IBM WebSphere Application Server V8.5 lab
Cross-component trace

Scenario

You are a system administrator responsible for managing web application server installations who needs to learn how to configure and use the new cross component trace (XCT) feature available in IBM WebSphere Application Server V8.5.

To better cover cases where a request is serviced by more than one thread, process, or even server, XCT annotates the logs so that related entries can be identified as belonging to the same unit of work.

XCT helps identify the root causes of problems across components, which provides these benefits:

- Enables administrators and support teams to follow the flow of a request from end-to-end as it traverses thread or process boundaries, or travels between stack products and WebSphere Application Server.
- Helps to resolve questions about which component is responsible for a request that fails.

For more information, see the WebSphere Application Server V8.5 information center topic: Cross Component Trace (XCT)

Goals

During this lab, you will learn to:

1. Enable and configure XCT using the WebSphere administrative console.

2. Enable and configure XCT using the wsadmin command line interface.

3. Use the WebSphere Cross Component Trace Logviewer to analyze various JMS and HTTP scenarios.

Prerequisites

Scenarios 1-4

The lab materials file, `WASv85Labs_XCT.zip`, includes sample log files for scenarios 1-4 (JMS); as a result, no WebSphere Application Server installation is required. To download this file, visit the WebSphere Application Server V8.5 area of the IBM Education Assistant site.

In order to view the sample log files, you will need the IBM Support Assistant (ISA) Workbench with the IBM WebSphere Cross Component Trace Logviewer add-on. Basic installation instructions are included in the Getting Started section on page 5 of this document.

Scenario 5

In order to complete scenario 5 (HTTP), you will need a WebSphere Application Server installation. The lab instructions and examples assume that you are using a deployment manager cell environment with a local application server node that includes one server (server1); however, with relatively minor adjustments, you can use a stand-alone application server instead. Note that in either case, the application server must include the Default Application (DefaultApplication). The setup of the assumed cell environment is outlined below.

- WebSphere Application Server V8.5
  - Application server root
    - Windows: C:\Program Files\IBM\WebSphere\AppServer
    - UNIX or Linux: /opt/IBM/WebSphere/AppServer
  - Deployment Manager Node
    - Profile name: Dmgr01
    - Profile path
      - Windows: C:\Program Files\IBM\WebSphere\AppServer\profiles\Dmgr01
      - UNIX or Linux: /opt/IBM/WebSphere/AppServer/profiles/Dmgr01
    - Cell name: DmgrCell01
    - Node name: DmgrCellManager01
    - Server name: dmgr
    - Administrative Account
      - User name: was
      - Password: was
  - Federated Application Server Node 1
    - Profile name: AppSrv01
    - Profile path
      - Windows: C:\Program Files\IBM\WebSphere\AppServer\profiles\AppSrv01
      - UNIX or Linux: /opt/IBM/WebSphere/AppServer/profiles/AppSrv01
    - Node name: AppSrv01Node
    - Server name: server1
      - Installed applications
        - DefaultApplication

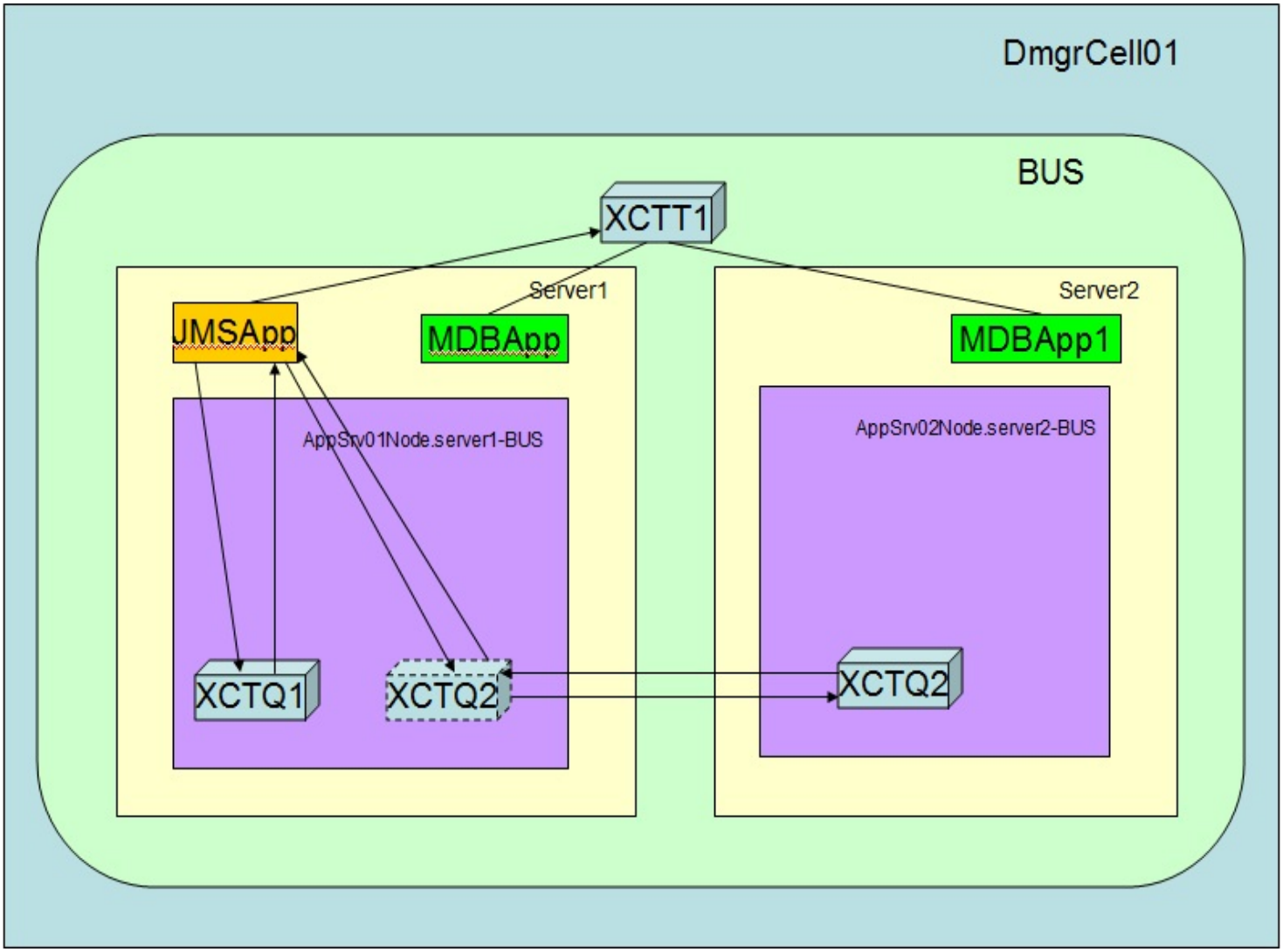
Procedure

JMS XCT instrumentation analysis

Overview

In this section of the lab, you will use the WebSphere Cross Component Trace Logviewer to analyze sample log files that track JMS messaging activity in a basic cell environment. The messaging topology is illustrated below. An outline of the assumed application server environment follows.

