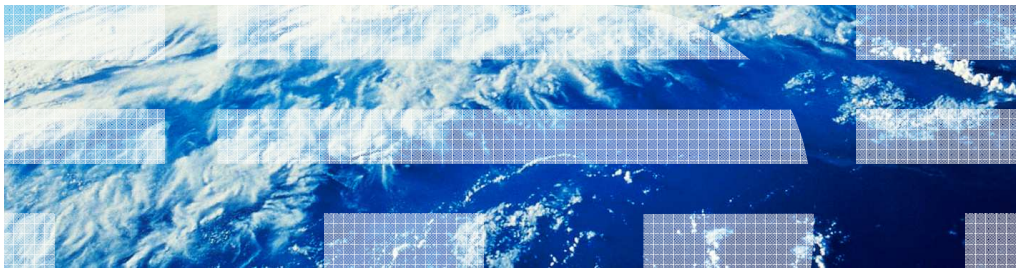
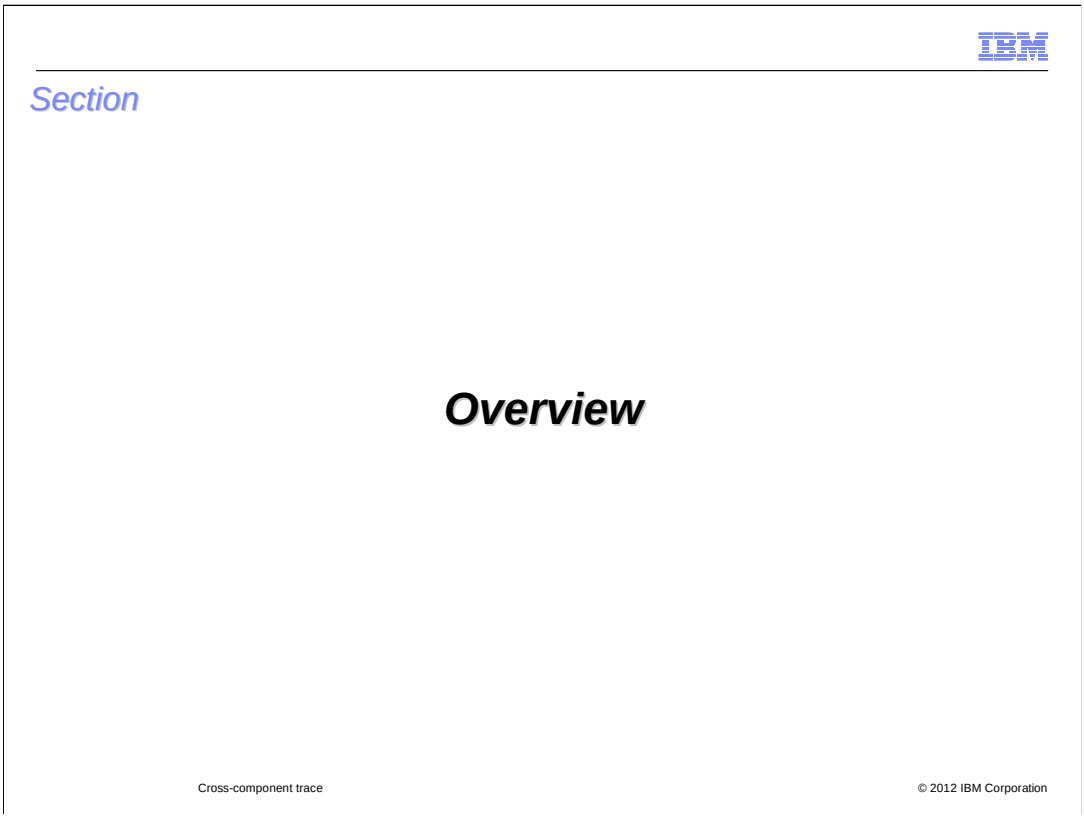
# IBM WebSphere Application Server V8.5

## Cross-component trace





This presentation describes support for cross-component trace (XCT) included in IBM WebSphere® Application Server V8.5



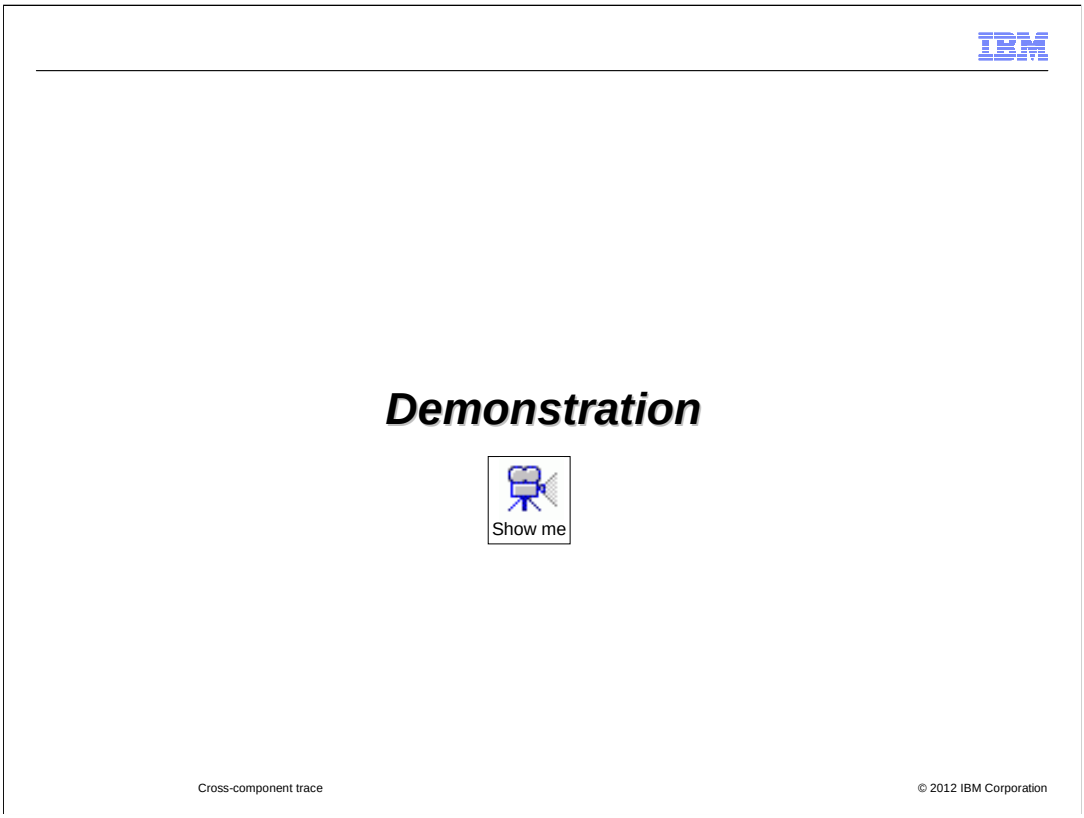

This section is an overview of cross component trace.

