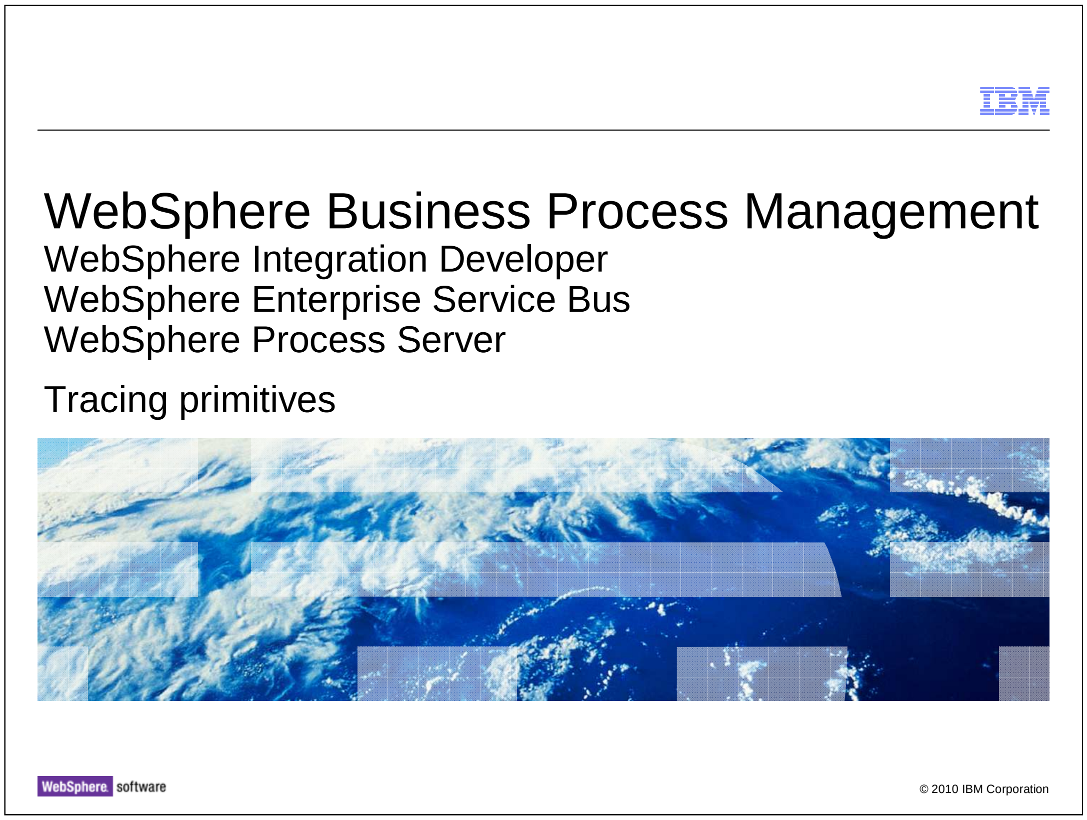
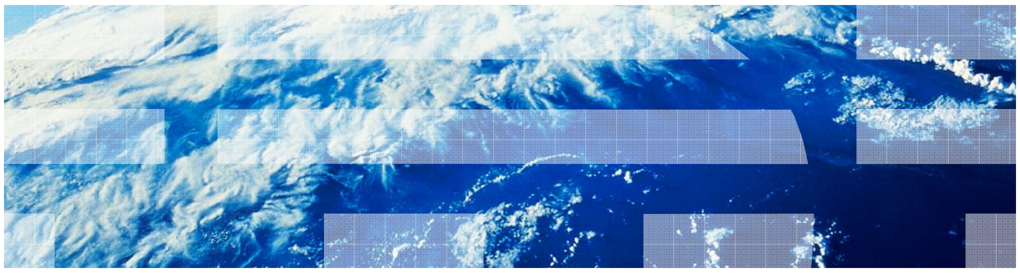


WebSphere Business Process Management

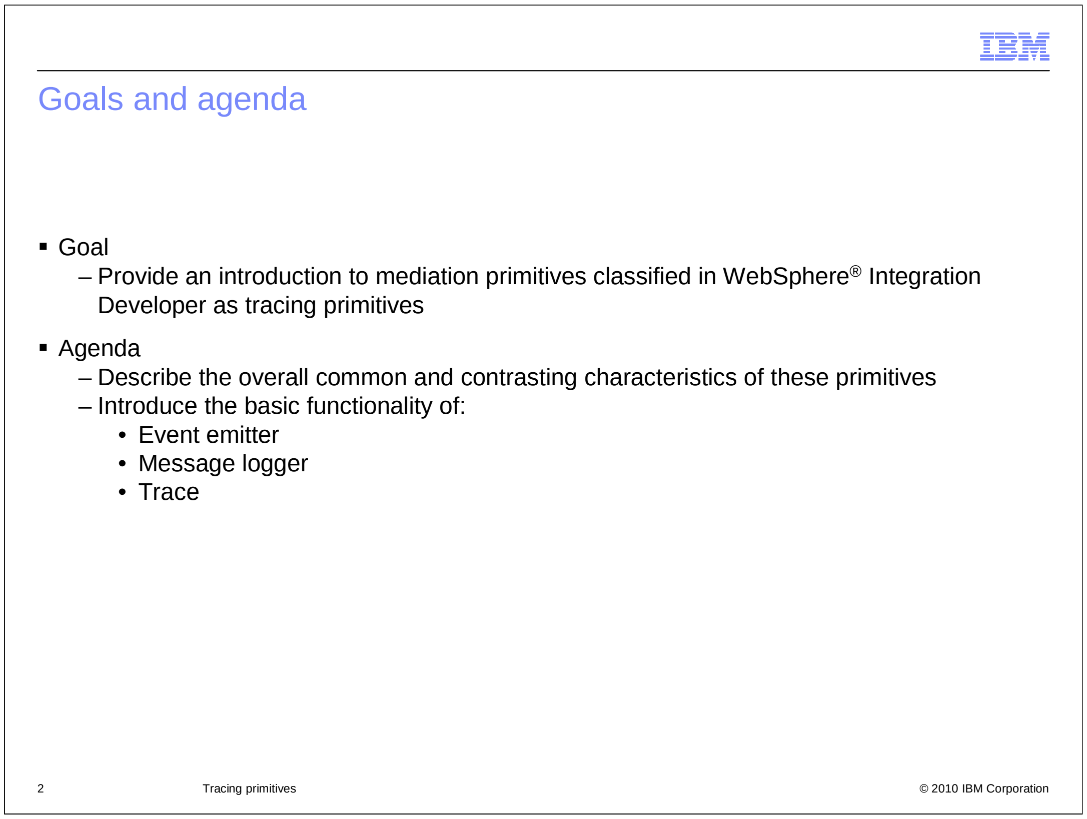
WebSphere Integration Developer
WebSphere Enterprise Service Bus
WebSphere Process Server

Tracing primitives



WebSphere software

This presentation introduces the tracing primitives used to capture message content and state information at selected points in a mediation flow.



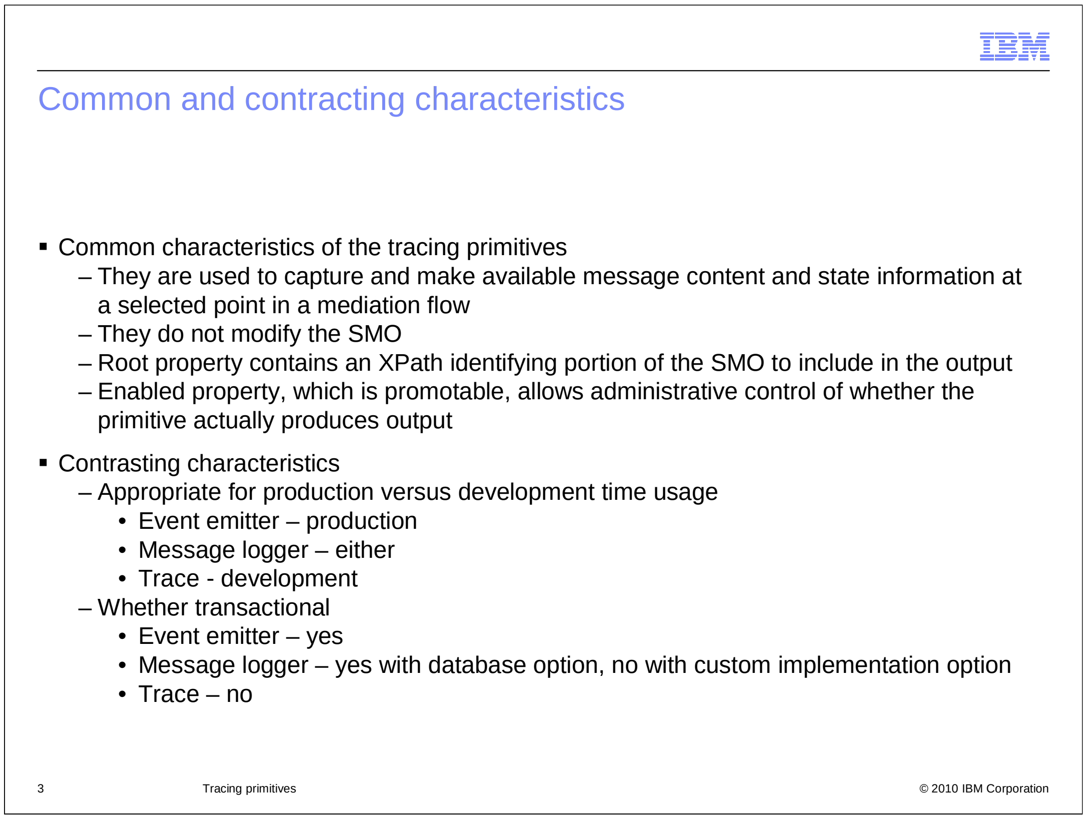
Goals and agenda

- Goal
  - – Provide an introduction to mediation primitives classified in WebSphere® Integration Developer as tracing primitives
- Agenda
  - – Describe the overall common and contrasting characteristics of these primitives
  - – Introduce the basic functionality of:
    - Event emitter
    - Message logger
    - Trace

