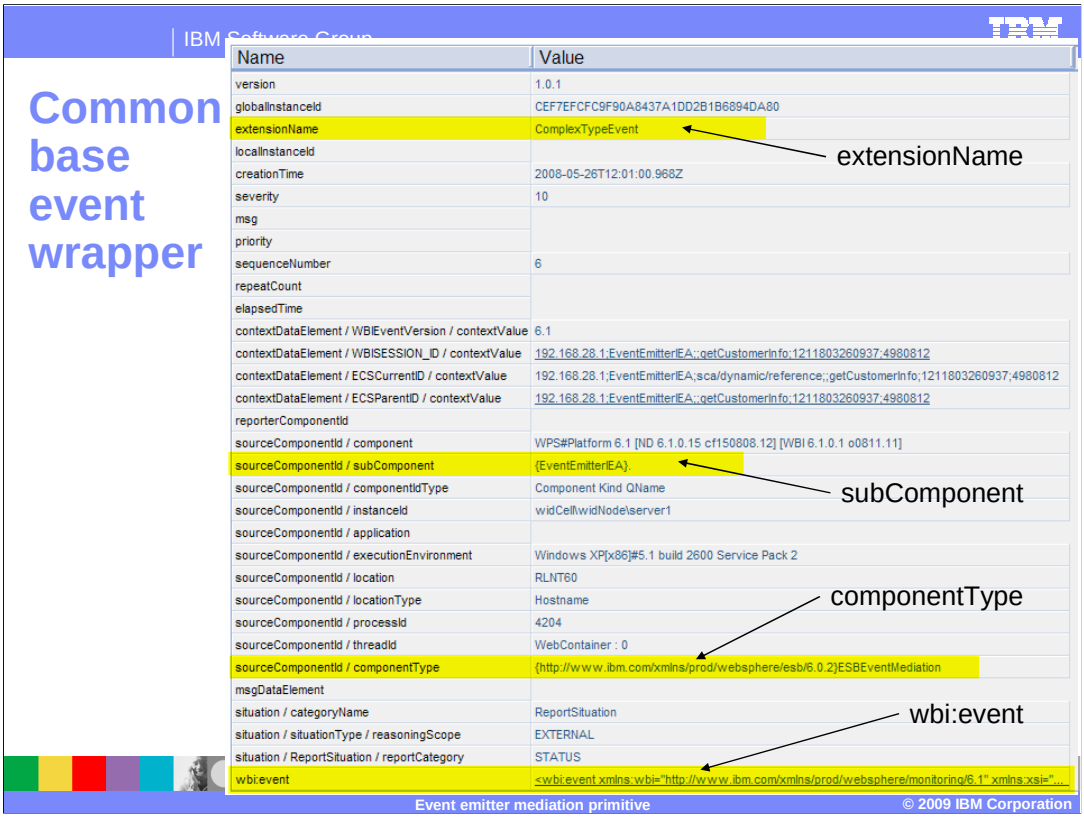
You are able to configure what event format you prefer for the emitted events. By default, the version 6.1 format is used, but you can change the configuration to use the version 6.0.2 format.

The version 6.1 format is based on the wbi:event XML schema definition (XSD). The XML representation of the event is placed into a common base event wrapper, so that it is compatible with the common event infrastructure. This results in a small amount of information being duplicated in the common base event and the XML, but the application data itself only appears in the XML.

In the version 6.0.2 format, the application data follows the common base event format, using the extendedDataElement to contain the application data. Because this data is not self describing, event definitions using the version 6.0.2 format can be exported for use by monitoring applications.

The configuration option for choosing between the version 6.0.2 and version 6.1 formats is specified at the individual operations level of the mediation flow component, as is shown in the screen capture. However, changing the setting for one operation changes it for all the operations of the interface. Because of this, the result is that all event emitters within the all the flows for the mediation flow component use the same event format.

The remainder of this presentation only looks at the version 6.1 event formats. For additional information about the version 6.0.2 event formats, see the event emitter primitive presentation for version 6.0.2.