## What is cross component trace?

- Cross component trace (XCT) is a log/trace correlation technology.
- XCT enables you to determine which log/trace entries are part of each request.
- XCT can be used in any of three different modes:
- Request ID mode
- Request ID and correlation log record mode
- Request ID, correlation log record, and data snapshot mode
- XCT works best in combination with High Performance Extensible Logging (HPEL).
- HPEL stores XCT request IDs
- HPEL can filter log/trace records by XCT request ID




Cross component trace is a correlation technology that helps administrators see the flow of requests that span multiple threads or processes. XCT simplifies the task of determining which log or trace entries, in each application server log file, are part of each request. When enabled, XCT can be used in any of three different modes. In the first mode, XCT request IDs are added to existing log and trace records. In the second mode, XCT request IDs are added to existing log and trace records and XCT log records are added to log files. In the third mode, XCT request IDs are added to existing log and trace records, XCT log records are added to log files, and data snapshots are captured. XCT works best with the application server's High Performance Extensible Logging (HPEL) log and trace framework. XCT Request Ids are only ever added to HPEL log and trace records – they cannot be stored in SystemOut.log. HPEL also provides the ability to filter log and trace files by request ID and helps minimize the performance impact of enabling XCT log records.




For a demonstration of how to enable high performance extensible logging, pause this presentation and click the demonstration icon.

## What is cross component trace?

- IBM WebSphere Cross Component Trace Logviewer can be used to view files augmented with correlation log records.
  - Available for the IBM Support Assistant
  - Can load multiple files simultaneously
  - Can show flat or hierarchical views




A tool called IBM WebSphere cross component trace log viewer can be used to view log or trace files augmented with correlation log records. The tool is available as a tool add-on for the IBM Support Assistant. It is able to load multiple log files at the same time, and can display log content in either a flat chronologically-ordered view, or a hierarchical request-ordered view.

## What is cross component trace?

- XCT comparison with PMI Request Metrics
  - XCT is for log and trace correlation
  - PMI Request Metrics is for performance tracking




XCT and PMI Request Metrics have some overlap. Both technologies provide transaction tracking.

XCT is used for log and trace correlation, making it easy to see which log and trace entries are part of the same requests. XCT can also be used to capture request and response payload data. XCT instrumentation is sparse as it aims mostly to track where requests change threads or processes.

PMI Request Metrics is used for performance tracking. PMI Request Metrics can expose its data to Application Response Measurement (ARM) agents. PMI Request Metrics instrumentation can be verbose as it tracks the performance of individual components.

## Understanding XCT correlation log records

- Correlation log records look like this example:

```
[4/23/12 13:54:44:509 IST] 0000008e XCT           I   BEGIN AAADx/itMDz-
  AAAAAAAAAAA 00000000000-ccccccccc2 HTTPCF(InboundRequest
  /JMSApp/LocalMessageSend RemoteAddress(127.0.0.1) RequestContext(2082603117))
```

- Each XCT correlation log record is structured as shown below
  <Date> <Thread_ID> <XCT_Logger_Name> <Message_Type> <XCT_STATE>
  <XCT_ID> <XCT_PARENT_ID> <XCT_MESSAGE>
  - <Date>: The date and time when the log record was generated
  - <Thread_ID>: The thread which generated this message
  - <XCT_Logger_Name>: The XCT logger name is XCT. This logger is used to identify the XCT Records in the Log file
  - <Message_Type>: Type of the log message
  - <XCT_STATE>: Each XCT Record has a State, it can be BEGIN, END
  - <XCT_ID>: A Unique ID generated for correlating the XCT Records
  - <XCT_PARENT_ID>: The XCT_ID of the parent XCT context
  - <XCT_MESSAGE>: The XCT message contains the information about the XCT record; this can contain some Associations and Annotations

Cross-component trace



Each XCT record contains a date, thread ID, XCT logger name, message type, XCT state, XCT ID, XCT Parent ID, and message.

The date is the date and time when the message was generated.

The thread ID is the thread which generated the message.

The XCT logger name is the logger name in XCT, the logger is used to identify the XCT records in the log file.

The message type is the type of the log message.

The XCT state identifies whether an XCT context is beginning or ending. Think of an XCT context as a request running on a particular thread.

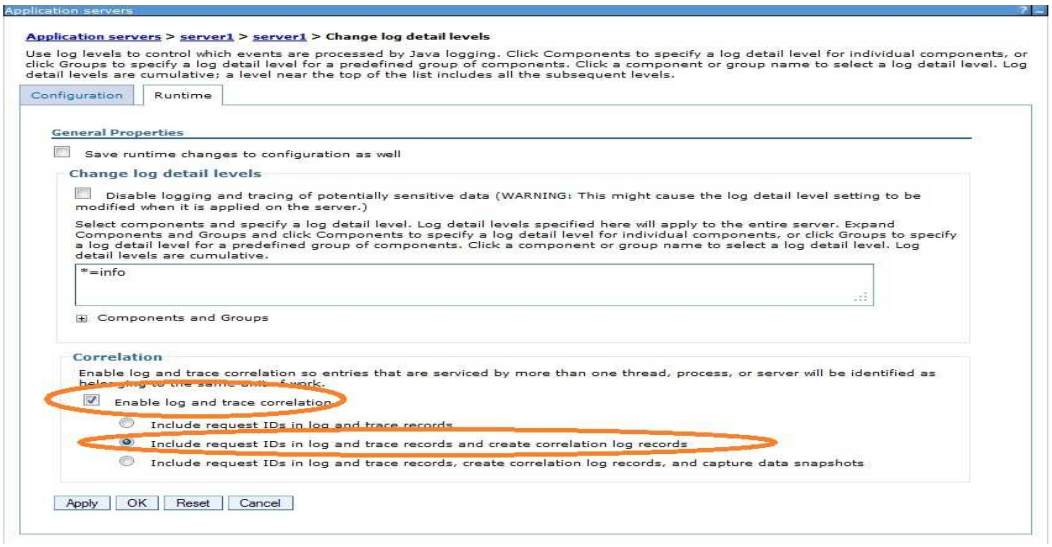
The XCT ID is the unique ID generated to correlate the XCT records.

The XCT parent ID is the XCT ID of the parent request.

The message contains the information about the XCT record; this can contain Associations and Annotations.

## Enabling XCT through administrative console

- The panel below is where XCT is enabled on the administrative console
  - The panel can be found here: WebSphere Application Servers > SERVER_NAME > Change log detail levels



Cross Component Trace (XCT ) can be enabled using the administrative console or WSADMIN scripts. In this illustration XCT is enabled using administrative console.