2 Tracing primitives © 2010 IBM Corporation

The goal of this presentation is to introduce a set of primitives that share some common characteristics, specifically those that are classified within WebSphere Integration Developer as tracing primitives.

The presentation starts out by explaining the common and contrasting characteristics of these primitives, and then provides a description of the basic functionality of each. The primitives presented are the event emitter primitive, the message logger primitive and the trace primitive.



Common and contracting characteristics

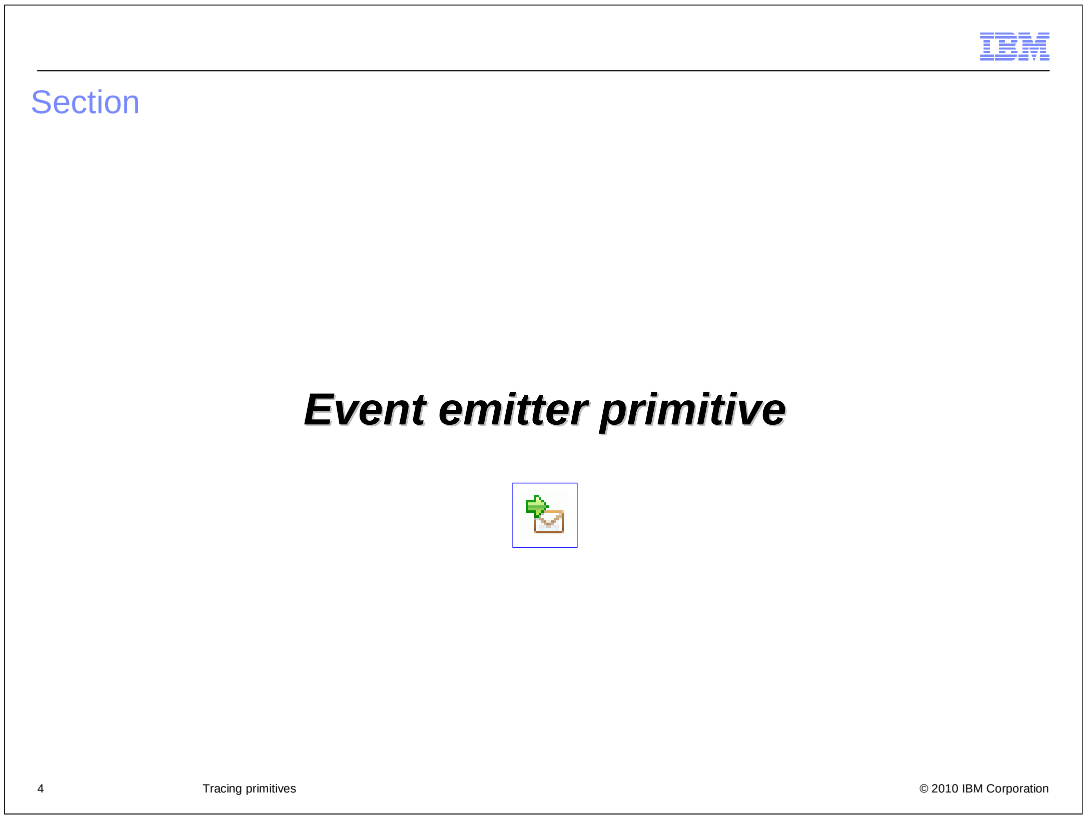
- Common characteristics of the tracing primitives
  - They are used to capture and make available message content and state information at a selected point in a mediation flow
  - They do not modify the SMO
  - Root property contains an XPath identifying portion of the SMO to include in the output
  - Enabled property, which is promotable, allows administrative control of whether the primitive actually produces output
- Contrasting characteristics
  - Appropriate for production versus development time usage
    - Event emitter – production
    - Message logger – either
    - Trace - development
  - Whether transactional
    - Event emitter – yes
    - Message logger – yes with database option, no with custom implementation option
    - Trace – no

3 Tracing primitives © 2010 IBM Corporation

The tracing primitives are all used to capture and make available information about message content and state within a mediation flow. They do not modify the contents of the SMO, so from the perspective of the flow logic, they have no effect on downstream primitives. What portion of the SMO to include in the output is configurable using an XPath expression in the root property. They all have the enabled property, which can be promoted, allowing an administrator the ability to toggle the primitive off and on. This can be useful when the primitive is used for debugging or other special case scenarios.

There are some contrasting characteristics of these primitives, one of which is appropriate usage of the primitive. The event emitter is most likely to be used for a production system to report error situations or other unusual and significant events. At the other end of the spectrum, the trace primitive is most likely used during development and debug of mediation flows. It provides you, the integration developer, a way to easily instrument your code to examine the SMO at various points in the flow. The message logger primitive falls more in the middle, and might be useful in a production scenario and can also be useful in a development scenario.

Another contrasting characteristic is transactionality. The event emitter and the database option of the message logger are both transactional. They can be configured to be committed within a local transaction or can participate in the transaction for the flow. This enables you to control whether a particular event or message is actually produced when the transaction for the flow does not complete successfully. The trace primitive, and the message logger with custom logging option, are not transactional, with the message being produced at the time the primitive runs in the flow.

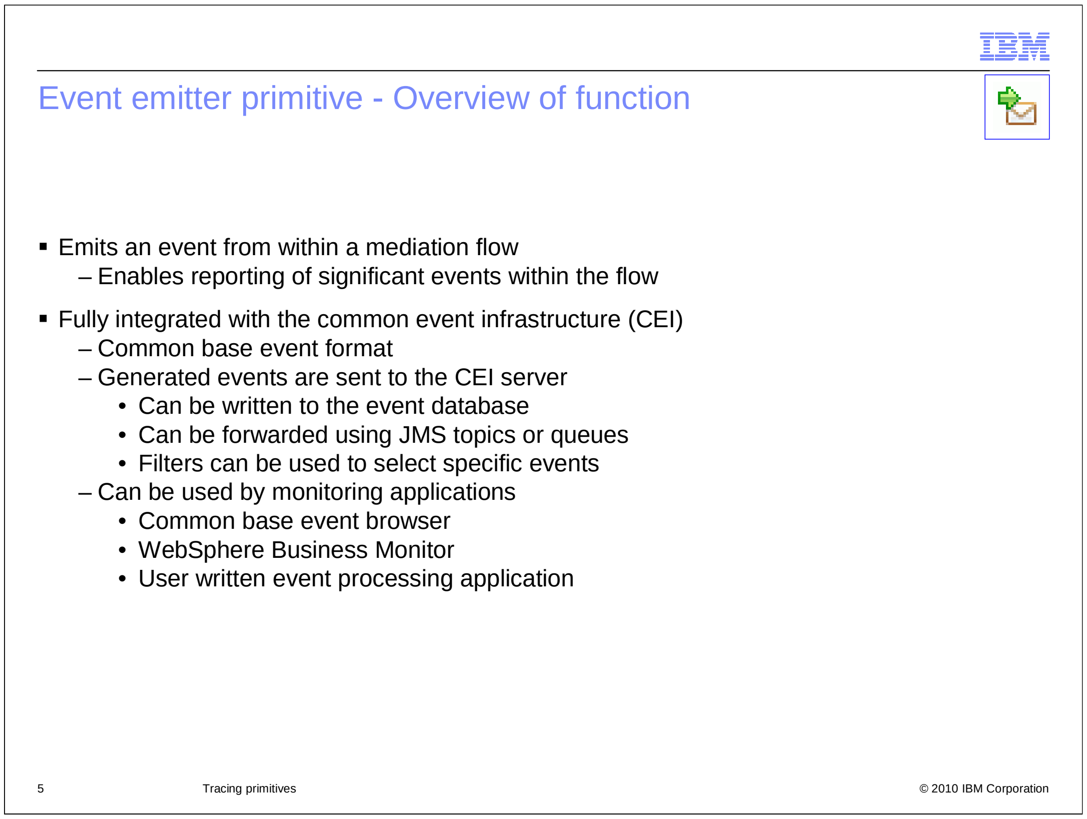


Section

Event emitter primitive

4 Tracing primitives © 2010 IBM Corporation

This section of the presentation provides an overview of the event emitter primitive.