The next several slides take a look at the contents of an event.

The screen capture of the event in this slide was taken from an event displayed in the common base event browser. The common base event browser is an application built into the WebSphere Enterprise Service Bus and WebSphere Process Server.
This is the common base event wrapper used to wrap the wbi:event XML containing the complete event data. This wrapper is used so that the wbi:event can flow though the common event infrastructure which is based on the handling of common base events.

Those fields in the common base event that have some particular relevance or relationship to an event emitter primitive are highlighted in the screen capture. The first thing to notice is the extensionName field of the event. Its value is set to the value of the label property of the event emitter primitive. The next is the subComponent field which contains the name of the mediation module. The componentType field identifies this as an event from an ESB mediation. Finally, the wbi:event field contains the entire XML document containing the complete version 6.1 event data.

wbi:event

```
<wbi:event>
⊞ <wbi:eventHeaderData>
⊞ <wbi:eventPointData xsi:type="esb:WESB.ESBEventMediation.CUSTOM" >
⊞ <wbi:applicationData>
</wbi:event>
```

- wbi:eventHeaderData
  - Session and correlation IDs
- wbi:eventPointData
  - Flow location and application data root
- wbi:applicationData
  - Contents of SMO identified by root

This slide shows the top level of the wbi:event. It is composed of the wbi:eventHeaderData containing session and correlation IDs. Also, the wbi:eventPointData contains information about the flow from which the event was emitted. Finally, the wbi:applicationData contains the portion of the SMO identified by the root property of the event emitter. Each of these is looked at in more detail in the subsequent slides.

wbi:eventHeaderData

```
<wbi:eventHeaderData>
    <wbi:WBISESSION_ID> 192.168.28.1;EventEmitterIEA;;getCustomerInfo;1211803260937;4980812</wbi:WBISESSION_ID>
    <wbi:ECSCurrentID> 192.168.28.1;EventEmitterIEA;sca/dynamic/reference;;getCustomerInfo;1211803260937;4980812</wbi:ECSCurrentID>
    <wbi:ECSParentID> 192.168.28.1;EventEmitterIEA;;getCustomerInfo;1211803260937;4980812</wbi:ECSParentID>
    <wbi:WBIEventVersion> 6.1</wbi:WBIEventVersion>
</wbi:eventHeaderData>
```

- Session ID
- Correlation IDs
- Event version

The screen capture on this slide shows the contents of the wbi:eventHeaderData portion of the event, which includes session and correlation IDs. Using these IDs, this event can be associated with other events that occurred as part of the same overall flow. These other events can be predefined events such as those that are emitted by SCA and BPEL, or they can also be application defined events. Also included in the wbi:eventHeaderData is the event version, identifying the format for the event. This can be useful if the schema for wbi:event is updated in a subsequent version.

wbi:eventPointData

```
<wbi:eventPointData xsi:type="esb:WESB.ESBEventMediation.CUSTOM" >
    <wbi:eventNature> CUSTOM</wbi:eventNature>
    <wbi:payloadType> full</wbi:payloadType>
    <esb:ModuleName> EventEmitterIEA</esb:ModuleName>
    <esb:MediationName> ComplexType</esb:MediationName>
    <esb:EventEmitterLabel> ComplexTypeEvent</esb:EventEmitterLabel>
    <esb:Root> /body/getCustomerInfoResponse/customerInfo</esb:Root>
</wbi:eventPointData>
```

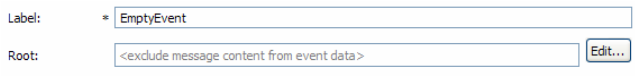
- eventNature and payloadType
  - Used to identify aspects of BPEL and SCA events
  - eventNature=Custom and payloadType=full for events from event emitter
- ModuleName
  - Name of the mediation module containing the flow
- MediationName
  - Name of the event emitter primitive producing this event
- EventEmitterLabel
  - Value from the label property of the event emitter
- Root
  - XPath expression defining portion of the SMO included in the application data

The screen capture on this slide shows the contents of the wbi:eventPointData portion of the event. The first two fields, the eventNature and payloadType, contain information that is relevant for events that are configured with a BPEL business process or SCA flow. For event emitter events from a mediation flow, the eventNature is set to CUSTOM and the payloadType is set to full.