To enable XCT for the server , server1, navigate as follows: Servers > Server Types > WebSphere Application Servers > server1 > Change log detail levels > Runtime Tab

Select the check box that says Enable log and trace correlation and select the radio button labeled Include request IDs in log and trace records and create correlation log records

Section

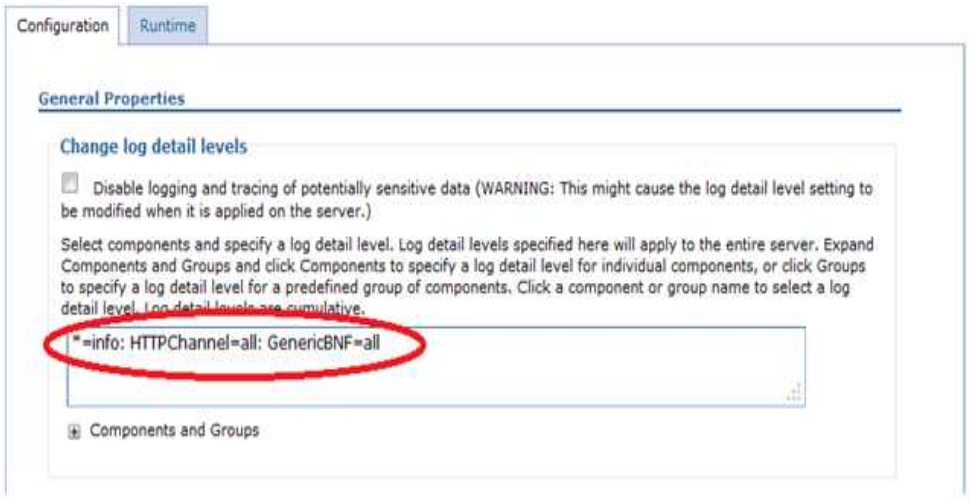
# *Usage scenarios*




Cross component trace is used in these scenarios.

## HTTP trace

- For the HTTP scenarios these traces were enabled
  - The panel below can be found on this page: WebSphere Application Servers > SERVER_NAME > Change log detail levels







For the HTTP scenarios to follow this trace was enabled:
*=info:HTTPChannel=all:GenericBNF=all

## Identifying all log/trace entries that are part of the same HTTP request

- The XCT requestID is added to all log and trace records associated with HTTP requests.
  - The requestID can only be seen when using the HPEL logViewer command-line tool with the advanced format
- Below is an example of logViewer output with the requestID circled in red
- The logViewer can be found within the bin directory on the server
  - Example: `logViewer.bat -format advanced`
  - The logViewer will display all log and trace entries

```
[5/9/12 19:27:20:843 EDT] 0000002b  I UOW= source=com.ibm.websphere.XCT
thread=[WebContainer : 2] requestID=[AABZvPwW/cp-AAAAAAAAAAH]
          BEGIN AABZvPwW/cp-AAAAAAAAAAH 00000000-cccccccccc2 HTTPCF(InboundRequest
          /snoop RequestContext(1283695288))
[5/9/12 19:27:20:843 EDT] 0000002b  3 UOW=
source=com.ibm.ws.http.channel.impl.HttpServiceContextImpl org=IBM prod=WebSphere
component=Application Server thread=[WebContainer : 2] requestID=[AABZvPwW/cp-AAAAAAAAAAH]
          parseMessage() returning true for
          com.ibm.ws.http.channel.impl.HttpRequestMessageImpl@876d44e5
[5/9/12 19:27:20:844 EDT] 0000002b  1 UOW=
source=com.ibm.ws.http.channel.impl.HttpBaseMessageImpl org=IBM prod=WebSphere
component=Application Server thread=[WebContainer : 2] requestID=[AABZvPwW/cp-AAAAAAAAAAH]
          Initializing message:
          com.ibm.ws.http.channel.impl.HttpResponseMessageImpl@89f7e3fc with
          com.ibm.ws.http.channel.inbound.impl.HttpInboundServiceContextImpl@4c83a2b8
```

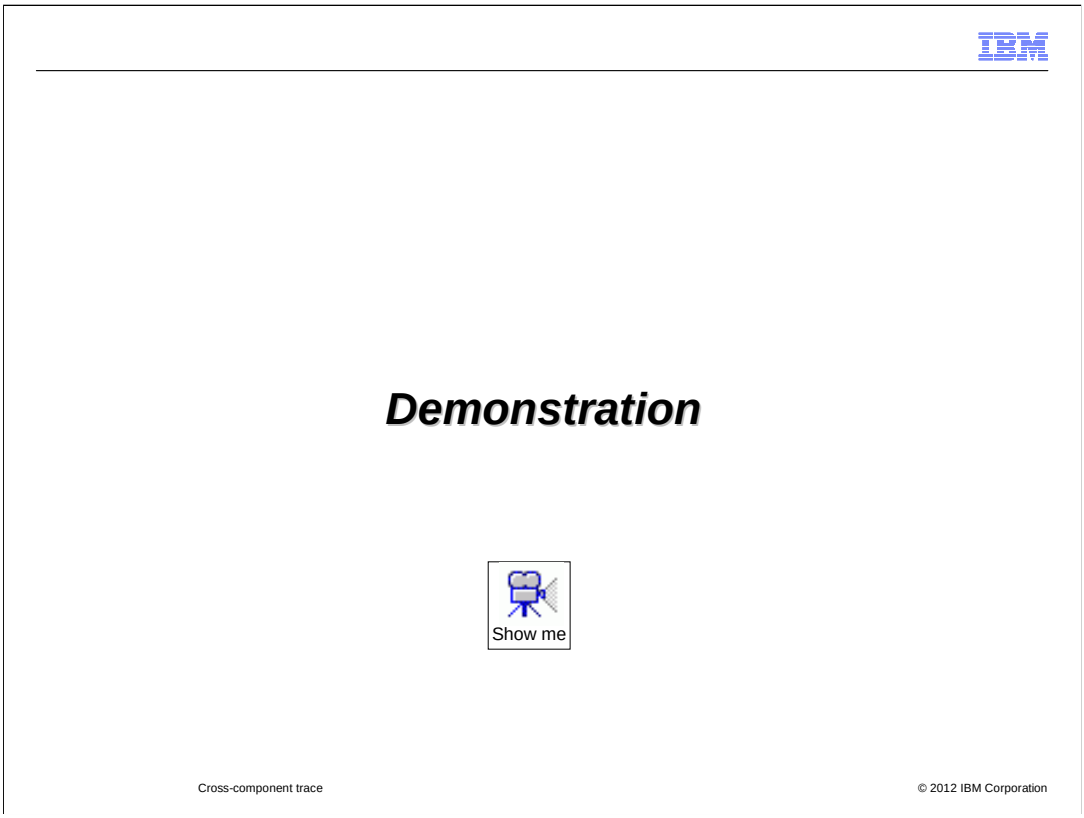
11 Cross-component trace 

An administrator might want to use XCT to identify what trace entries are part of an HTTP request. To accomplish this the HPEL logViewer command-line tool can be used. This tool is found within the bin directory of the server. When the “logviewer” is run with the advanced format option, the requestID can be seen on each trace entry. In order to have the requestID present XCT must be enabled.