Event emitter primitive - Overview of function

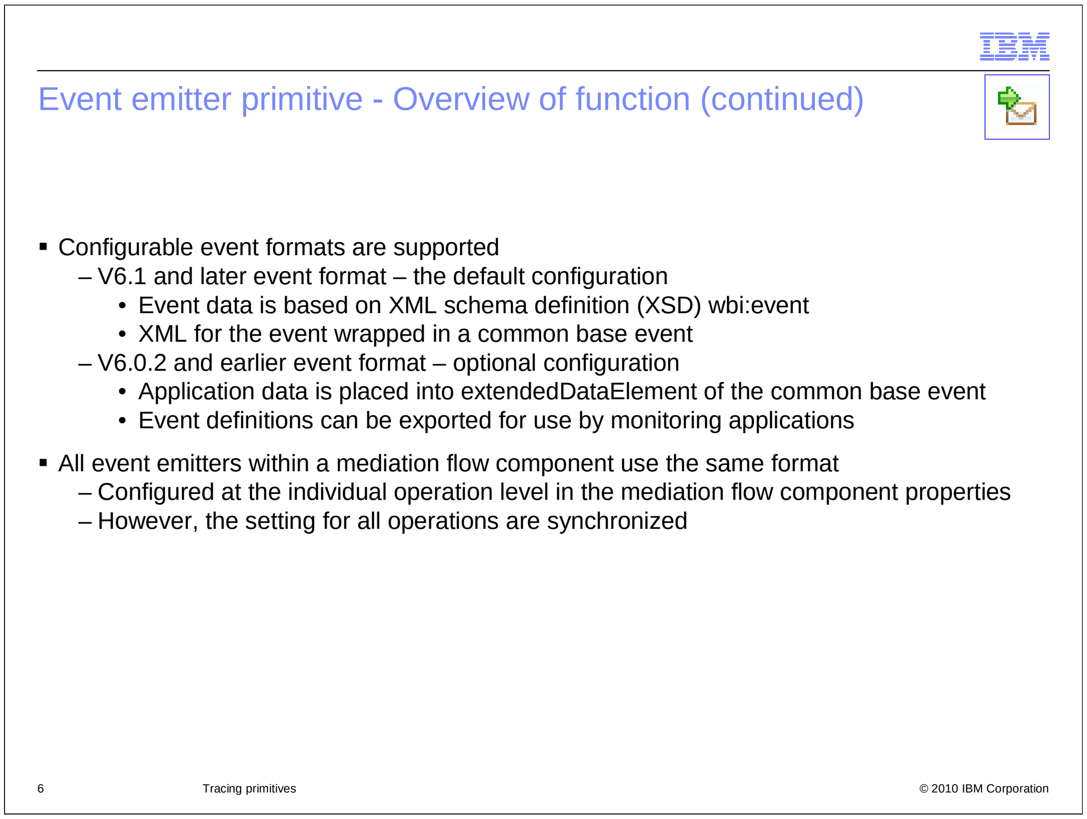
- Emits an event from within a mediation flow
  - Enables reporting of significant events within the flow
- Fully integrated with the common event infrastructure (CEI)
  - Common base event format
  - Generated events are sent to the CEI server
    - Can be written to the event database
    - Can be 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

5 Tracing primitives © 2010 IBM Corporation

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 a common base event and 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 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.



Event emitter primitive - Overview of function (continued)

- Configurable event formats are supported
  - 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
- All event emitters within a mediation flow component use the same format
  - Configured at the individual operation level in the mediation flow component properties
  - However, the setting for all operations are synchronized

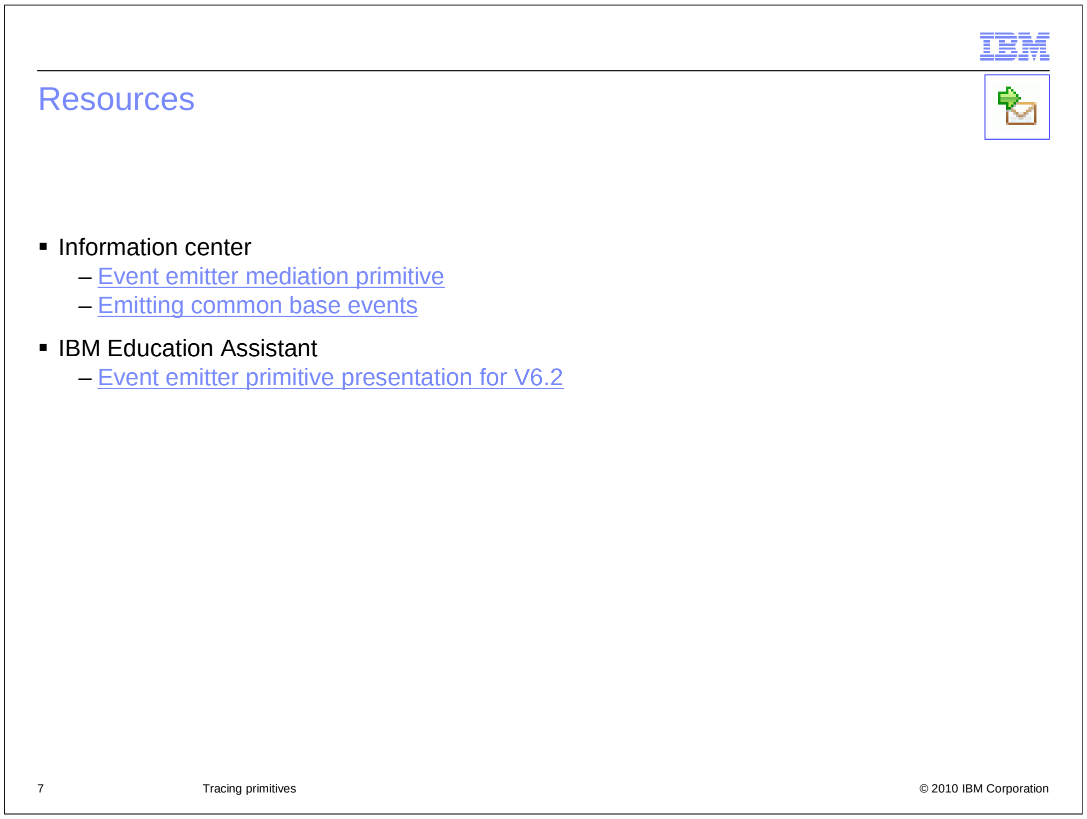
6 Tracing primitives © 2010 IBM Corporation

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

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 is only present 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 formats is specified at the individual operations level of the mediation flow component; 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 all the flows for the mediation flow component use the same event format.



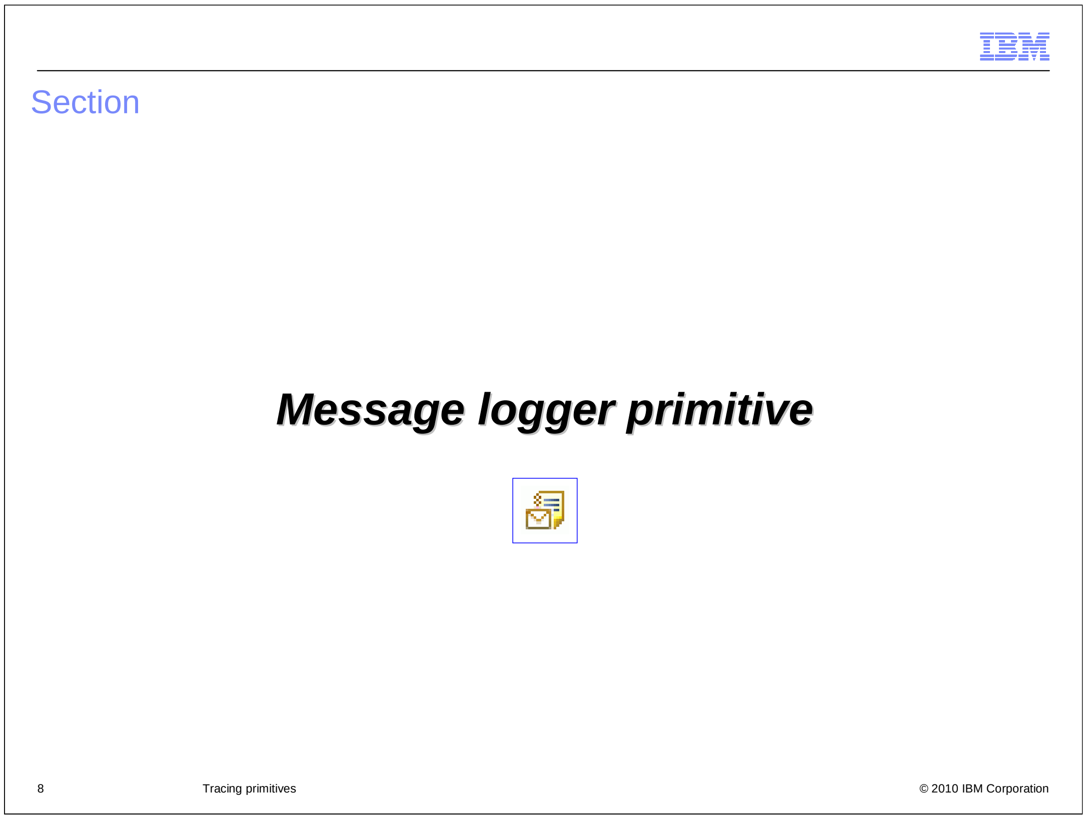
Resources

- Information center
  - Event emitter mediation primitive
  - Emitting common base events
- IBM Education Assistant
  - Event emitter primitive presentation for V6.2