The remaining fields are specific to event emitter generated events. The ModuleName is the name of the mediation module in which the event emitter exists. The MediationName is the name of the event emitter primitive which generated the event. The EventEmitterLabel contains the value of the label property configured for the event emitter primitive. The Root contains the XPath expression which defines what portion of the SMO is included in the event

wbi:applicationData – exclude message data

- Root property:
  - <exclude message content from event data>



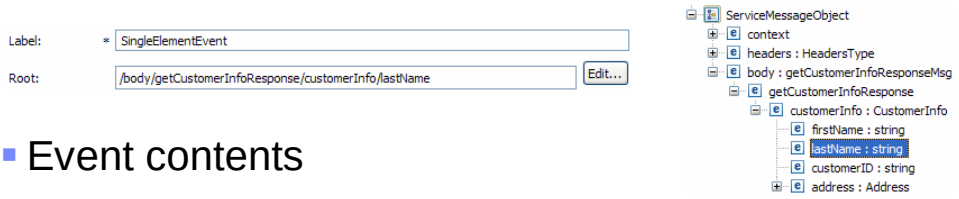
- Event contents
  - wbi:eventPointData – does not include Root
  - wbi:applicationData – not included

```
<wbi:event>
 <wbi:eventHeaderData>
 <wbi:eventPointData xsi:type="esb:WESB.ESBEventMediation.CUSTOM" >
     <wbi:eventNature> CUSTOM</wbi:eventNature>
     <wbi:payloadType> full</wbi:payloadType>
     <esb:ModuleName> EventEmitter1EA</esb:ModuleName>
     <esb:MediationName> Empty</esb:MediationName>
     <esb:EventEmitterLabel> EmptyEvent</esb:EventEmitterLabel>
  </wbi:eventPointData>
</wbi:event>
```


The next several slides look at the wbi:applicationData portion of the event, each examining the event content generated for a different value of the root property. The top screen capture shows a portion of the properties panel for the event emitter. The root property is set to the default value, “exclude message content from event data.” Looking at the lower screen capture you can see the wbi:eventPointData, which was described on the previous slide. Notice that the root field is not present. Also notice that the entire wbi:application section is not included in the event.