To search for log and trace records that match a particular requestID, use the command logViewer -includeExtensions requestID=<some Id>.

For example:

logViewer -includeExtensions requestID=AABZvPwW/cp-AAAAAAAAAAH




For a demonstration of how to Use request IDs to see log and trace entries related to a particular request, pause this presentation and click the demonstration icon.

## Identifying HTTP requests the server is executing

- When an HTTP request arrives, the server executes an XCT BEGIN
  - Indicates the request has started processing
  - The entry in the logs show this information:
    - Parent XCT ID
    - Current XCT ID
    - Type of request(InboundRequest or OutboundRequest)
    - URI of request
    - RequestContext object ID from HTTPChannel
    - RemoteAddress from the connection the request originated from
  - Will only display if XCT correlation log records are enabled

```
[5/29/12 7:15:29:787 EDT] 000000be XCT          I   BEGIN AABPtopIWgZ-AAAAAAA7oMO
AABPtopIWgZ-AAAAAAA7oK8 HTTPCF(InboundRequest /favicon.ico RemoteAddress(9.42.75.112)
RequestContext(-1245247681))
```




An administrator can see what HTTP requests the server is running. When the request arrives, the server does an XCT BEGIN. This marks the request as having started processing. In the logs an XCT BEGIN for an HTTP request can commonly be seen with attributes showing the XCT Parent ID, XCT Current ID, the type of request, which includes InboundRequest and OutboundRequest, the URI of the request, the HTTP Channel RequestContext object ID, and the Remote IP Address from the connection the request originated from.

## Identify what HTTP requests the server is executing

- When the request completes processing and XCT END is executed
  - Indicates the request is complete
  - The entry in the logs show this information:
    - Parent XCT ID
    - Current XCT ID
    - Type of request
    - Return Code of the response that is being returned
    - RequestContext object ID from HTTPChannel
  - Will only display if XCT correlation log records are enabled

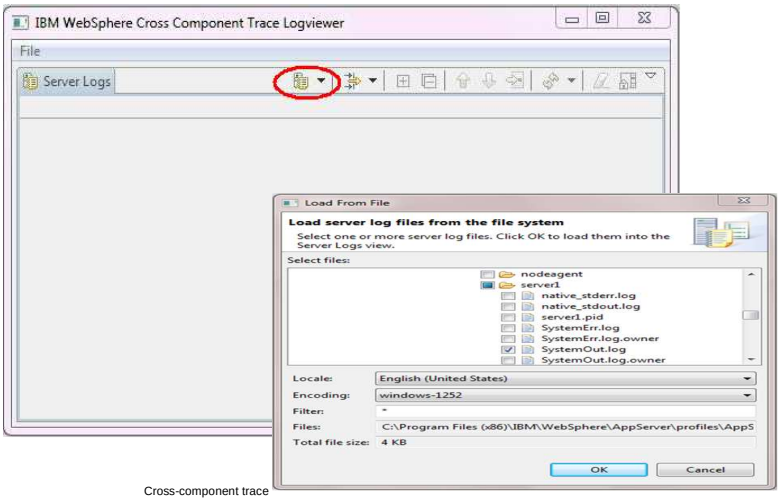
```
[5/29/12 7:15:29:886 EDT] 000000be XCT          I   END   AABPtopIWgZ-AAAAAAA7oMO
AABPtopIWgZ-AAAAAAA7oK8 HTTPCF(InboundRequest RC=404 RequestContext(-1245247681))
```




When the request completes processing the server does an XCT  END, this marks the request as finished. In the logs an XCT END for a HTTP request can commonly be seen with attributes showing the XCT Parent ID, XCT Current ID, the type of request, which includes InboundRequest and OutboundRequest, the return code of the response, and the HTTP Channel RequestContext object ID.

## IBM WebSphere cross component trace log viewer

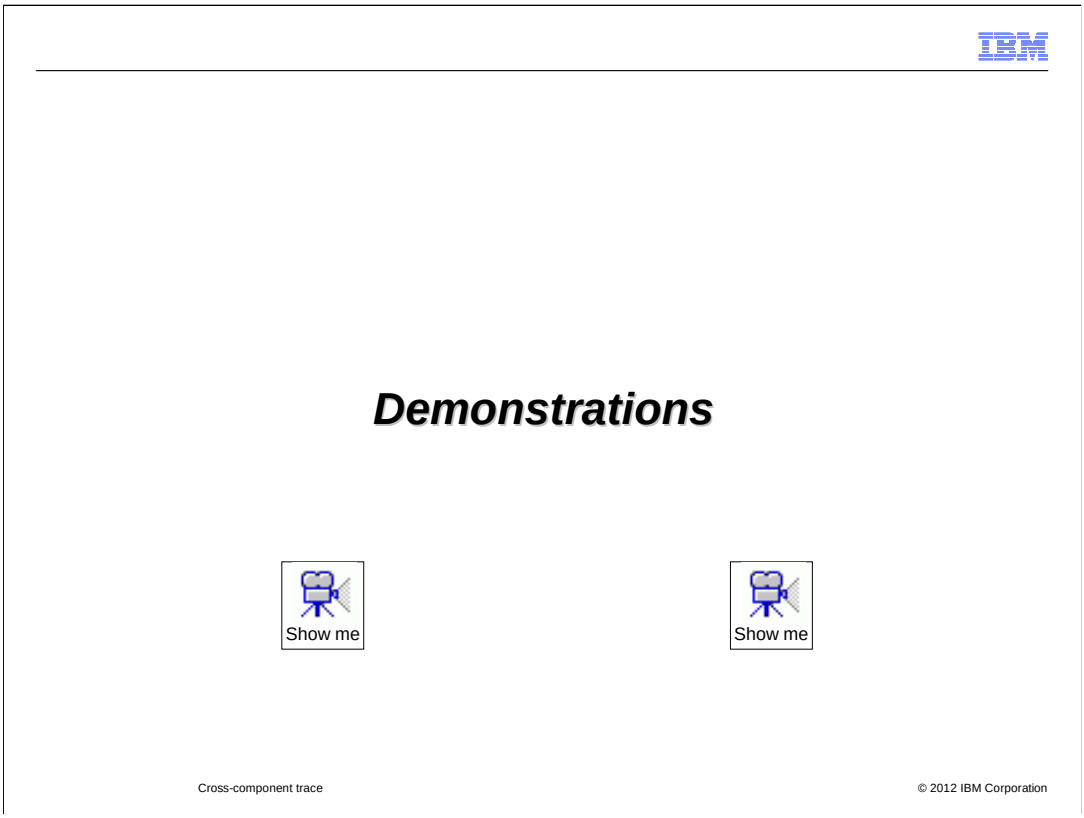
- The scenarios following this slide use the IBM WebSphere Cross Component Trace Logviewer – available as a tool add-on for the IBM Support Assistant
- Tool used to examine XCT entries in a log
- Logs can be loaded from multiple servers and they are stitched together