7 Tracing primitives © 2010 IBM Corporation

The first link is to the information center for the event emitter mediation primitive documentation. It provides the details of the primitive and its use, including a description of its properties and information about the two different event formats that are supported. The next link in the information center, emitting common base events, is to a page containing many other links that will help you understand the big picture of using event emitters with your mediation applications. It includes topics such as the common base event infrastructure and common base events, guidelines for selecting an event format, generating monitor models and best practices.

There is a more detailed presentation on the event emitter primitive in IBM Education Assistant for V6.2. The presentation provides more details on usage, properties, event formats and best practices.

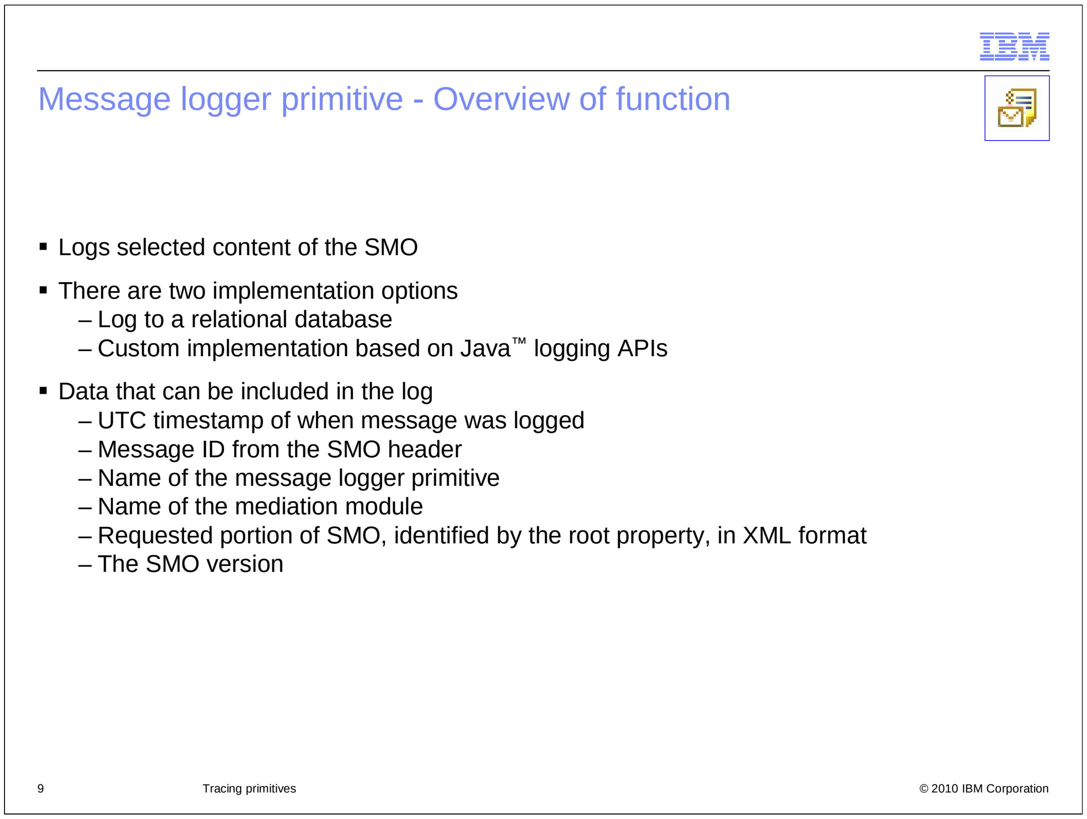


Section

Message logger primitive

8 Tracing primitives © 2010 IBM Corporation

This section provides an overview of the message logger primitive.



IBM

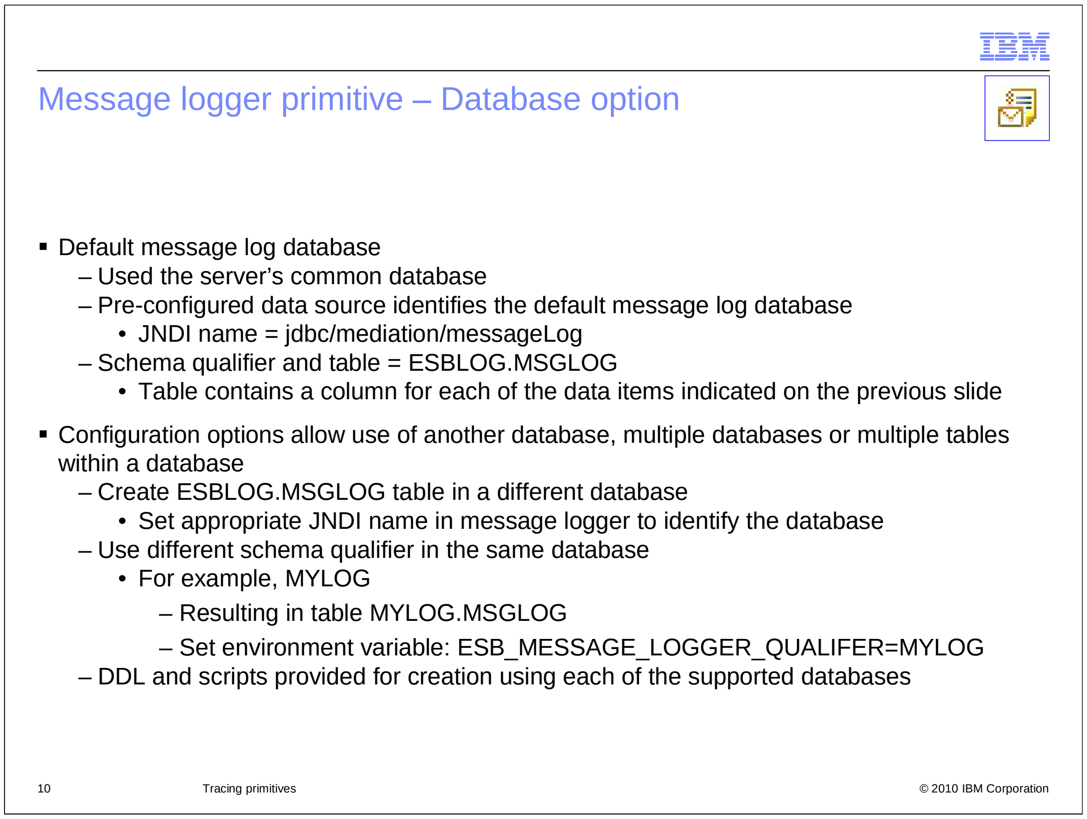
Message logger primitive - Overview of function

- Logs selected content of the SMO
- There are two implementation options
  - Log to a relational database
  - Custom implementation based on Java™ logging APIs
- Data that can be included in the log
  - UTC timestamp of when message was logged
  - Message ID from the SMO header
  - Name of the message logger primitive
  - Name of the mediation module
  - Requested portion of SMO, identified by the root property, in XML format
  - The SMO version

9 Tracing primitives © 2010 IBM Corporation

The purpose of a message logger primitive is to log selected content of the service message object (SMO). The message logger primitive provides you with the choice between two different implementations. One implementation writes log records to a relational database. The other implementation option is custom logging, which makes use of the Java logging APIs.

The data that can be logged includes the UTC timestamp of when the message was logged, the message ID taken from the SMO header, the names of the message logger primitive and mediation module, the portion of the SMO identified by an XPath expression in the root property, and the version of the SMO.