wbi:applicationData – leaf element

- Root property identifies a leaf element

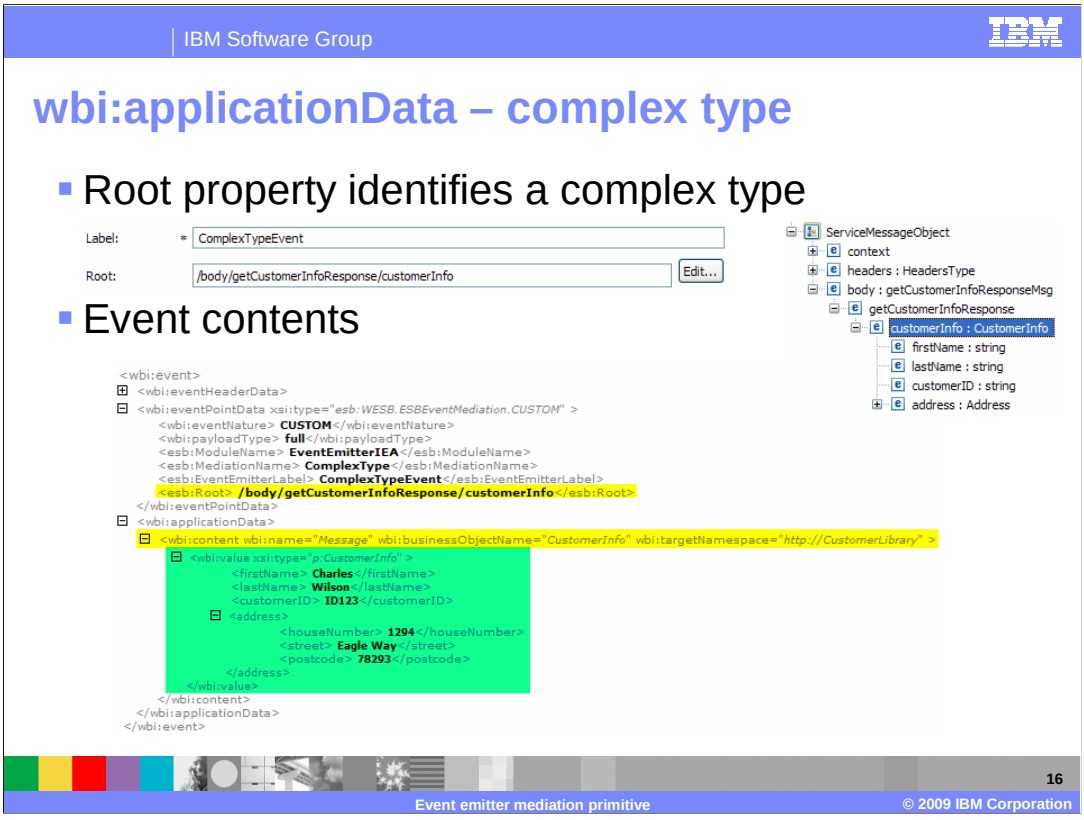


- Event contents

```
<wbi:event>
  <wbi:eventHeaderData>
  <wbi:eventPointData xsi:type="esb:WESB.ESBEventMediation.CUSTOM" >
      <wbi:eventNature> CUSTOM</wbi:eventNature>
      <wbi:payloadType> full</wbi:payloadType>
      <esb:ModuleName> EventEmitterIEA</esb:ModuleName>
      <esb:MediationName> SingleElement</esb:MediationName>
      <esb:EventEmitterLabel> SingleElementEvent</esb:EventEmitterLabel>
      <esb:Root> /body/getCustomerInfoResponse/customerInfo/lastName</esb:Root>
  </wbi:eventPointData>
  <wbi:applicationData>
    <wbi:content wbi:name="Message" >
        <wbi:value xsi:type="xsd:string" > Wilson</wbi:value>
    </wbi:content>
  </wbi:applicationData>
</wbi:event>
```