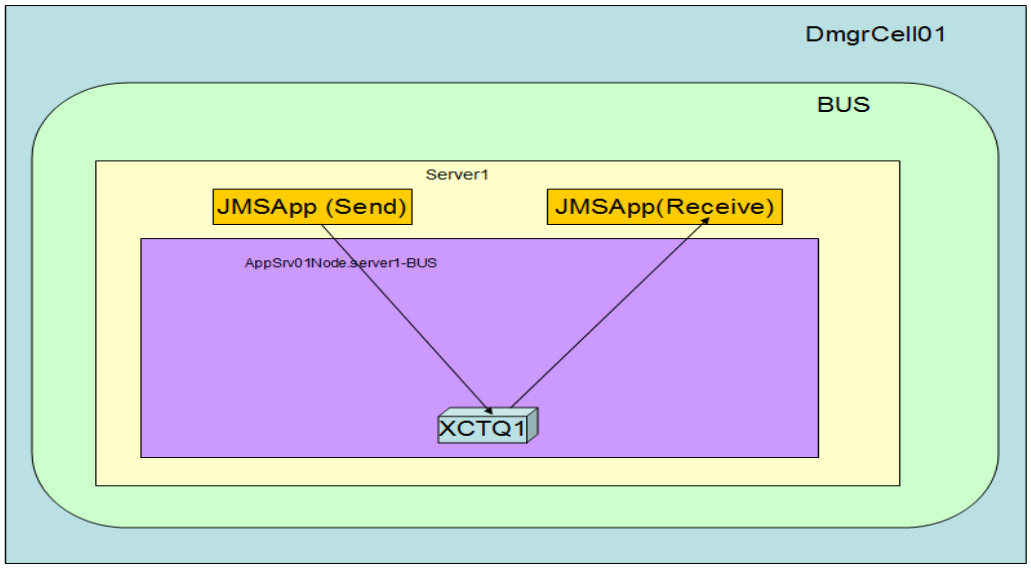


The scenario that follows uses the IBM WebSphere cross component trace log viewer. The tool is used to examine XCT entries from a server log. Log files from multiple servers can be loaded and they are stitched together for a combined view.




For demonstrations on how to get the cross-component trace log viewer and on how to see request hierarchy in the log view, pause this presentation and click each demonstration icon.

## Messaging topology for sending and receiving a JMS message from a local server scenario



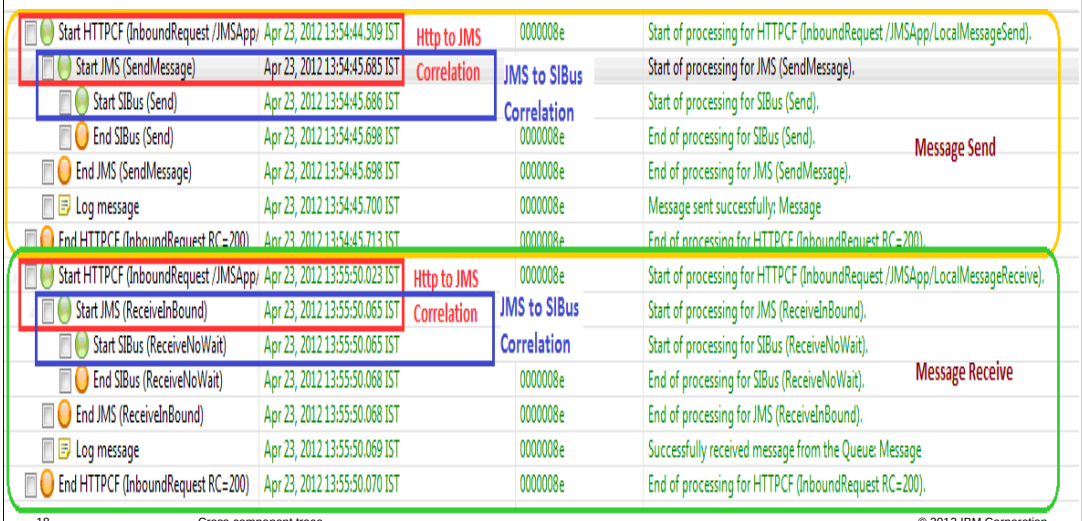

17 Cross-component trace 

In this scenario, the JMS application and messaging engine are running in the same server process.

The JMS message is sent to a local queue destination and the message is received from the local queue destination synchronously.

## Sending and receiving a JMS message from a local server

- JMS Applications and the Messaging Engine are running in the same process
- Message is received synchronously

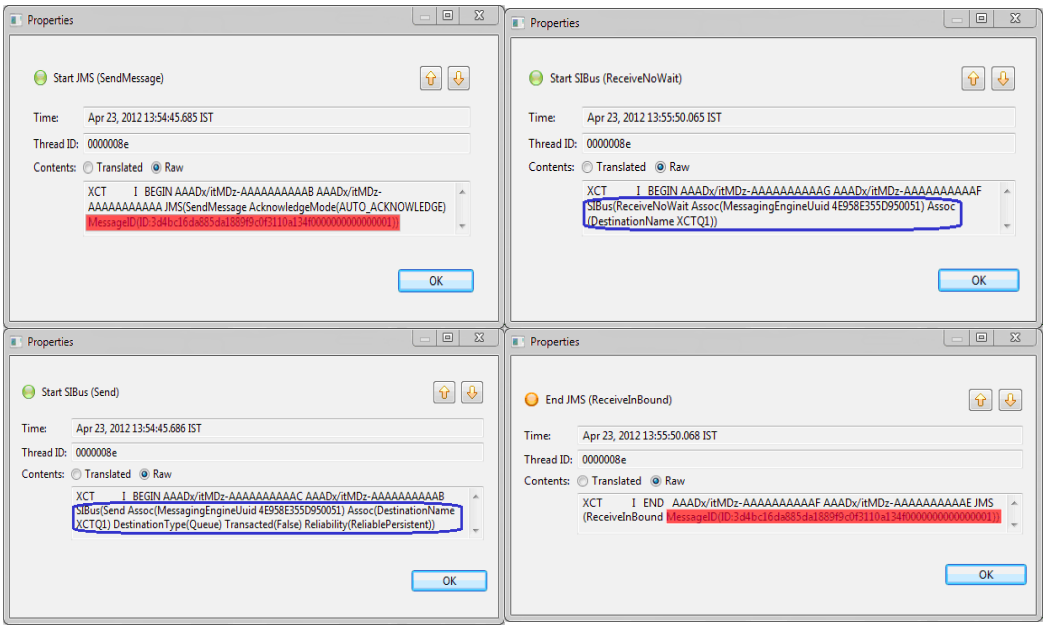



18 Cross-component trace 

Since the JMS application and messaging engine are running in the same server process, the SystemOut.log from that server is loaded

HTTP to JMS Correlation and JMS to systems integration bus correlation can be clearly seen in the IBM WebSphere Cross Component Trace Logviewer