Message logger primitive – Database option

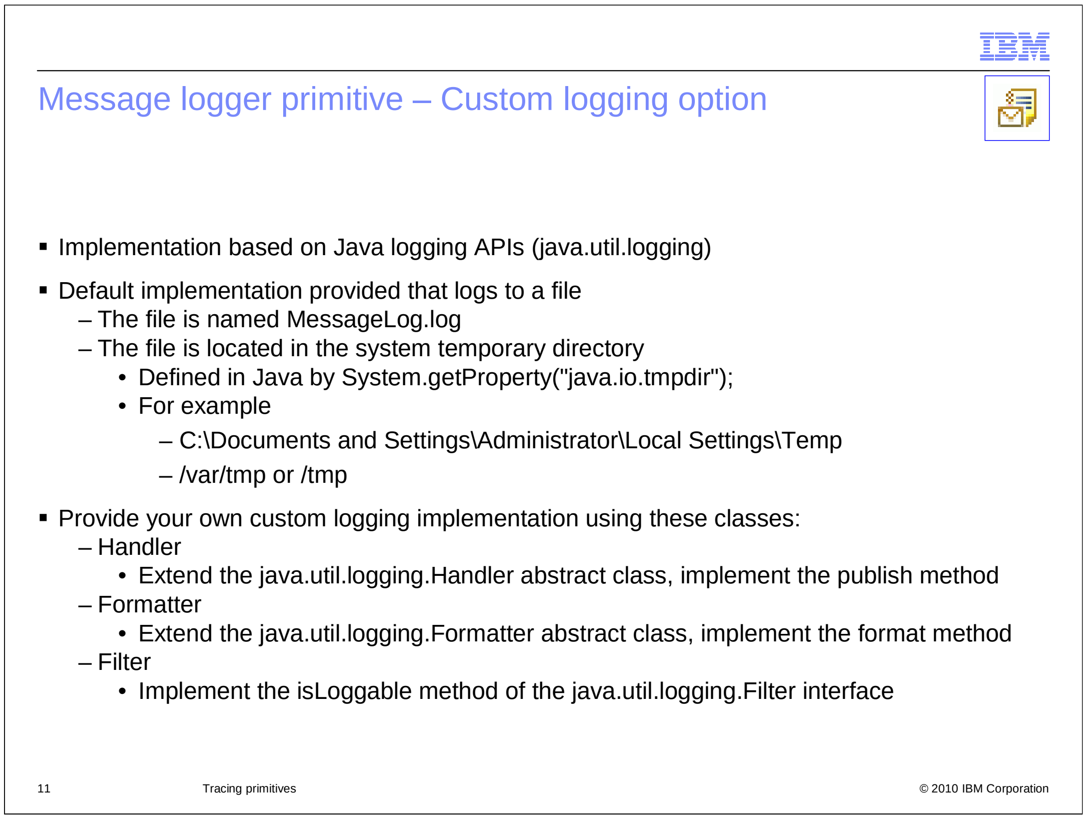
- Default message log database
  - Used the server's common database
  - Pre-configured data source identifies the default message log database
    - JNDI name = jdbc/mediation/messageLog
  - Schema qualifier and table = ESBLOG.MSGLOG
    - Table contains a column for each of the data items indicated on the previous slide
- Configuration options allow use of another database, multiple databases or multiple tables within a database
  - Create ESBLOG.MSGLOG table in a different database
    - Set appropriate JNDI name in message logger to identify the database
  - Use different schema qualifier in the same database
    - For example, MYLOG
      - Resulting in table MYLOG.MSGLOG
      - Set environment variable: ESB_MESSAGE_LOGGER_QUALIFER=MYLOG
  - DDL and scripts provided for creation using each of the supported databases

10 Tracing primitives © 2010 IBM Corporation

The database option provides you with a default message log database table located in the server's common database. The server is pre-configured to contain a JNDI name that identifies the message log database, with the JNDI name jdbc/mediation/messageLog. The message logger primitive writes to the database using the table named MSGLOG with a schema qualifier of ESBLOG. This table contains a column for each of the six data items indicated on the previous slide.

There are alternatives to the use of the ESBLOG.MSGLOG table in the common database. One approach is to use a different database that contains an ESBLOG.MSGLOG table. When configured this way, it is the use of the JNDI data source configured for the message logger that identifies the database to use. Another approach is to use the common database but write to a MSGLOG table that has a different schema qualifier, for example, MYLOG. When doing this, it is the use of the environment variable ESB_MESSAGE_LOGGER_QUALIFER that identifies the schema qualifier to use. Of course, the two approaches can be combined, using a different database in addition to different schema qualifiers.

The DDL and scripts needed to create a message log database are provided with the product, with variations for each of the supported database types.



Message logger primitive – Custom logging option

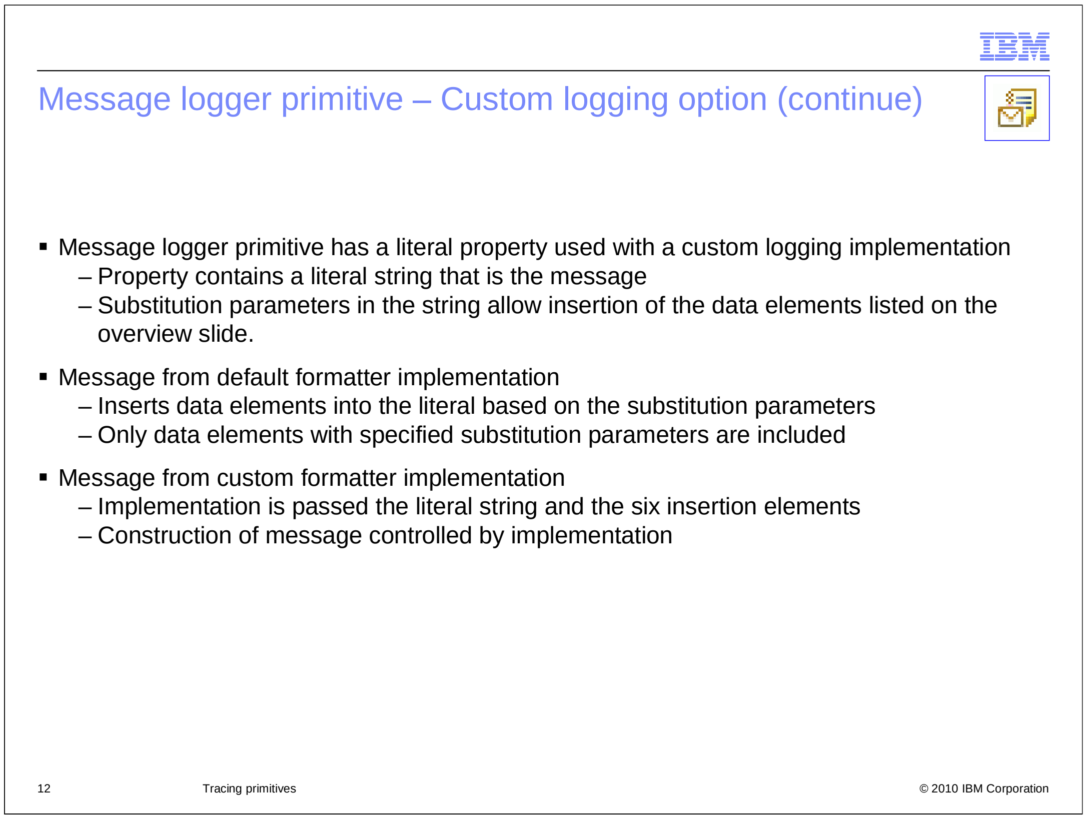
- Implementation based on Java logging APIs (java.util.logging)
- Default implementation provided that logs to a file
  - The file is named MessageLog.log
  - The file is located in the system temporary directory
    - Defined in Java by System.getProperty("java.io.tmpdir");
    - For example
      - C:\Documents and Settings\Administrator\Local Settings\Temp
      - /var/tmp or /tmp
- Provide your own custom logging implementation using these classes:
  - Handler
    - Extend the java.util.logging.Handler abstract class, implement the publish method
  - Formatter
    - Extend the java.util.logging.Formatter abstract class, implement the format method
  - Filter
    - Implement the isLoggable method of the java.util.logging.Filter interface

11 Tracing primitives © 2010 IBM Corporation

The custom logging implementation is based on the Java logging APIs, defined in the Java package java.util.logging.

The default implementation provided with the server logs messages to a file named MessageLog.log. This file is located in the system temporary directory that is identified by the Java property java.io.tmpdir. The value of this will vary by system, but typical locations are C:\Documents and Settings\Administrator\Local Settings\Temp on a Windows® system and /var/tmp or /tmp on a UNIX® system.

When you are providing your own custom implementation, there are three classes that need to be implemented. The abstract class java.util.logging.Handler needs to be extended to implement the publish method. This is the method that writes the formatted log message. The abstract class java.util.logging.Formatter needs to be extended to implement the format method, which does the formatting of the log message. Finally, the interface java.util.logging.Filter needs to be implemented with the isLoggable method providing an indication if this message should be logged. This might always return true, unless you have some filtering criteria by which you determine which messages should actually be logged.