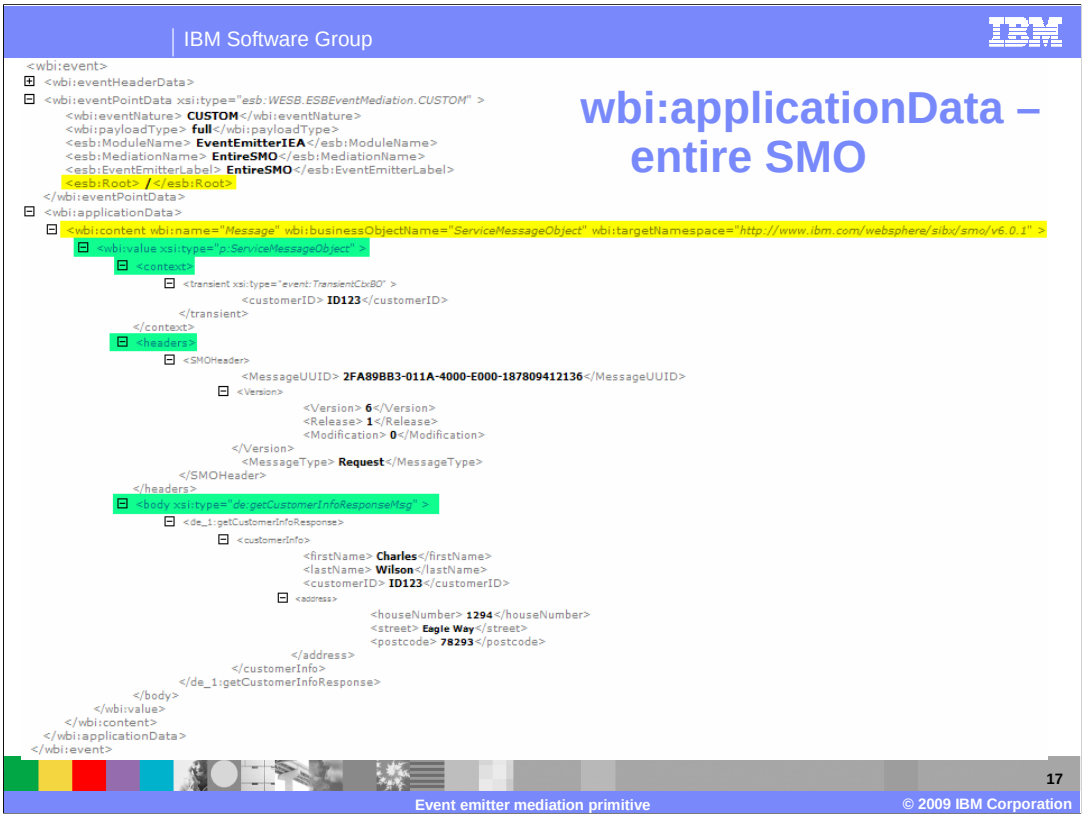

In this case, the wbi:applicationData reflects a root property that identifies a leaf element in the SMO. The screen capture of the properties from the event emitter shows the XPath expression identifying the lastName field in the body of the SMO. This is also reflected in the expanded SMO body shown with a screen capture from the XPath expression builder data viewer panel.

Looking at the contents of the event, you can see that the wbi:eventHeaderData contains an esb:Root containing the XPath expression identifying the lastName field in the SMO body. In the wbi:applicationData section, you can see that the wbi:content tag contains the wbi:name value of Message with no further qualifications. The wbi:value tag identifies the value to be “Wilson” of type string. There is no indication in the wbi:applicationData section that this value is from the lastName field of the SMO body. That information is only obtained from the root specification in the wbi:eventHeaderData.



This slide shows the wbi:applicationData reflecting a root property that identifies a complex type within the SMO. The screen capture of the properties from the event emitter shows the XPath expression identifying the customerInfo field in the body of the SMO, which is of type CustomerInfo. This is also reflected in the expanded SMO body shown with a screen capture from the XPath expression builder.

In the event, you can see that the wbi:eventHeaderData contains and esb:Root with an XPath expression reflecting the root property from the event emitter. In the wbi:applicationData section, you can see that the wbi:content tag again contains the wbi:name value of Message. However, this time there is additional information identifying the type of business object to be CustomerInfo and the target namespace is also specified. The wbi:value tag also identifies the type to be CustomerInfo, containing an exploded view of the complex type showing all the values for its attributes, including the nested address business object. Similar to the previous case, the wbi:applicationData section does not identify that this data comes from the customerInfo element of the SMO. That information is only obtained from the root specification in the wbi:eventHeaderData.



The entire SMO is shown in the wbi:applicationData in this slide, reflecting a root property of forward slash (“/”), which you can see in the esb:Root of the wbi:eventHeaderData.

In the wbi:applicationData section, you can see that the wbi:content tag again contains the wbi:name value of Message and this time identifies the type of business object to be a ServiceMessageObject along with its target namespace. The wbi:value tag also identifies the type to be ServiceMessageObject. In the exploded view, you can see that the context, headers and body sections of the SMO are all included. Fields in the SMO with null values are not included. For example, in the context section only the transient context appears because that is the only portion of the context containing information. Notice that in the body section the type reflects the SMO message type, getCustomerInfoResponseMsg.