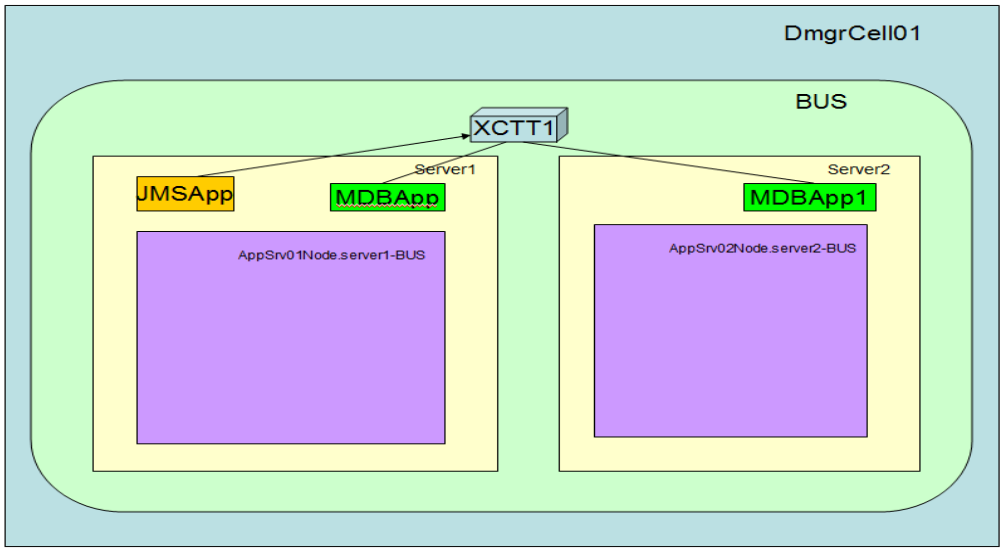
## JMS XCT records




19 Cross-component trace 

By double clicking the entry in the record list in the IBM WebSphere Cross Component Trace Logviewer the XCT records with annotations can be seen. In the JMS layer, the JMS Message ID is captured , which helps in correlating the message sent with the message received. In the systems integration bus layer, information related to the destination where the message is sent and from where the message is received is captured, such as messaging engine UUID , destination name etc.

## Messaging topology for receiving message asynchronously (PubSub – Topicspace) JMS scenario







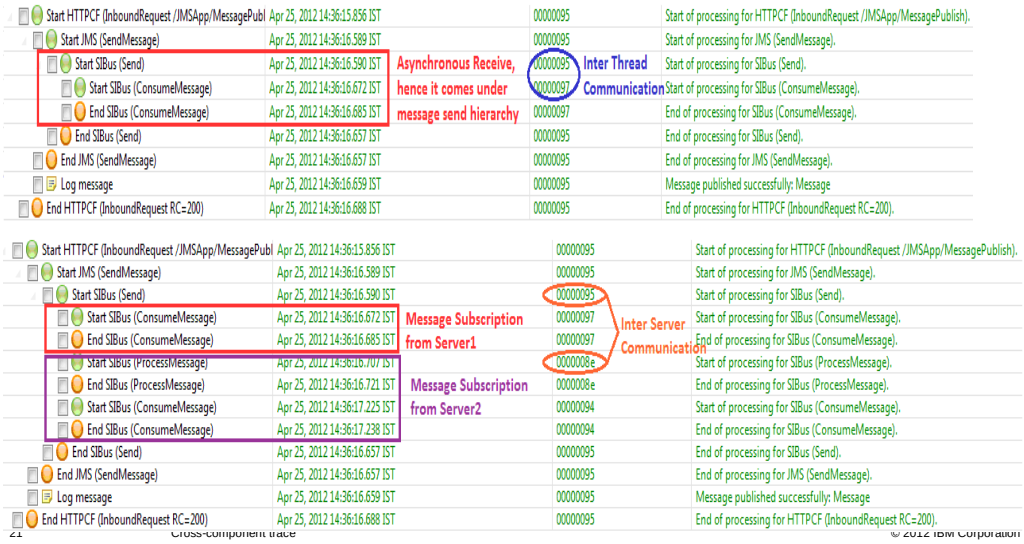
In this scenario, two servers are involved. Two MDB applications are deployed -- one in server1 and another in server2.

The JMS application running in server1 publishes a message to a topic that is subscribed to by the MDB applications running in server1 and server2

The message is asynchronously consumed by the MDB application.

## Receiving message asynchronously (PubSub - Topicspace)

- One MDB Application and the Messaging Engine are running in the same server process
- One MDB Application is running in a different server process
- Message is received asynchronously





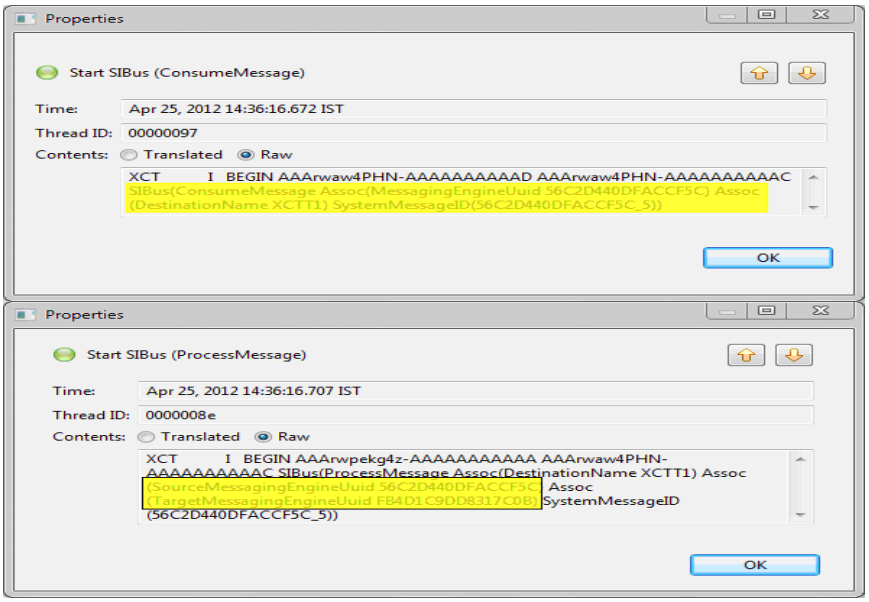


The SystemOut.log from server1 is loaded to show the inter-thread communication. Inter-thread communication is captured by XCT and displayed in IBM WebSphere cross component trace log viewer, where the JMS application and MDB application are running in the same server process.

The SystemOut.log from server1 and server2 are loaded to show the Inter server/process communication

Inter server/process communication is captured by XCT and displayed in the IBM WebSphere cross component trace log viewer, where the JMS application and the MDB application are running in different server processes.