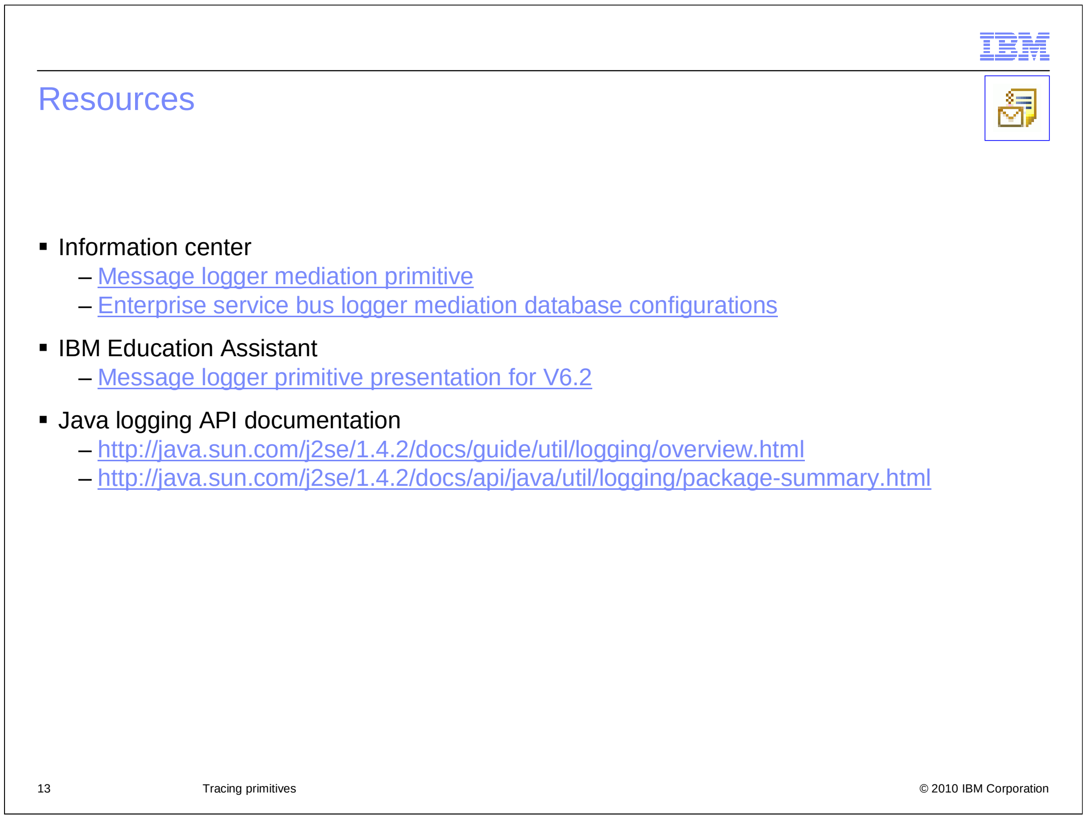
Message logger primitive – Custom logging option (continue)

- Message logger primitive has a literal property used with a custom logging implementation
  - Property contains a literal string that is the message
  - Substitution parameters in the string allow insertion of the data elements listed on the overview slide.
- Message from default formatter implementation
  - Inserts data elements into the literal based on the substitution parameters
  - Only data elements with specified substitution parameters are included
- Message from custom formatter implementation
  - Implementation is passed the literal string and the six insertion elements
  - Construction of message controlled by implementation

12 Tracing primitives © 2010 IBM Corporation

When using the custom logging option, the message logger has a property called literal that contains the message to be logged. The string provided can contain substitution parameters for the six variable data elements listed on the overview slide of this section. When using the default implementation, the variable data elements are inserted into the literal string based on the occurrence of substitution parameters. Any elements without a corresponding substitution parameter are not included in the message.

If you are using your own implementation of the formatter, the literal string and the six variable elements are made available. Your implementation can uses them as it chooses to format the message.



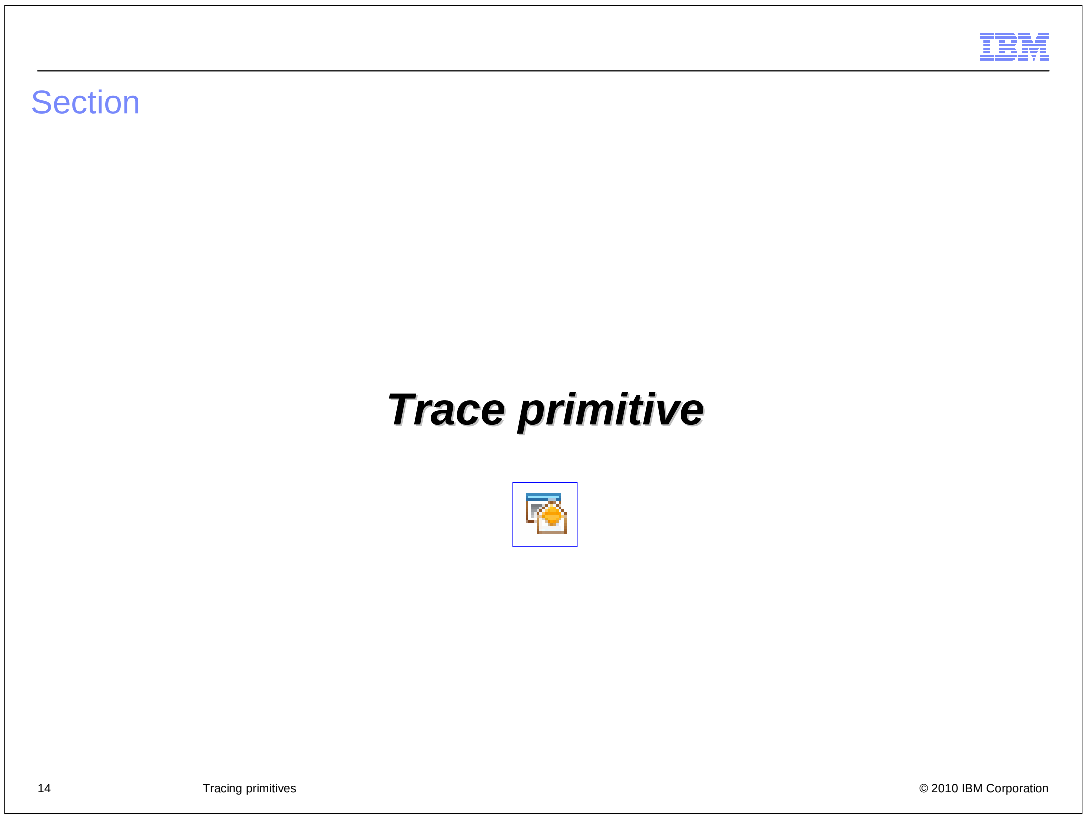
Resources

- Information center
  - Message logger mediation primitive
  - Enterprise service bus logger mediation database configurations
- IBM Education Assistant
  - Message logger primitive presentation for V6.2
- Java logging API documentation
  - http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/overview.html
  - http://java.sun.com/j2se/1.4.2/docs/api/java/util/logging/package-summary.html

13 Tracing primitives © 2010 IBM Corporation

On this slide are several links that you can reference to get a better understanding of the message logger primitive. The information center contains documentation of the primitive itself, describing the details of its usage for both options, defining all of the properties and providing several considerations for its usage. There are also links from there to other topics with further considerations when using the database option. There is another link on this slide to information about supported database types, configuration script names and their locations; profile creation configuration actions; schema upgrades and user ID privileges.

The next link is to IBM Education Assistant that provides a more in-depth presentation on the message logger primitive in V6.2. It provides more background on the two implementation options, including guidance on creating your own custom implementation. The last two links on the slide are to the documentation for the Java logging APIs, which you will need if developing your own custom implementation.

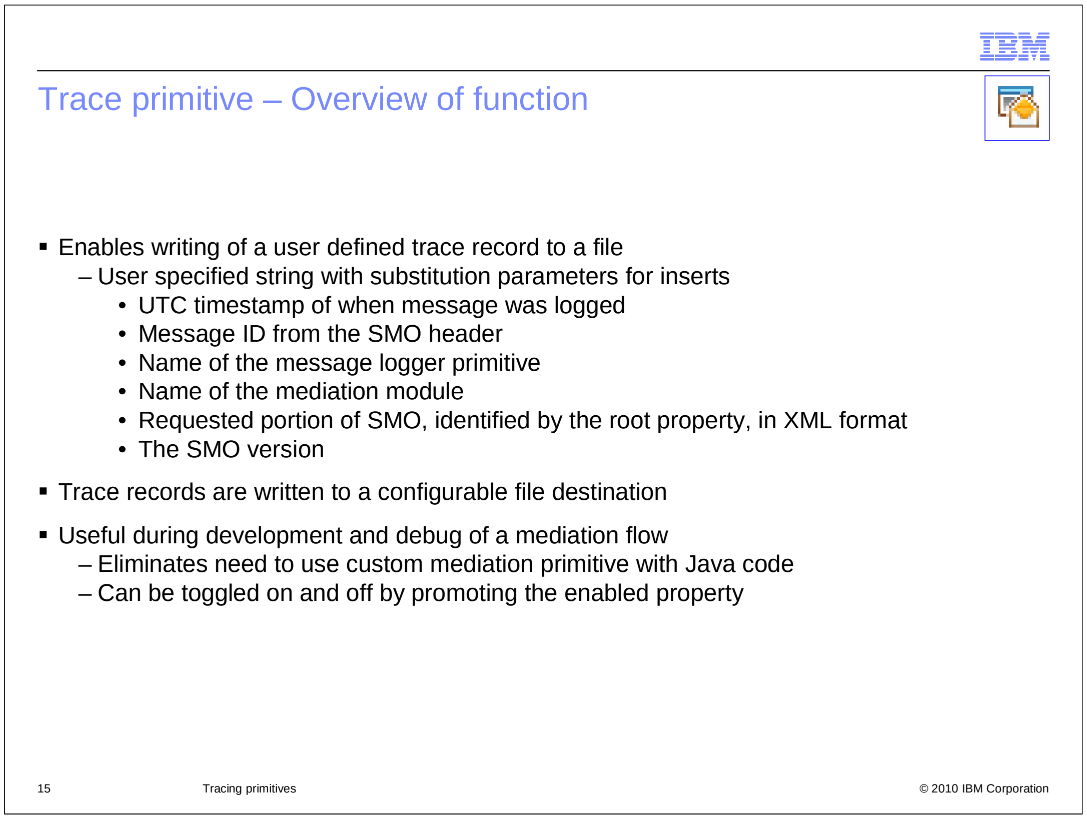


Section

Trace primitive

14 Tracing primitives © 2010 IBM Corporation

The trace primitive is examined in this section.