Messaging topology



Application server environment

- WebSphere Application Server V8.5
  - Application server root
    - Windows: C:\Program Files\IBM\WebSphere\AppServer
    - UNIX or Linux: /opt/IBM/WebSphere/AppServer
  - Deployment Manager Node
    - Profile name: Dmgr01
    - Profile path
      - Windows: C:\Program Files\IBM\WebSphere\AppServer\profiles\Dmgr01
      - UNIX or Linux: /opt/IBM/WebSphere/AppServer/profiles/Dmgr01
    - Cell name: DmgrCell01
    - Node name: DmgrCellManager01
    - Server name: dmgr
    - Administrative Account
      - User name: was
      - Password: was
  - Federated Application Server Node 1
    - Profile name: AppSrv01
    - Profile path
      - Windows: C:\Program Files\IBM\WebSphere\AppServer\profiles\AppSrv01
      - UNIX or Linux: /opt/IBM/WebSphere/AppServer/profiles/AppSrv01
    - Node name: AppSrv01Node
    - Server name: server1
  - Federated Application Server Node 2
    - Profile name: AppSrv02
    - Profile path
      - Windows: C:\Program Files\IBM\WebSphere\AppServer\profiles\AppSrv02
      - UNIX or Linux: /opt/IBM/WebSphere/AppServer/profiles/AppSrv02
    - Node name: AppSrv02Node
    - Server name: server2

Getting started

1. Install the WebSphere Cross Component Trace Logviewer.

If the IBM Support Assistant (ISA) Workbench is not present on your system, do these steps:

a. Download the appropriate ISA Workbench installation archive from http://www.ibm.com/software/support/isa

b. Extract the installation files into a suitable directory.

c. Launch the installer.

d. Use the installation wizard to install the ISA Workbench and the tools add-on IBM WebSphere Cross Component Trace Logviewer (at a minimum).

If the IBM Support Assistant (ISA) Workbench is already present on your system, do these steps:

a. Start the ISA Workbench in the operating system shell.

b. Click **Update > Find New > Tools Add-ons**.

c. Within the Find new tools add-ons wizard, select the tool **JVM-based Tools > IBM WebSphere Cross Component Trace Logviewer**. Then follow the prompts to complete the installation.

d. When prompted, restart the ISA Workbench.

For more information, see the Using IBM Support Assistant topic in the WebSphere Application Server V8.5 information center.

2. Extract the lab materials file.

Extract lab materials file `WASv85Labs_XCT.zip` into a suitable directory. You can extract this file, and any other WebSphere Application Server V8.5 lab materials files, into the root directory. For example, on Windows, extracting the file into the `C:\` directory will create lab materials directory `C:\WASv85Labs\XCT`, and so on. You can also use the same basic approach on UNIX or Linux.

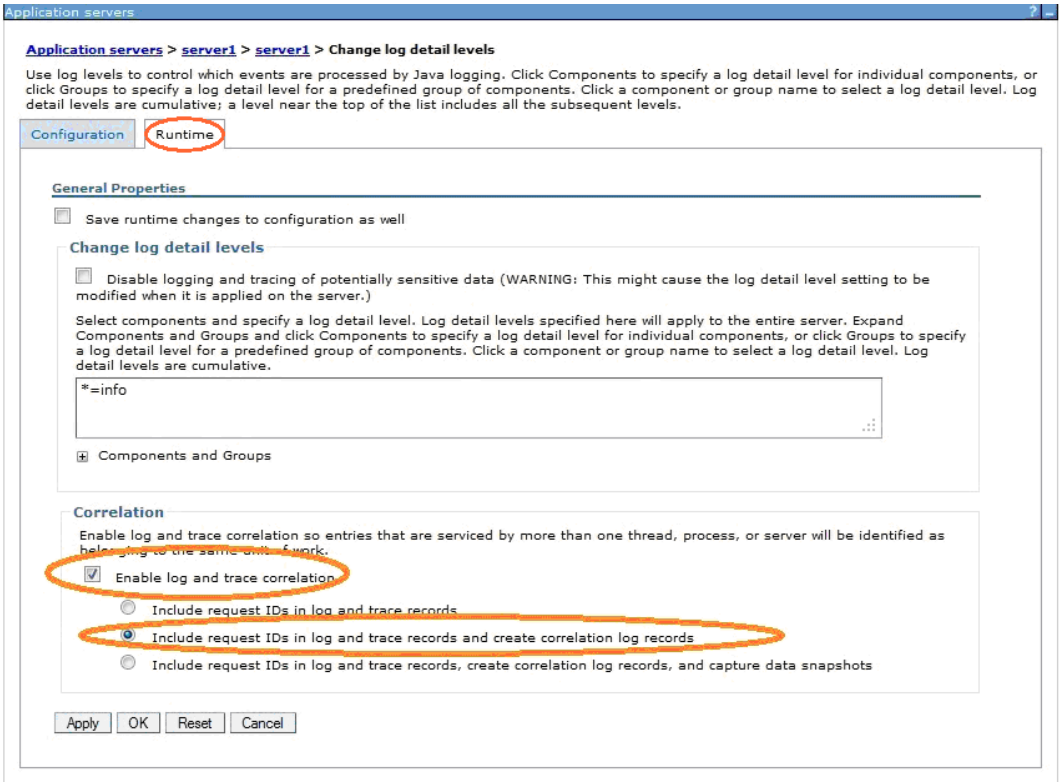
Scenario 1 (sending and receiving a message from a local server) [synchronous receive]

IBM tasks - Generating sample log file

To generate the sample `SystemOut.log` file for this scenario, this is what was done:

1. Used the startServer command to start the deployment manager, node agents, and server1.

2. Used the deployment manager administrative console to enable XCT for server1.
   a. Displayed the following console screen: *Servers* > *Server Types* > *WebSphere Application Servers*.
   b. Clicked **server1**.
   c. Clicked **Change log detail levels**.
   d. Clicked the **Runtime** tab.
   e. Selected:
      -Enable log and trace correlation
      -Include request IDs in log and trace records and create correlation log records



   f. Verified that changes were made with the Runtime tab selected and clicked **OK** to save the changes.

3. Instructed the JMSApp servlet (on server1) to send a message.

4. Instructed the JMSApp servlet (on server1) to receive the message that was sent in the previous step.

Customer tasks

To analyze the cross component logging information, do the following:

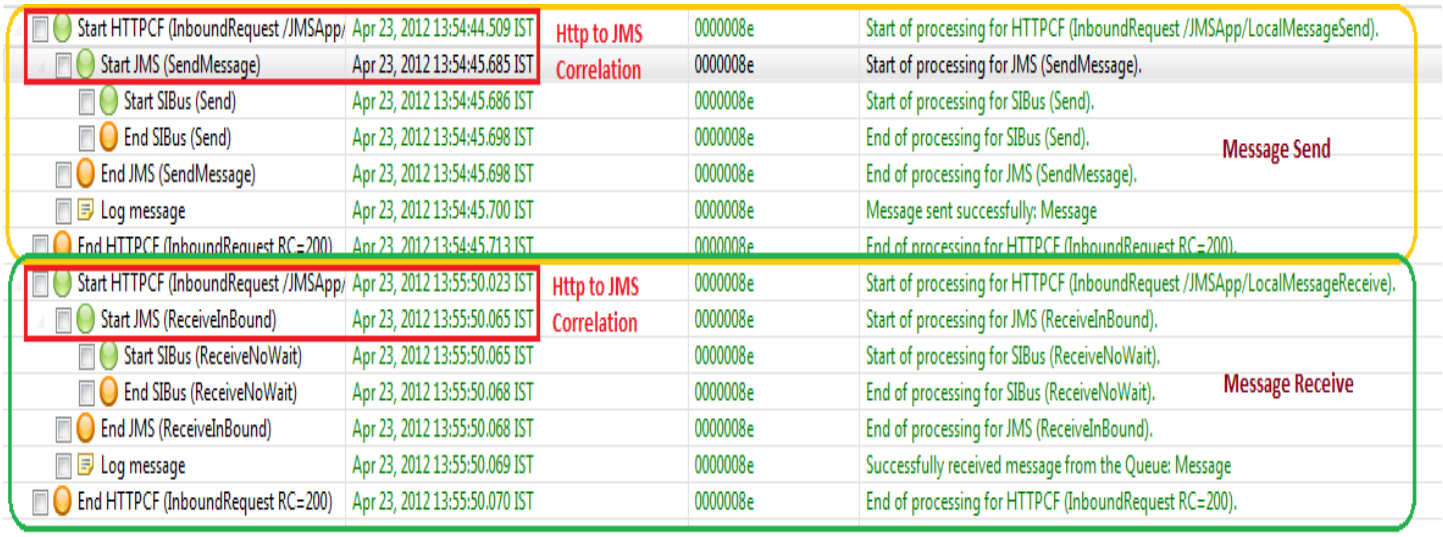
1. Use the Cross Component Trace Logviewer to display the `SystemOut.log` file for server1.

    a. Start the Cross Component Trace Logviewer.