Error processing

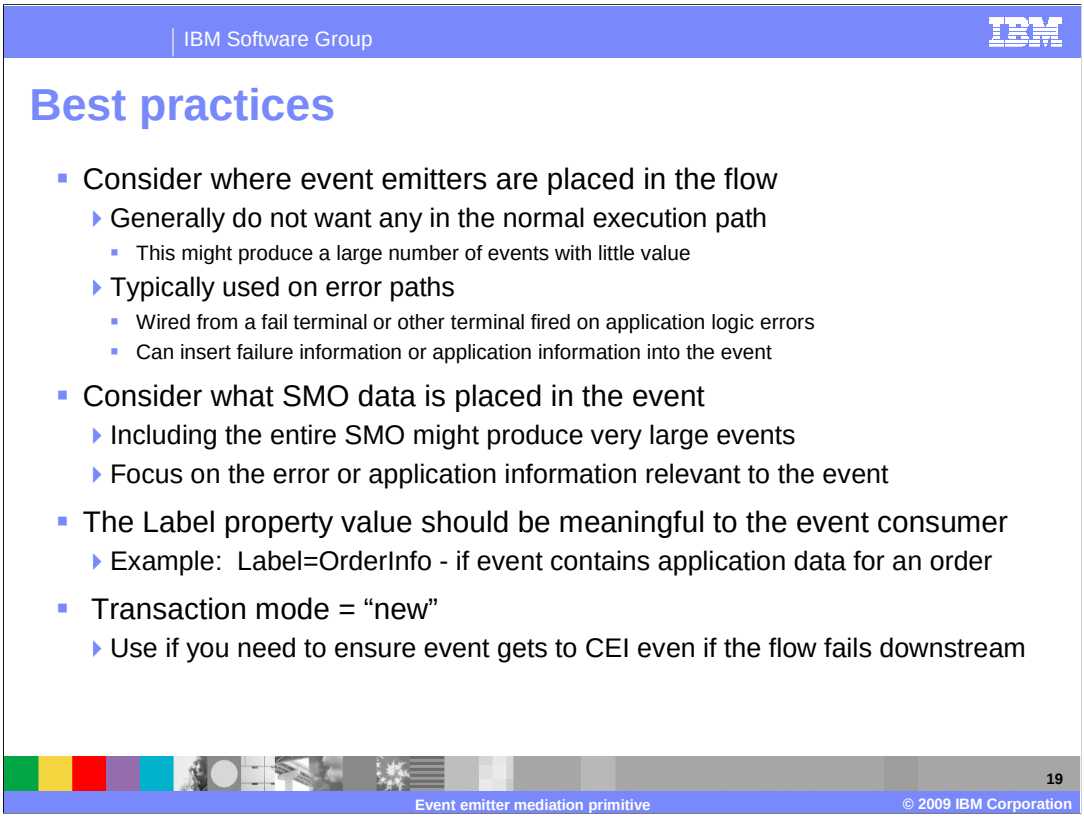
- MediationRuntimeException thrown for:
  - Root property XPath expression syntax is not valid
  - Label property value is:
    - null
    - longer than 64 characters
  - Transaction Mode property value is not valid
    - WebSphere Integration Developer ensures values set is valid
    - A value which is not valid might be set administratively if the property is promoted
- MediationBusinessException (Fail terminal flow)
  - Any transient error, such as CEI being unavailable
- Root XPath value not found in Service Message Object
  - Not considered an error condition
  - The event is produced without a Message extendedDataElement

The error processing details and considerations are examined in this slide.

A MediationRuntimeException is thrown for several reasons. One case is where the root property contains an XPath expression with a syntax that is not valid. Another cause is when the label property is null or if it is longer than 64 characters. The MediationRuntimeException also occurs when there is a transaction mode property value that is not valid. This only occurs in the situation where the transaction mode property is promoted, and the value that is set administratively is not valid. The WebSphere Integration Developer prevents you from setting a value that is not valid at development time.