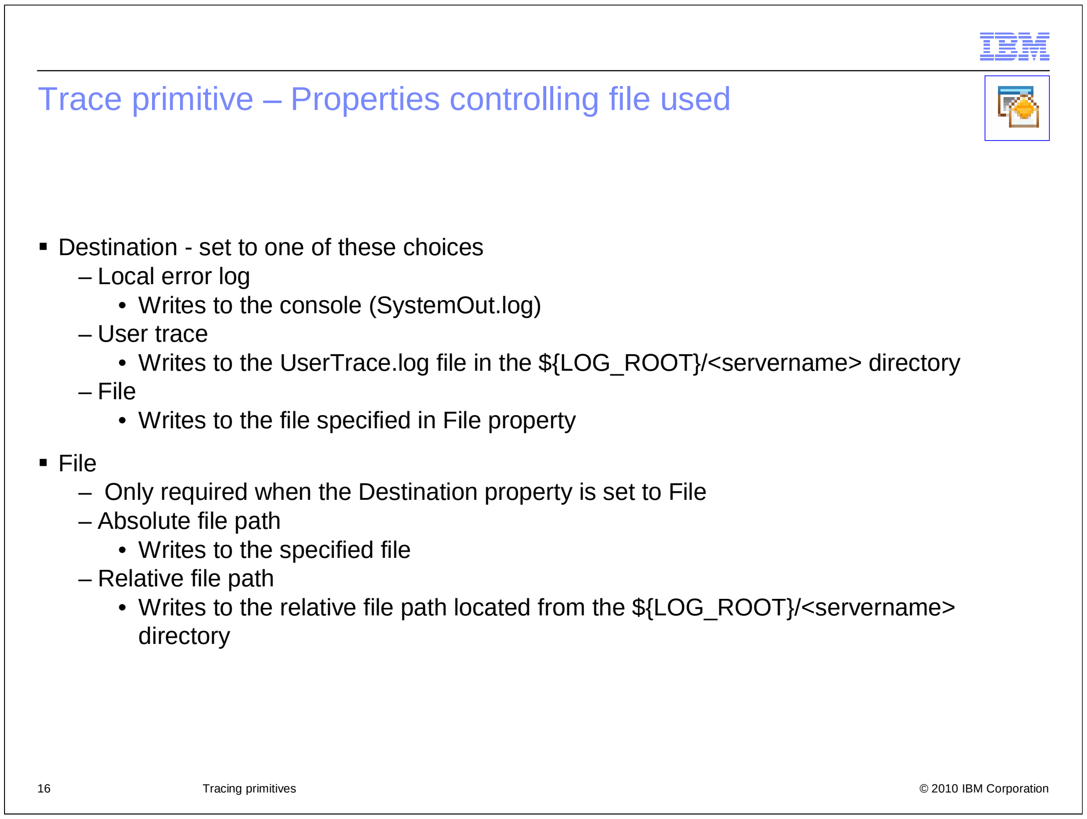
Trace primitive – Overview of function

- Enables writing of a user defined trace record to a file
  - User specified string with substitution parameters for inserts
    - UTC timestamp of when message was logged
    - Message ID from the SMO header
    - Name of the message logger primitive
    - Name of the mediation module
    - Requested portion of SMO, identified by the root property, in XML format
    - The SMO version
- Trace records are written to a configurable file destination
- Useful during development and debug of a mediation flow
  - Eliminates need to use custom mediation primitive with Java code
  - Can be toggled on and off by promoting the enabled property

15 Tracing primitives © 2010 IBM Corporation

The trace primitive enables the writing of a user defined trace record to a file. Similar to the message logger with custom logging option, the trace record is defined by a string containing substitution parameters identifying six variable elements that can be inserted. The substitution parameters are the UTC timestamp of when the message was written, the message ID taken from the SMO header, the names of the trace primitive and mediation module, the portion of the SMO identified by an XPath expression in the root property, and the version of the SMO.

The trace records are written to a configurable file destination, which is explained in more detail on the next slide. As was mentioned at the beginning of the presentation, the trace primitive is intended for use during the development and debugging of a mediation flow. The promotable enabled property can be used to toggle the actual tracing on and off, allowing code instrumented with trace primitives to be deployed to production without actually tracing. The trace record eliminates the need to insert custom mediation primitives with Java code to write trace records, as was commonly done in releases before V7.



Trace primitive – Properties controlling file used

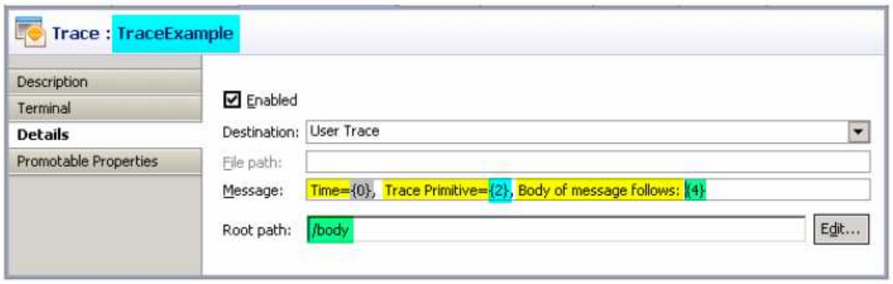
- Destination - set to one of these choices
  - Local error log
    - Writes to the console (SystemOut.log)
  - User trace
    - Writes to the UserTrace.log file in the ${LOG_ROOT}/<servername> directory
  - File
    - Writes to the file specified in File property
- File
  - Only required when the Destination property is set to File
  - Absolute file path
    - Writes to the specified file
  - Relative file path
    - Writes to the relative file path located from the ${LOG_ROOT}/<servername> directory

16 Tracing primitives © 2010 IBM Corporation

The file to which the trace records are written is controlled by the destination and file properties. The destination property is set to one of three values. Setting local error log causes the trace records to be written to the console, which is the SystemOut.log file. The setting user trace causes the trace records to be written to a file named UserTrace.log that is located in the $(LOG_ROOT)/<servername> directory. $(LOG_ROOT) is typically the logs directory in the server profile. Finally, if the destination property is set to file, the file property defines the file to be used. This is the only case in which the file property is used. If it contains an absolute path, the trace records are written to that file. If the file property contains a relative path, then trace records are written to that file relative to the $(LOG_ROOT)/<servername> directory.

IBM

Trace – Example output



☑ Enabled

Destination: User Trace

File path:

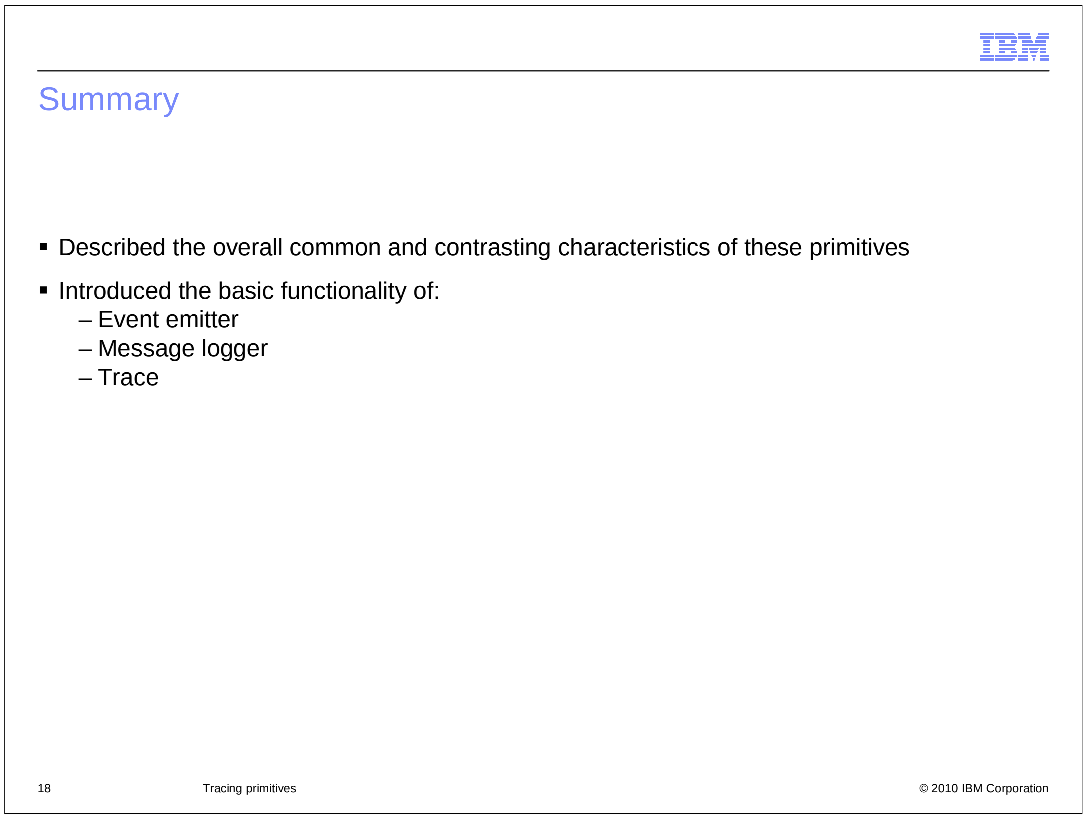
Message: Time={0}, Trace Primitive={2}, Body of message follows: {4}

Root path: /body

```
Time=7/15/09 5:00 PM, Trace Primitive=TraceExample, Body of message=<?xml version="1.0" encoding="UTF-8"?>
<body xsi:type="med:doItRequestMsg" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:med="wsdl.http://TestMediation/TestMed" xmlns:med_1="http://TestMediation/TestMed">
   <med_1:doIt>
      <in>This is the input parameter</in>
   </med_1:doIt>
</body>
```

17 Tracing primitives © 2010 IBM Corporation

This is an example of the properties for a trace primitive along with a corresponding trace record. Notice the property message, which contains the literal string for the message including substitution parameters. The property message and the actual message are color coded so that you can see the relationship. The yellow indicates the literal part of the message, the gray is the substitution parameter for the timestamp, and the yellow is the substitution parameter for the trace primitive’s name. The green is the substitution parameter for the SMO content, which is defined by the XPath expression in the root path property.

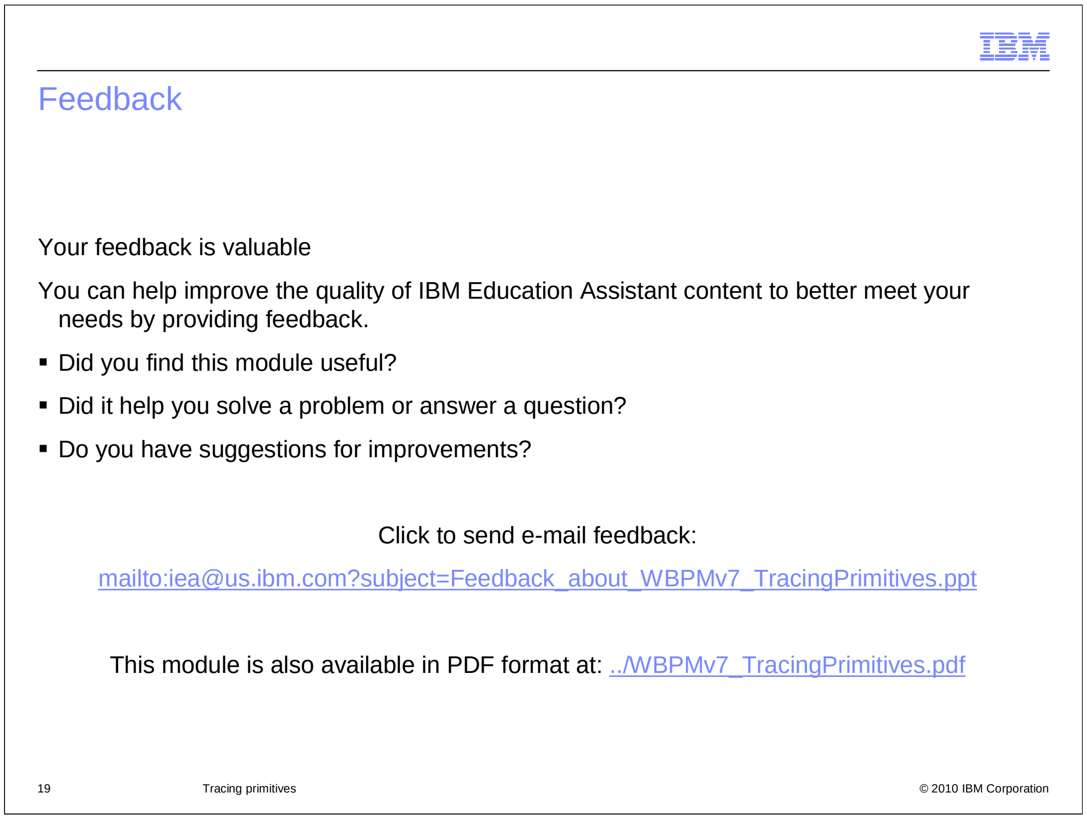


Summary

- Described the overall common and contrasting characteristics of these primitives
- Introduced the basic functionality of:
  - Event emitter
  - Message logger
  - Trace

18 Tracing primitives © 2010 IBM Corporation

The presentation started out by explaining the common and contrasting characteristics of these primitives, and then provided a description of the basic functionality of each. The primitives presented were the event emitter primitive, the message logger primitive and the trace primitive.



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?

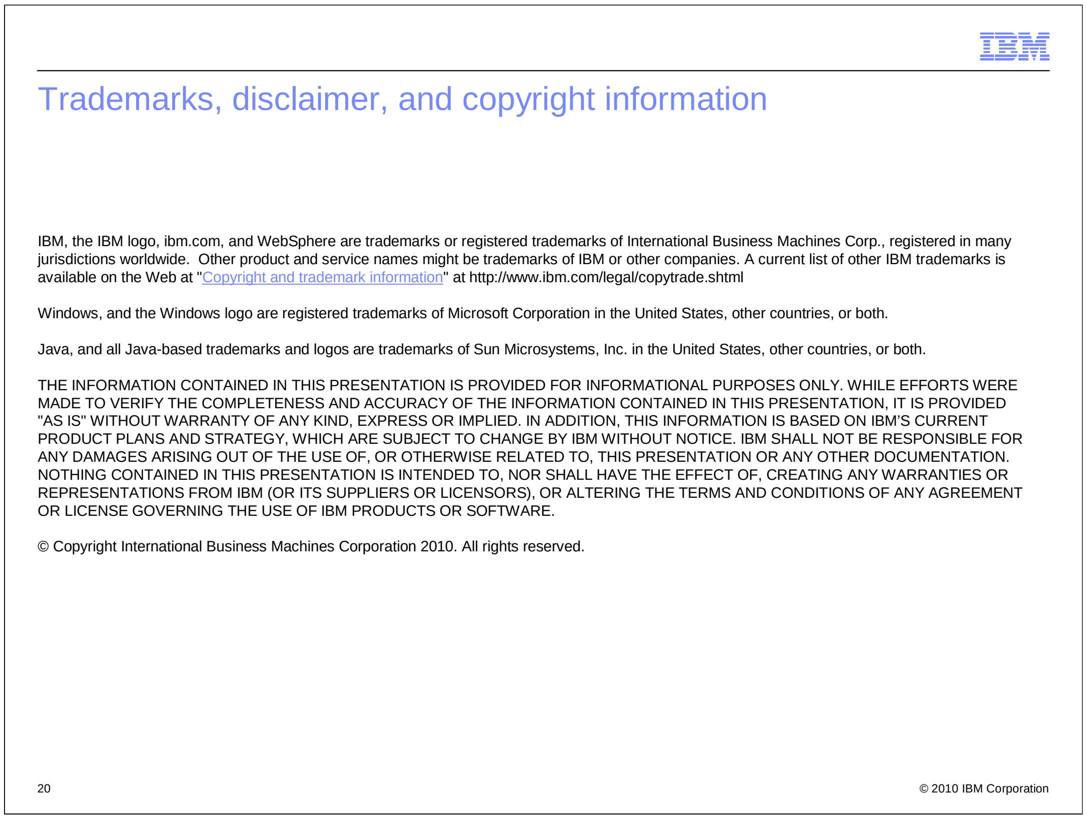
Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WBPMv7_TracingPrimitives.ppt

This module is also available in PDF format at: ../WBPMv7_TracingPrimitives.pdf

19 Tracing primitives © 2010 IBM Corporation

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



Trademarks, disclaimer, and copyright information