    First start the ISA Workbench. Next, click **Analyze Problem**. Then select the **Tools** tab. Finally, select **IBM WebSphere Cross Component Trace Logviewer** and click **Launch**.

    b. Within the log viewer button bar, click the **Load Server Console or Log** button (icon). Then load the `server1_SystemOut.log` file that is located in lab directory `/WASv85Labs/XCT/Scenario1`

2. Within the log viewer, scroll to the end of the record list. Then fully expand the two Start HTTPCF messages and the two End HTTPCF messages. You will see this information:



XCT Records from SystemOut.log

```
5/7/12 22:07:48:813 IST] 000000ae XCT           I   BEGIN AAAECSc7MA9-AAAAAAAAAAA 00000000000-
ccccccccc2 HTTPCF(InboundRequest /JMSApp/LocalMessageSend RemoteAddress(127.0.0.1) RequestCon-
text(-2096270047))

[5/7/12 22:07:48:837 IST] 000000ae servlet       I com.ibm.ws.webcontainer.servlet.ServletWrapper
init SRVE0242I: [JMSApp] [/JMSApp] [LocalMessageSend]: Initialization successful.

[5/7/12 22:07:49:647 IST] 000000ae XCT           I   BEGIN AAAECSc7MA9-AAAAAAAAAAB AAAECSc7MA9-
AAAAAAAAAAA JMS(SendMessage AcknowledgeMode(AUTO_ACKNOWLEDGE) Mes-
sageID(ID:56c13f47a20f595d45598a8d110a134f0000000000000001))

[5/7/12 22:07:49:648 IST] 000000ae XCT           I   BEGIN AAAECSc7MA9-AAAAAAAAAAC AAAECSc7MA9-
AAAAAAAAAAB SIBus(Send Assoc(MessagingEngineUuid F17867761B42C374) Assoc(DestinationName XCTQ1)
DestinationType(Queue) Transacted(False) Reliability(ReliablePersistent))

[5/7/12 22:07:49:674 IST] 000000ae XCT           I   END   AAAECSc7MA9-AAAAAAAAAAC AAAECSc7MA9-
AAAAAAAAAAB SIBus(Send SystemMessageID(F17867761B42C374_9000005))

[5/7/12 22:07:49:674 IST] 000000ae XCT           I   END   AAAECSc7MA9-AAAAAAAAAAB AAAECSc7MA9-
AAAAAAAAAAA JMS(SendMessage)
```

```
[5/7/12 22:07:49:674 IST] 000000ae SystemOut     O Message sent successfully: Message

[5/7/12 22:07:49:684 IST] 000000ae XCT           I   END   AAAECSc7MA9-AAAAAAAAAAA 00000000000-
ccccccccc2 HTTPCF(InboundRequest RC=200 RequestContext(-2096270047))

[5/7/12 22:07:49:716 IST] 000000ae XCT           I   BEGIN AAAECSc7MA9-AAAAAAAAAAD 00000000000-
ccccccccc2 HTTPCF(InboundRequest /favicon.ico RemoteAddress(127.0.0.1) RequestContext(-
2096270047))

[5/7/12 22:07:49:719 IST] 000000ae XCT           I   END   AAAECSc7MA9-AAAAAAAAAAD 00000000000-
ccccccccc2 HTTPCF(InboundRequest RC=404 RequestContext(-2096270047))

[5/7/12 22:07:49:724 IST] 000000ae XCT           I   BEGIN AAAECSc7MA9-AAAAAAAAAAE 00000000000-
ccccccccc2 HTTPCF(InboundRequest /favicon.ico RemoteAddress(127.0.0.1) RequestContext(-272089357))

[5/7/12 22:07:49:726 IST] 000000ae XCT           I   END   AAAECSc7MA9-AAAAAAAAAAE 00000000000-
ccccccccc2 HTTPCF(InboundRequest RC=404 RequestContext(-272089357))

[5/7/12 22:07:49:729 IST] 000000ae XCT           I   BEGIN AAAECSc7MA9-AAAAAAAAAAF 00000000000-
ccccccccc2 HTTPCF(InboundRequest /favicon.ico RemoteAddress(127.0.0.1) RequestContext(1898687204))

[5/7/12 22:07:49:730 IST] 000000ae XCT           I   END   AAAECSc7MA9-AAAAAAAAAAF 00000000000-
ccccccccc2 HTTPCF(InboundRequest RC=404 RequestContext(1898687204))

[5/7/12 22:07:55:713 IST] 000000ae XCT           I   BEGIN AAAECSc7MA9-AAAAAAAAAAG 00000000000-
ccccccccc2 HTTPCF(InboundRequest /JMSApp/LocalMessageReceive RemoteAddress(127.0.0.1) RequestCon-
text(-431652354))

[5/7/12 22:07:55:718 IST] 000000ae servlet       I com.ibm.ws.webcontainer.servlet.ServletWrapper
init SRVE0242I: [JMSApp] [/JMSApp] [LocalMessageReceive]: Initialization successful.

[5/7/12 22:07:55:759 IST] 000000ae XCT           I   BEGIN AAAECSc7MA9-AAAAAAAAAAH AAAECSc7MA9-
AAAAAAAAAAG JMS(ReceiveInBound DestinationType(Queue))

[5/7/12 22:07:55:760 IST] 000000ae XCT           I   BEGIN AAAECSc7MA9-AAAAAAAAAAI AAAECSc7MA9-
AAAAAAAAAAH SIBus(ReceiveNoWait Assoc(MessagingEngineUuid F17867761B42C374) Assoc(DestinationName
XCTQ1))

[5/7/12 22:07:55:768 IST] 000000ae XCT           I   END   AAAECSc7MA9-AAAAAAAAAAI AAAECSc7MA9-
AAAAAAAAAAH SIBus(ReceiveNoWait Assoc(XctId AAAECSc7MA9-AAAAAAAAAAC) Assoc(XctRootId AAAECSc7MA9-
AAAAAAAAAAA))

[5/7/12 22:07:55:769 IST] 000000ae XCT           I   END   AAAECSc7MA9-AAAAAAAAAAH AAAECSc7MA9-
AAAAAAAAAAG JMS(ReceiveInBound MessageID(ID:56c13f47a20f595d45598a8d110a134f0000000000000001))

[5/7/12 22:07:55:769 IST] 000000ae SystemOut     O Successfully received message from the Queue:
Message

[5/7/12 22:07:55:770 IST] 000000ae XCT           I   END   AAAECSc7MA9-AAAAAAAAAAG 00000000000-
ccccccccc2 HTTPCF(InboundRequest RC=200 RequestContext(-431652354))
```

XCT Record Format

Each XCT record is structured as shown below.

<Date> <Thread_ID> <XCT_Logger_Name> <Message_Type> <XCT_STATE> <XCT_ID>

<XCT_PARENT_ID> <XCT_MESSAGE>

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))*

<Date>: The date and time when the log was generated.

<Thread_ID>: The thread which generated the message.

<XCT_Logger_Name>: The XCT logger name is **XCT.** It is used to identify the XCT records in the log file.

<Message_Type>: Type of the log message such as I-Information, E-Error, and so on.

<XCT_STATE>: Each XCT record has a state such as **BEGIN** or **END**.