## XCT records



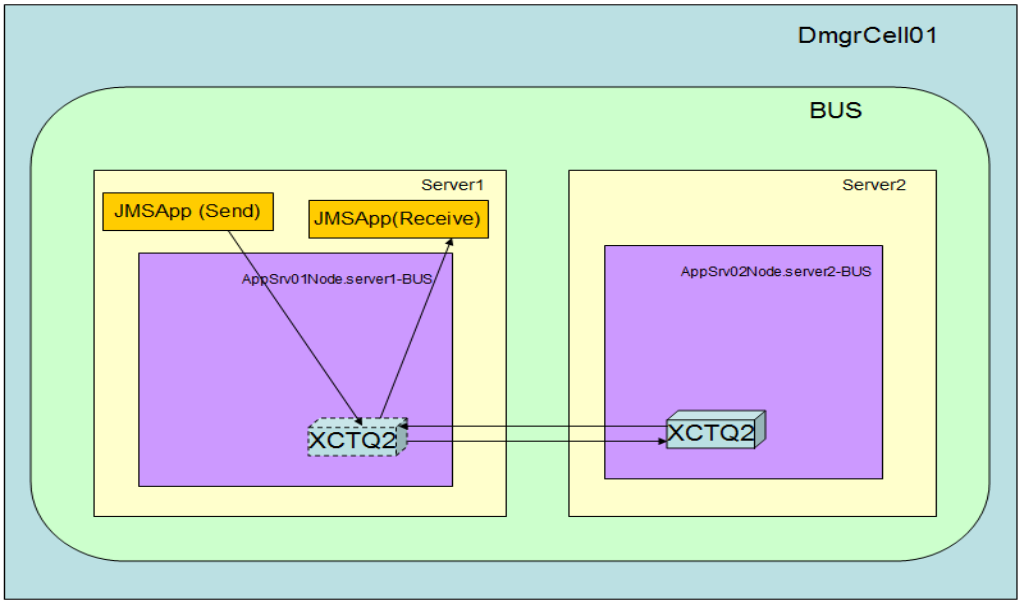




The top image shows the message being consumed by the first server, where the JMS application and the MDB application are running in the same server process with the messaging engine. The messaging engine UUID, the destination name from where the message is consumed and the system message ID are captured.

The bottom image shows the message being processed by the second server, where the MDB application runs remotely from the JMS application. The messaging engine UUID of both the source and target messaging engines, the destination name where the message is consumed, and the system message ID are captured.

## Messaging topology for store and forward and remote get scenario




23 Cross-component trace 

In this scenario, the JMS application deployed in server1 sends a message to a destination in server2 and receives a response message from that remote destination

The message is sent from server1 to server2 and a response is sent back from server2 to server1

## Store and forward and remote get

Sending a message to, and receiving a message from a remote server



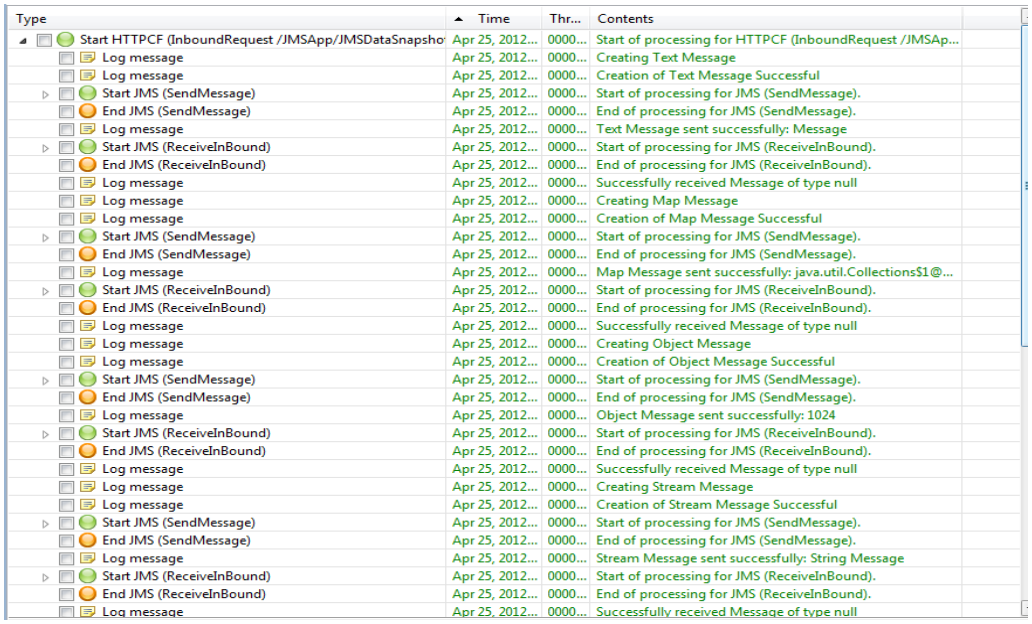


In this case, two servers are involved.

The SystemOut.log from server1 and server2 are loaded to show the message flow from server1 to server2 and vice versa