A MediationBusinessException occurs for any transient problems, such as CEI not being available. These exceptions cause the fail terminal flow to be taken.

It is possible for the root property to have a syntactically correct XPath expression that does not identify an element within the SMO. This can occur do to a mistake in specifying the property or from an optional element not being present in the SMO. This is considered a normal situation by the event emitter primitive, and it produces an event without the application data being present.



When designing your flow, there are some best practices that you should consider regarding the use of event emitter primitives.

The first of these is to be judicious about where in the flow you place them. Generally speaking, unless you have a specific application requirement, you do not want to place an event emitter on the normal path of a flow. This has the potential to raise a large number of events with potentially little value. A typical use is to put an event emitter on an error path, such as wired to a fail terminal or some other output terminal that represents an application error. Such an event might contain the failure information from the SMO header or some application specific data from the body of the SMO.

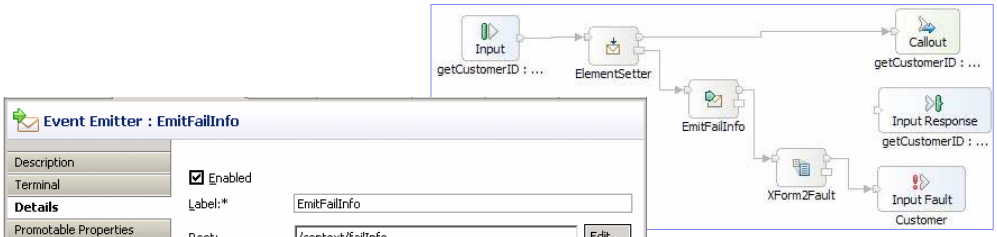
Placing the entire SMO into an event can be appropriate in some instances. However, a smaller event is generated if you can be selective about the specific data you need to see for the situation you are reporting with the event.

Give the label property a value which has meaning to the consumer of the event. This might be something like OrderInfo for an event that contains application data for an order.

Finally, using a transaction mode setting of new ensures the event gets emitted to CEI even if there is a failure in your flow downstream from the event emitter.

Example

- Raise an event when failure in mediation flow
  - Wire event emitter to fail terminals of primitives in flow
  - Configure emitter to put failInfo into event
    - This identifies the failure
    - Entire SMO might be useful to help determine what caused the failure
  - If interface has a fault, wire emitter to XSLT and return fault
    - Otherwise wire to a Stop or Fail primitive



In this example, an event emitter is being used to report the occurrence of a failure in the mediation flow. The flow contains a message element setter primitive. For a normal flow with no failures, this message element setter is the only primitive through which the flow passes.

The event emitter is wired to the fail terminal of the message element setter. You can see in the properties, the root specifies that the failInfo from the context of the SMO is to be inserted into the event, thus providing detail of why the failure occurred. If the application data might be helpful in this case, the entire SMO can be placed into the event so that the failInfo and the SMO body are both available. This also provides the full context and header information, which in some cases can also be helpful.

Notice that in this particular flow, the event emitter is wired to an XSL transformation primitive, which transforms the input message into a fault message that can be returned using the input fault node. An alternative is to wire the event emitter to a stop or fail primitive.

Summary

- Examined the event emitter primitive

Event emitter

  - Overview of function
  - Use of terminals
  - Definition of properties
  - Event formats and contents of an event
  - Error handling
  - Best practices for usage
  - Example usage

In summary, this presentation presented an overview of the event emitter mediation primitive, along with information about the primitive's use of terminals and its properties. There was an explanation of the event formats along with examples of the contents of the events emitted. The error handling characteristics were then covered along with some information regarding best practices for the use of event emitters. Finally, an example usage of an event emitter was illustrated and explained.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WBPMv62_EventEmitterPrimitive.ppt

This module is also available in PDF format at: ../WBPMv62_EventEmitterPrimitive.pdf

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

Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

WebSphere

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Other company, product, or service names may be trademarks or service marks of others.

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.

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.