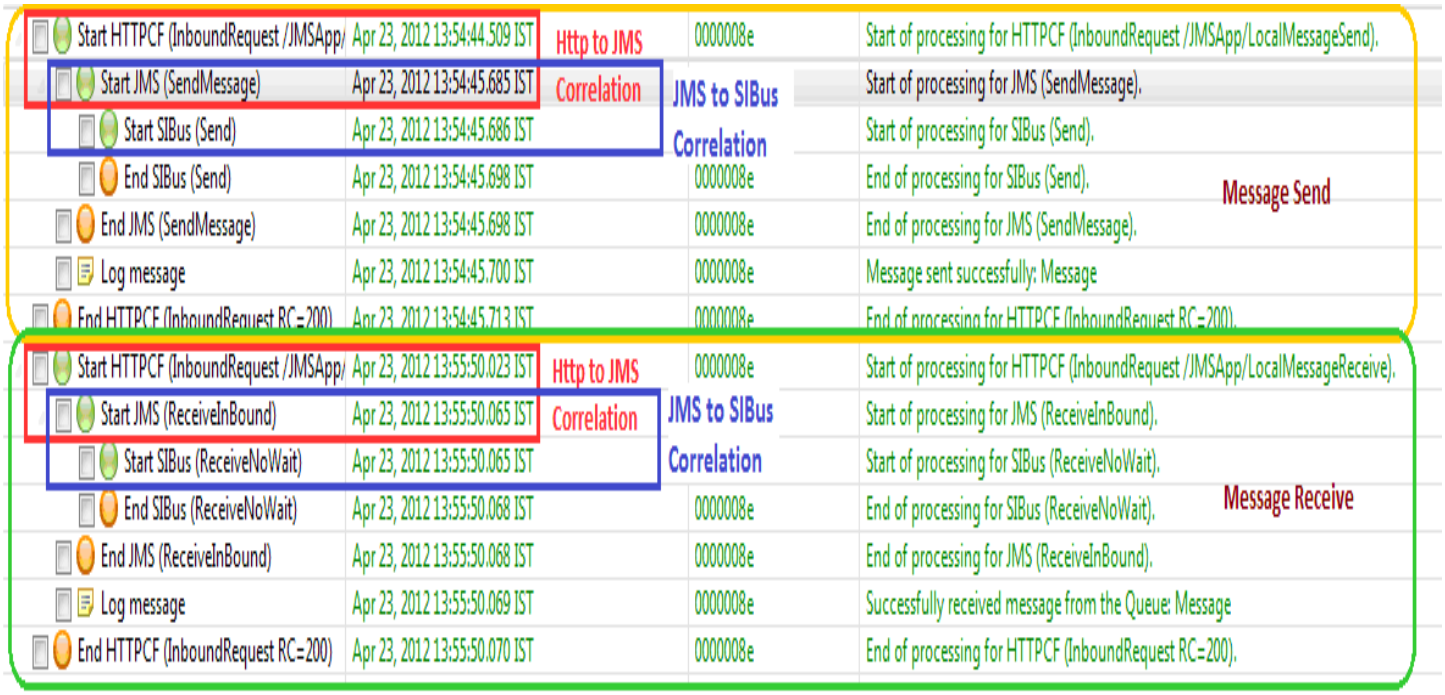
<XCT_ID>: A Unique ID generated for correlating the XCT records.

<XCT_PARENT_ID>: XCT_ID of the parent record.

<XCT_MESSAGE>: The XCT message contains the information about the XCT record; it may contain some associations and annotations.

XCT Records in the Logviewer

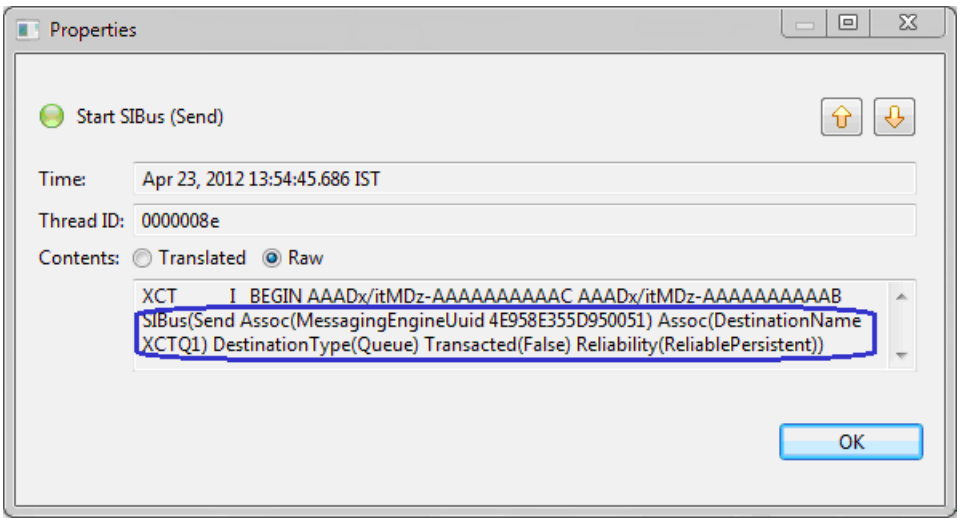
When you go through the logs in the Cross Component Trace Logviewer, a set of XCT records are generated and displayed in the form of tree structure based on the flow. You can see the section marked in **yellow** is for the **Message Send** part and the section marked in **Green** is for the **Message Receive**. The highlighted part in **red** and **blue** indicates the ***unit of work transferred from HTTP layer to JMS layer and JMS layer to SIBus layer respectively***.



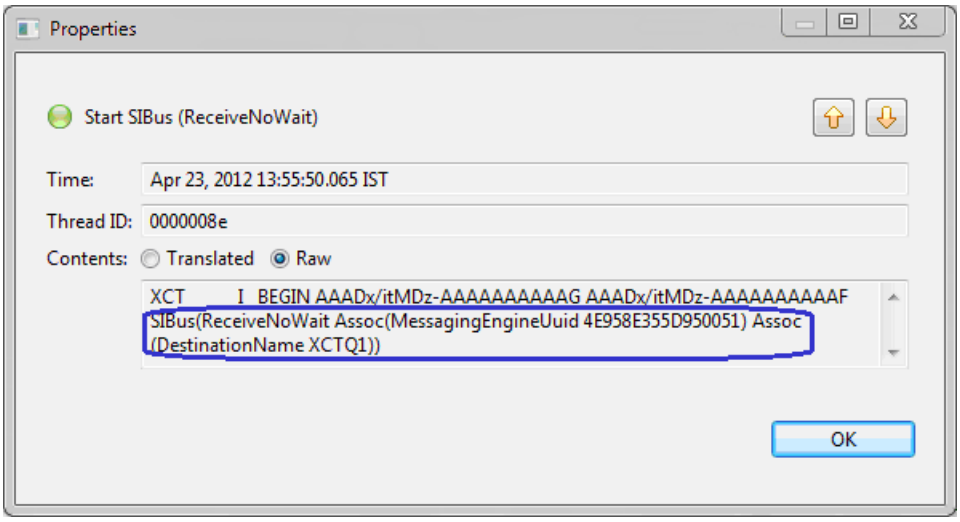
Key XCT Record Types

To view the annotations and the description of the XCT record, double-click the entry in the record list. Key XCT records types are described below.

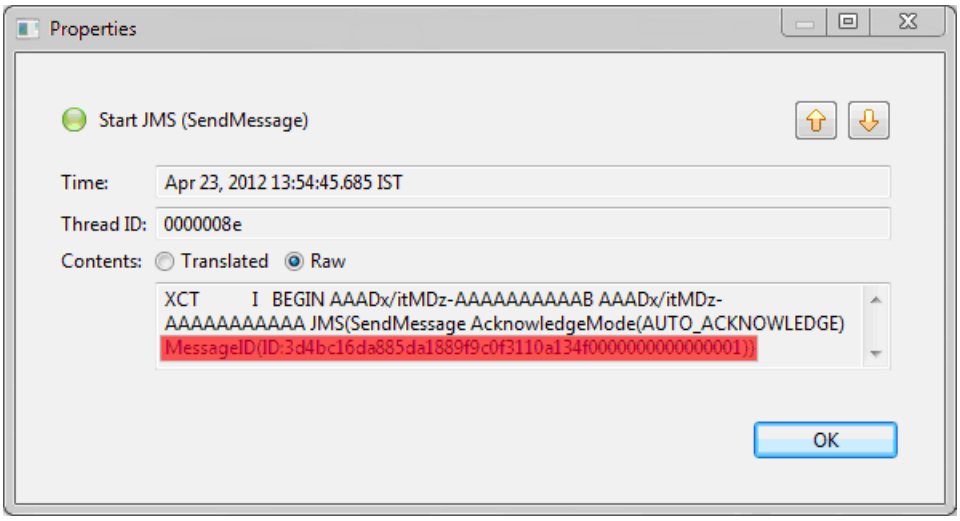
a. The Start SIBus Send record is included below. It includes information about the DestinationName, MessagingEngineUuid, DestinationType, Transaction Type, and Reliability.