The source and target messaging engines UUID's are captured by XCT

## Message send and received with DataSnapShot enabled



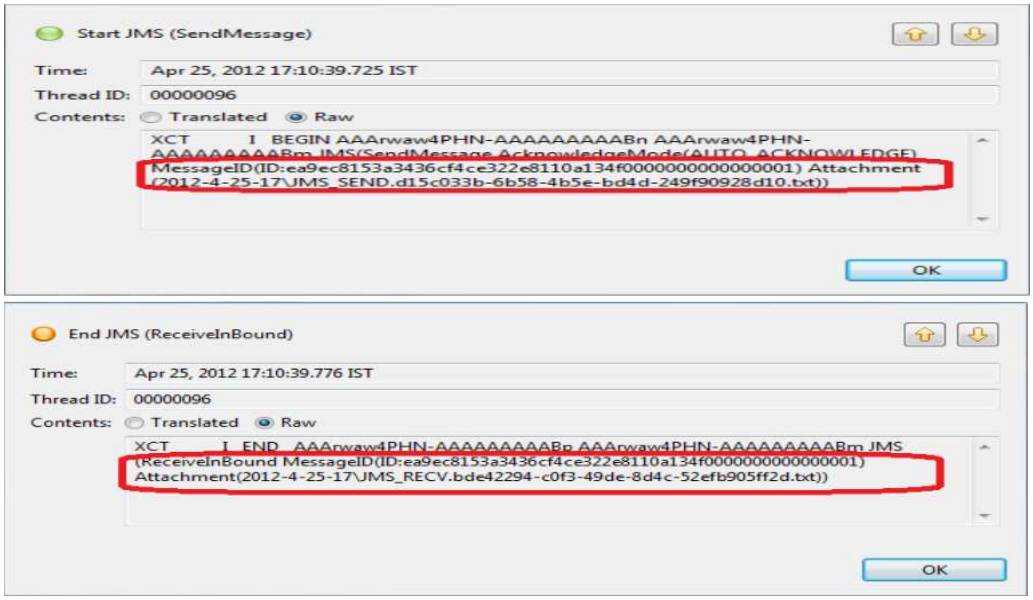
| Type | Time | Thr... | Contents |
|---|---|---|---|
| Start HTTPCF (InboundRequest /JMSApp/JMSDataSnapsho | Apr 25, 2012... | 0000... | Start of processing for HTTPCF (InboundRequest /JMSAp... |
| Log message | Apr 25, 2012... | 0000... | Creating Text Message |
| Log message | Apr 25, 2012... | 0000... | Creation of Text Message Successful |
| Start JMS (SendMessage) | Apr 25, 2012... | 0000... | Start of processing for JMS (SendMessage). |
| End JMS (SendMessage) | Apr 25, 2012... | 0000... | End of processing for JMS (SendMessage). |
| Log message | Apr 25, 2012... | 0000... | Text Message sent successfully: Message |
| Start JMS (ReceiveInBound) | Apr 25, 2012... | 0000... | Start of processing for JMS (ReceiveInBound). |
| End JMS (ReceiveInBound) | Apr 25, 2012... | 0000... | End of processing for JMS (ReceiveInBound). |
| Log message | Apr 25, 2012... | 0000... | Successfully received Message of type null |
| Log message | Apr 25, 2012... | 0000... | Creating Map Message |
| Log message | Apr 25, 2012... | 0000... | Creation of Map Message Successful |
| Start JMS (SendMessage) | Apr 25, 2012... | 0000... | Start of processing for JMS (SendMessage). |
| End JMS (SendMessage) | Apr 25, 2012... | 0000... | End of processing for JMS (SendMessage). |
| Log message | Apr 25, 2012... | 0000... | Map Message sent successfully: java.util.Collections$1@... |
| Start JMS (ReceiveInBound) | Apr 25, 2012... | 0000... | Start of processing for JMS (ReceiveInBound). |
| End JMS (ReceiveInBound) | Apr 25, 2012... | 0000... | End of processing for JMS (ReceiveInBound). |
| Log message | Apr 25, 2012... | 0000... | Successfully received Message of type null |
| Log message | Apr 25, 2012... | 0000... | Creating Object Message |
| Log message | Apr 25, 2012... | 0000... | Creation of Object Message Successful |
| Start JMS (SendMessage) | Apr 25, 2012... | 0000... | Start of processing for JMS (SendMessage). |
| End JMS (SendMessage) | Apr 25, 2012... | 0000... | End of processing for JMS (SendMessage). |
| Log message | Apr 25, 2012... | 0000... | Object Message sent successfully: 1024 |
| Start JMS (ReceiveInBound) | Apr 25, 2012... | 0000... | Start of processing for JMS (ReceiveInBound). |
| End JMS (ReceiveInBound) | Apr 25, 2012... | 0000... | End of processing for JMS (ReceiveInBound). |
| Log message | Apr 25, 2012... | 0000... | Successfully received Message of type null |
| Log message | Apr 25, 2012... | 0000... | Creating Stream Message |
| Log message | Apr 25, 2012... | 0000... | Creation of Stream Message Successful |
| Start JMS (SendMessage) | Apr 25, 2012... | 0000... | Start of processing for JMS (SendMessage). |
| End JMS (SendMessage) | Apr 25, 2012... | 0000... | End of processing for JMS (SendMessage). |
| Log message | Apr 25, 2012... | 0000... | Stream Message sent successfully: String Message |
| Start JMS (ReceiveInBound) | Apr 25, 2012... | 0000... | Start of processing for JMS (ReceiveInBound). |
| End JMS (ReceiveInBound) | Apr 25, 2012... | 0000... | End of processing for JMS (ReceiveInBound). |
| Log message | Apr 25, 2012... | 0000... | Successfully received Message of type null |




In this scenario, a message is sent to and received from a local queue destination with the XCT Data Snapshot option enabled.

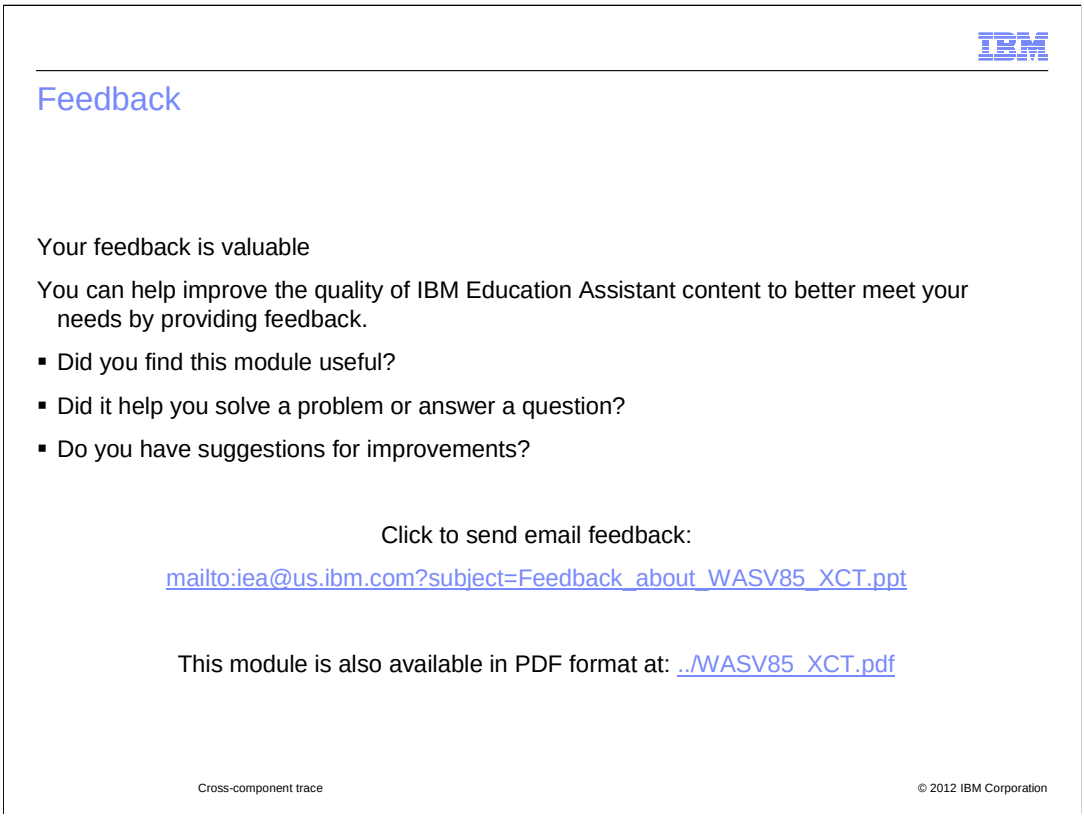
When the message is sent and received the message data is stored in a file under the snapdata directory which is typically found under the server log root.

## XCT records




26 Cross-component trace 

The XCT log record has the information on the attachment created




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

## Trademarks, disclaimer, and copyright information