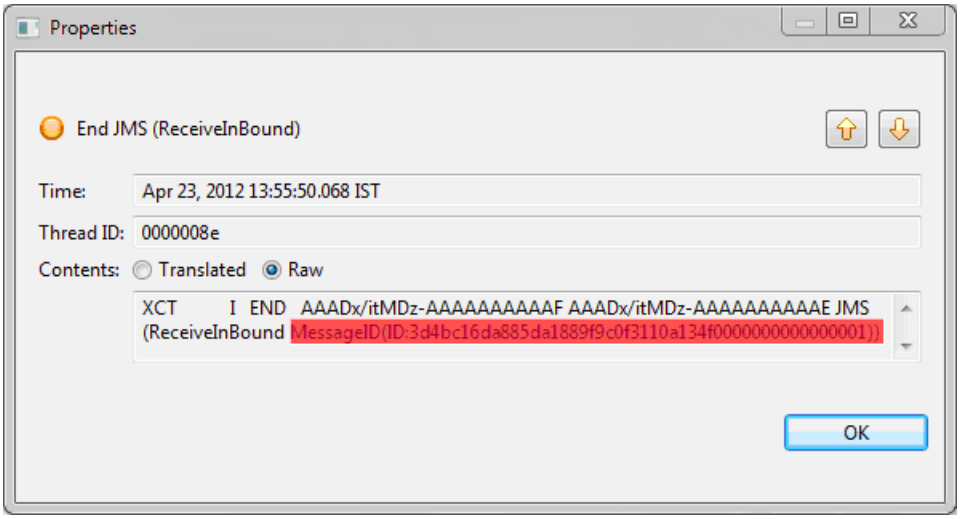


b. The Start SIBus ReceiveNoWait record is included below. It includes information about the MessagingEngineUuid and the DestinationName.



c. The records below illustrate how you can use the MessageID to correlate the JMS Start of the send side and JMS End of the receive side.





3. Close the `SystemOut.log` file.

IBM Tasks - Wrap-up

To prepare for the next scenario, this is what has been done:

1. Used the deployment manager administrative console to stop server1.
    a. Displayed the following console screen: *Servers > Server Types > WebSphere Application Servers*.
    b. Selected (checked) **server1**.
    c. Clicked **Stop** and responded to the confirmation or message prompts as appropriate.

2. Deleted the `SystemOut.log` file for server1, for example,

Windows
```
C:\Program Files\IBM\WebSphere\AppServer\profiles\AppSrv01\logs\server1\SystemOut.log
```

UNIX or Linux
```
/opt/IBM/WebSphere/AppServer/profiles/AppSrv01/logs/server1/SystemOut.log
```

Scenario 2: Asynchronous Message Receiving (PubSub - Topic)

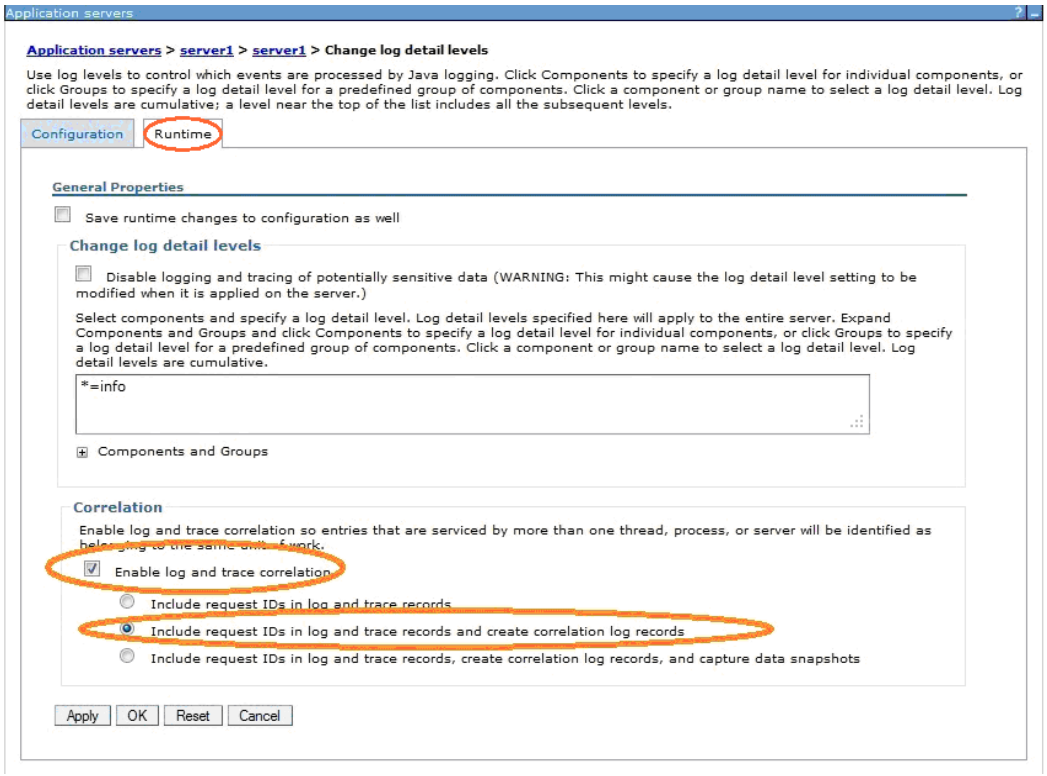
IBM Tasks - Generating Sample Log Files

To generate the sample log files for this scenario, this is what has been done:

1. Used the deployment manager administrative console to start both servers and enable XCT on both servers.

    a. Started server1 and server2.
        i. Displayed the following screen: *Servers > Server Types > WebSphere Application Servers*.
        ii. Selected (checked) **server1** and **server2**.
        iii. Clicked **Start** and responded to the message prompts as appropriate.

    b. Enabled XCT for server1.
        i. Clicked **server1**.
        ii. Clicked **Change log detail levels**.
        iii. Clicked the **Runtime** tab.
        iv. Selected the following (as illustrated below):
            -Enable log and trace correlation
            -Include request IDs in log and trace records and create correlation log records



        v. Verified that the changes were made with the Runtime tab selected and clicked **OK**.

    c. Repeated step b for server2.

2. Instructed the JMSApp servlet (on server1) to publish a message to a topic which was subscribed by the Message Driven Bean.

Customer Tasks

To analyze the cross component logging information, do the following:

1. Use the Cross Component Trace Logviewer to display the `SystemOut.log` file for server1.

    a. If necessary, start the Cross Component Trace Logviewer.

    b. Within the log viewer button bar, click the **Load Server Console or Log** button (icon). Then load the `server1_SystemOut.log` file that is located in lab directory `/WASv85Labs/XCT/Scenario2`

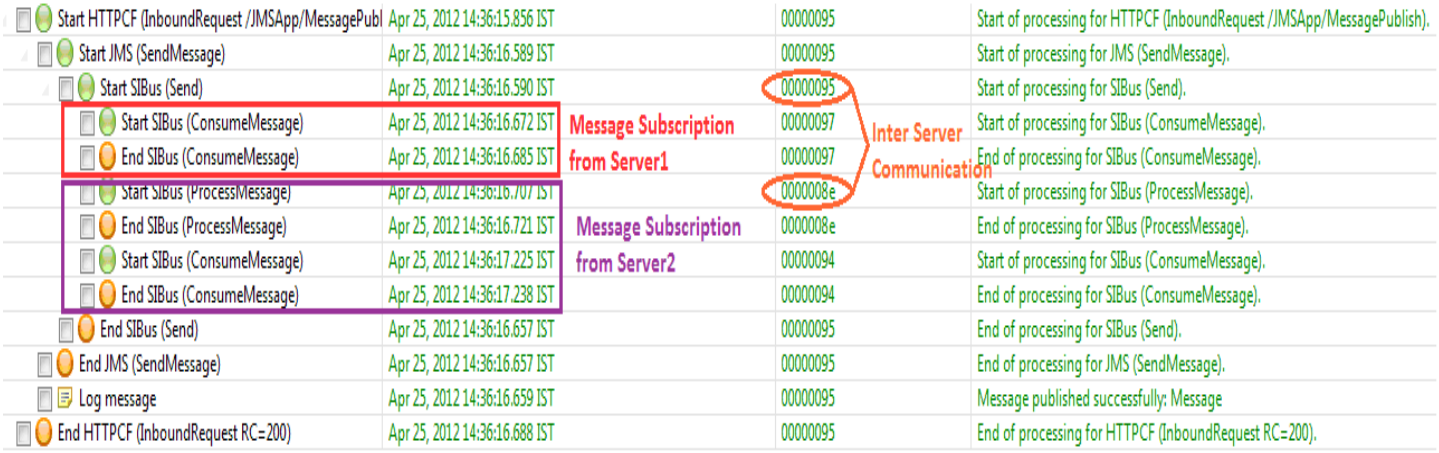
2. Within the log viewer, scroll to the end of the record list. Then fully expand the Start HTTPCF message. You will see the XCT flow on server1 as illustrated below. The blue oval indicates the correlation between various threads in the process/server.

XCT log records for server1

3. Within the log viewer button bar, click the **Load Server Console or Log** button (icon). Then load the `server1_SystemOut.log` and `server2_SystemOut.log` files that are located in lab directory `/WASv85Labs/XCT/Scenario2`

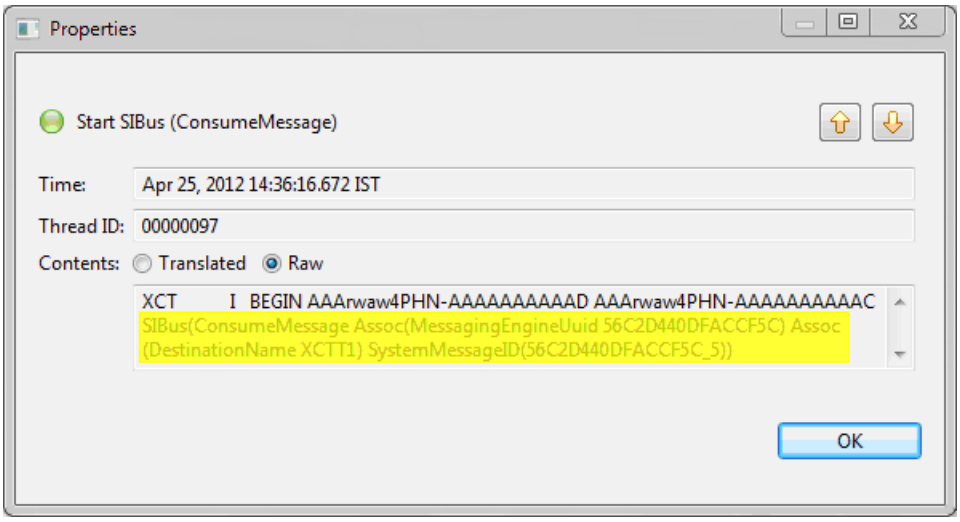
4. Within the log viewer, scroll to the end of the record list. Then fully expand the Start HTTPCF message. You will see the XCT flow with server logs merged as illustrated below. The orange ovals indicate the correlation between different processes/servers.

Merged XCT log records for server1 and server2
(MDB is running on both servers, so message gets subscribed on both servers)



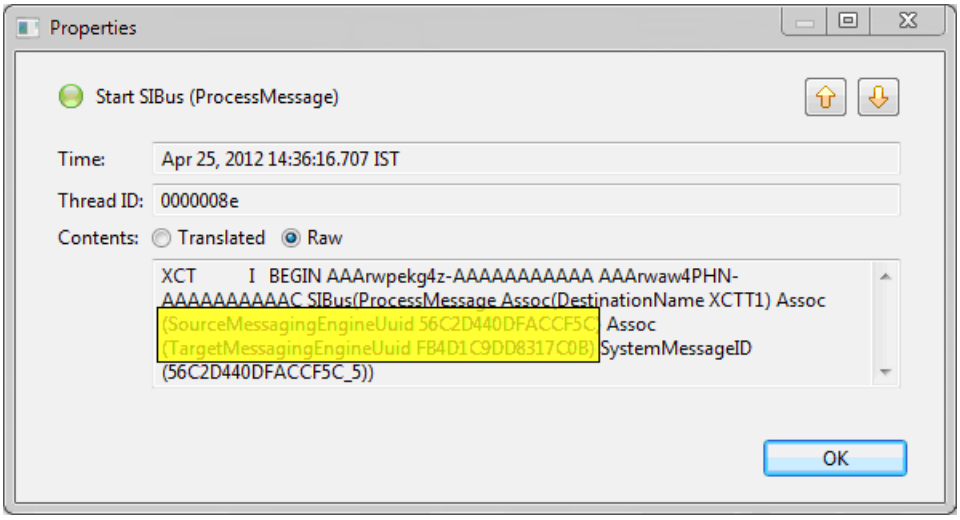
Key XCT Records

a. The Start SIBus ConsumeMessage XCT record is added at the receiver side in the case of asynchronous message consumption.



b. The Start SIBus ProcessMessage XCT record is added as a part of inter-process/server communication, i.e., when the message is sent from one messaging engine to another messaging engine. It contains the following information:

- Messaging engine from which the message is sent (SourceMessagingEngineUuid)
- Messaging engine where the destination is available (TargetMessagingEngineUuid)



5. Close all `SystemOut.log` files.

IBM Tasks - Wrap-up

To prepare for the next scenario, this is what has been done:

1. Used the deployment manager administrative console to stop server1 and server2.
    a. Displayed the following console screen: *Servers > Server Types > WebSphere Application Servers*.
    b. Selected (checked) **server1** and **server2**.
    c. Clicked **Stop** and responded to the confirmation or message prompts as appropriate.

2. Deleted the `SystemOut.log` files for server1 and server2, for example,

Windows
```
C:\Program Files\IBM\WebSphere\AppServer\profiles\AppSrv01\logs\server1\SystemOut.log
C:\Program Files\IBM\WebSphere\AppServer\profiles\AppSrv02\logs\server2\SystemOut.log
```

UNIX or Linux
```
/opt/IBM/WebSphere/AppServer/profiles/AppSrv01/logs/server1/SystemOut.log
/opt/IBM/WebSphere/AppServer/profiles/AppSrv02/logs/server2/SystemOut.log
```

Scenario 3: Store and Forward and Remote Get (Sending and receiving a message from a destination existing on a different server)

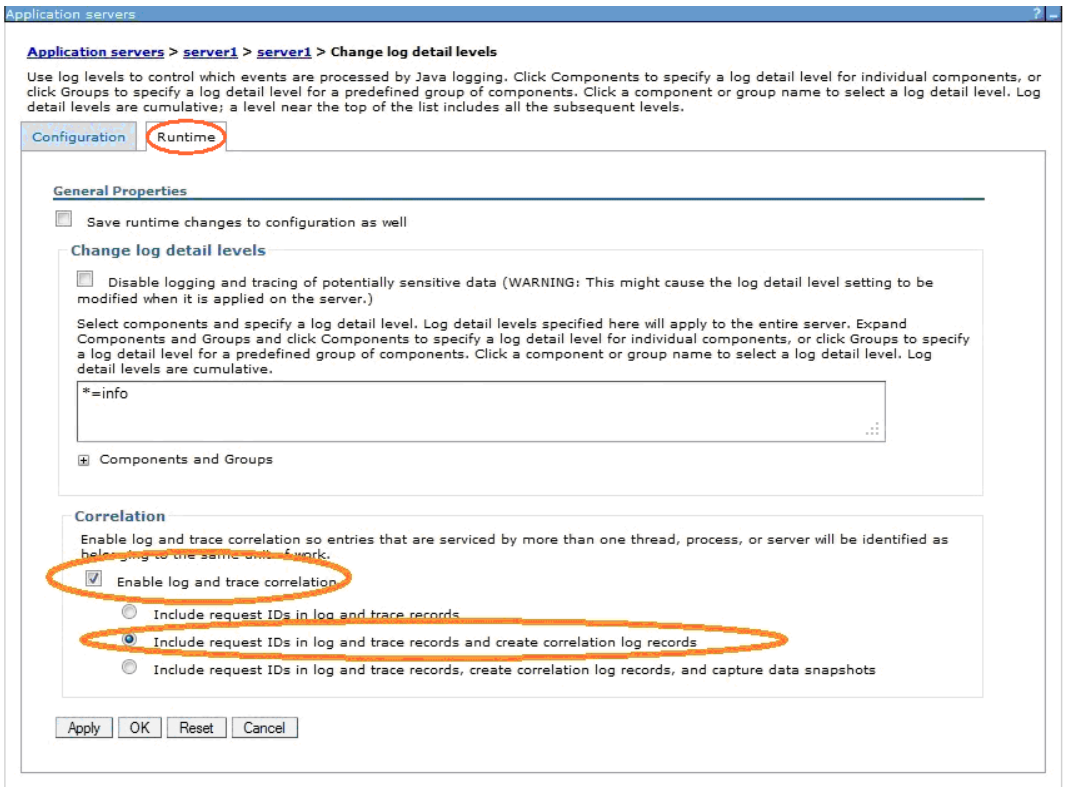
IBM Tasks - Generating Sample Log Files

To generate the sample log files for this scenario, this is what was done:

1. Used the deployment manager administrative console to start both servers and enable XCT on both servers.

    a. Started server1 and server2.
        i. Displayed the following screen: *Servers > Server Types > WebSphere Application Servers*.
        ii. Selected (checked) **server1** and **server2**.
        iii. Clicked **Start** and responded to the message prompts as appropriate.

    b. Enabled XCT for server1.
        i. Clicked **server1**.
        ii. Clicked **Change log detail levels**.
        iii. Clicked the **Runtime** tab.
        iv. Selected the following (as illustrated below):
            -Enable log and trace correlation
            -Include request IDs in log and trace records and create correlation log records



        v. Verified that the changes were made with the Runtime tab selected and clicked **OK**.

    c. Repeated step b for server2.

2. Instructed the JMSApp servlet (on server1) to send a message to the queue that exists on the remote server (server2).

3. Instructed the JMSApp servlet (on server1) to receive the message from the queue which exists on the remote server (server2).

Customer Tasks

To analyze the cross component logging information, do the following:

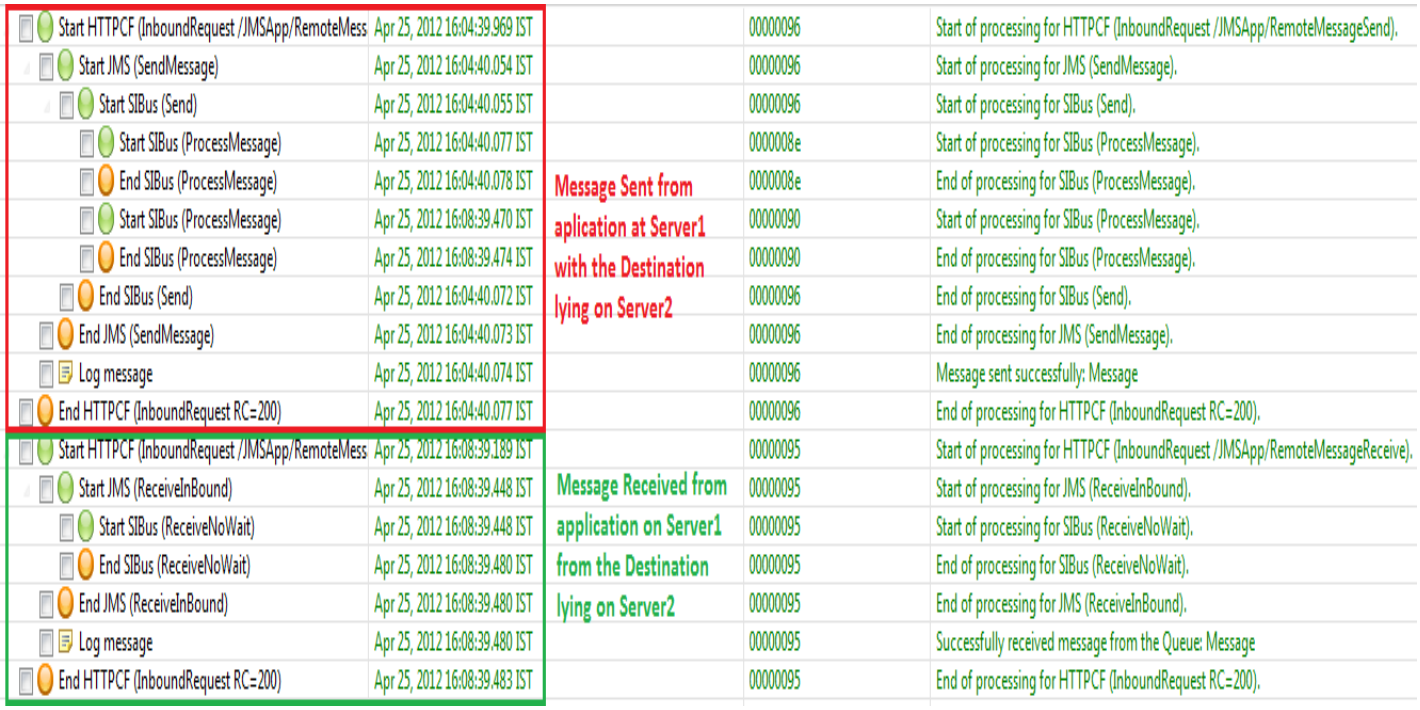
1. Use the Cross Component Trace Logviewer to display the `SystemOut.log` files for server1 and server2.

    a. If necessary, start the Cross Component Trace Logviewer.

    b. Within the log viewer button bar, click the **Load Server Console or Log** button (icon). Then load the `server1_SystemOut.log` and `server2_SystemOut.log` files that are located in lab directory `/WASv85Labs/XCT/Scenario3`

2. Within the log viewer, scroll to the end of the record list. Then fully expand both Start HTTPCF messages.

The graphic illustrates the XCT flow for the send and receive phases. This scenario involves two servers, with the application existing on one server (server1), and the destination existing on the other server (server2). The log viewer is displaying a merged view of the records in both server log files. This approach provides a complete picture of how the message moves across various protocols, threads, and servers.



For more information about the XCT records, see the previous two scenarios.

3. Close all `SystemOut.log` files.

IBM Tasks - Wrap-up

To prepare for the next scenario, this is what was done:

1. Used the deployment manager administrative console to stop server1 and server2.
    a. Displayed the following console screen: *Servers > Server Types > WebSphere Application Servers*.
    b. Selected (checked) **server1** and **server2**.
    c. Clicked **Stop** and responded to any confirmation prompts as appropriate.

2. Deleted the `SystemOut.log` file for server1, for example,

Windows
```
C:\Program Files\IBM\WebSphere\AppServer\profiles\AppSrv01\logs\server1\SystemOut.log
```

UNIX or Linux
```
/opt/IBM/WebSphere/AppServer/profiles/AppSrv01/logs/server1/SystemOut.log
```

Scenario 4: Local Message Send (Data Snapshot Feature)

IBM Tasks - Generating Sample Log Files

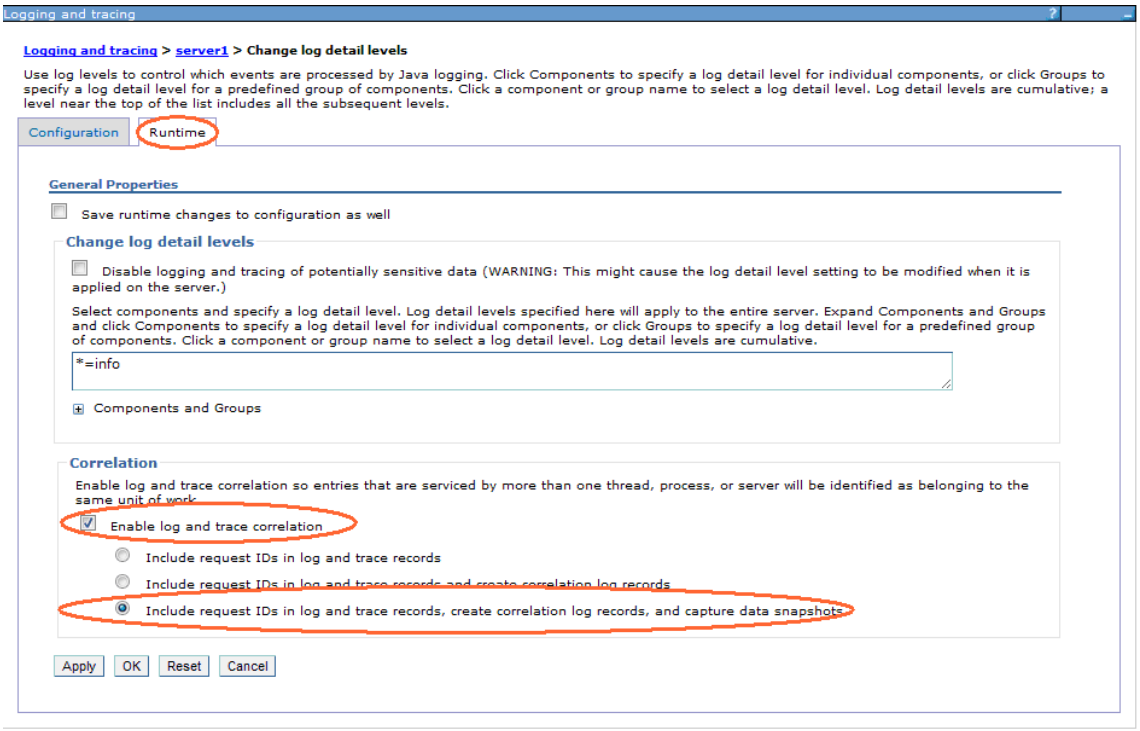
To generate the sample log files for this scenario, this is what was done:

1. Used the deployment manager administrative console to start server1 and enable XCT on server1.

    a. Started server1.
        i. Displayed the following screen: *Servers* > *Server Types* > *WebSphere Application Servers*.
        ii. Selected (checked) **server1**.
        iii. Clicked **Start** and responded to the message prompts as appropriate.

    b. Enabled XCT for server1.
        i. Clicked **server1**.
        ii. Clicked **Change log detail levels**.
        iii. Clicked the **Runtime** tab.
        iv. Selected **Enable log and trace correlation** and **Include request IDs in log and trace records, create correlation log records, and capture data snapshots** (as illustrated below).

        Note: The correlation settings are different for this scenario.



        v. Verified that the changes were made with the Runtime tab selected and clicked **OK**.

2. Instructed the JMSApp servlet (on server1) to send various types of messages to a queue existing on the same server.

Customer Tasks

To analyze the cross component logging information, do the following:

1. Use the Cross Component Trace Logviewer to display the `SystemOut.log` file for server1.

    a. If necessary, start the Cross Component Trace Logviewer.

    b. Within the log viewer button bar, click the **Load Server Console or Log** button (icon). Then load the `server1_SystemOut.log` file that is located in lab directory `/WASv85Labs/XCT/Scenario4`

2. Within the log viewer, scroll to the end of the record list. Then fully expand the Start HTTPCF message. You will see many different types of messages, as illustrated below.

Type	Time	Thread ID	Contents
Start HTTPCF (InboundRequest /JMSApp/JMSDataSnapshot)	May 20, 2012 13:23:07.511 EDT	000000c3	Start of processing for HTTPCF (InboundRequest /JMSApp/JMSDataSnapshot).
Log message	May 20, 2012 13:23:11.317 EDT	000000c3	Creating Text Message
Log message	May 20, 2012 13:23:11.337 EDT	000000c3	Creation of Text Message Successful
Start JMS (SendMessage)	May 20, 2012 13:23:11.737 EDT	000000c3	Start of processing for JMS (SendMessage).
Start SIBus (Send)	May 20, 2012 13:23:11.847 EDT	000000c3	Start of processing for SIBus (Send).
End SIBus (Send)	May 20, 2012 13:23:11.968 EDT	000000c3	End of processing for SIBus (Send).
End JMS (SendMessage)	May 20, 2012 13:23:11.968 EDT	000000c3	End of processing for JMS (SendMessage).
Log message	May 20, 2012 13:23:11.968 EDT	000000c3	Text Message sent successfully: Message
Start JMS (ReceiveInBound)	May 20, 2012 13:23:11.968 EDT	000000c3	Start of processing for JMS (ReceiveInBound).
Start SIBus (ReceiveNoWait)	May 20, 2012 13:23:11.978 EDT	000000c3	Start of processing for SIBus (ReceiveNoWait).
End SIBus (ReceiveNoWait)	May 20, 2012 13:23:11.978 EDT	000000c3	End of processing for SIBus (ReceiveNoWait).
End JMS (ReceiveInBound)	May 20, 2012 13:23:11.998 EDT	000000c3	End of processing for JMS (ReceiveInBound).
Log message	May 20, 2012 13:23:11.998 EDT	000000c3	Successfully received Message of type null
Log message	May 20, 2012 13:23:11.998 EDT	000000c3	Creating Map Message
Log message	May 20, 2012 13:23:12.098 EDT	000000c3	Creation of Map Message Successful
Start JMS (SendMessage)	May 20, 2012 13:23:12.108 EDT	000000c3	Start of processing for JMS (SendMessage).
Start SIBus (Send)	May 20, 2012 13:23:12.128 EDT	000000c3	Start of processing for SIBus (Send).
End SIBus (Send)	May 20, 2012 13:23:12.178 EDT	000000c3	End of processing for SIBus (Send).
End JMS (SendMessage)	May 20, 2012 13:23:12.178 EDT	000000c3	End of processing for JMS (SendMessage).
Log message	May 20, 2012 13:23:12.178 EDT	000000c3	Map Message sent successfully: java.util.Collections$1@fa69079d
Start JMS (ReceiveInBound)	May 20, 2012 13:23:12.178 EDT	000000c3	Start of processing for JMS (ReceiveInBound).
Start SIBus (ReceiveNoWait)	May 20, 2012 13:23:12.178 EDT	000000c3	Start of processing for SIBus (ReceiveNoWait).
End SIBus (ReceiveNoWait)	May 20, 2012 13:23:12.218 EDT	000000c3	End of processing for SIBus (ReceiveNoWait).
End JMS (ReceiveInBound)	May 20, 2012 13:23:12.248 EDT	000000c3	End of processing for JMS (ReceiveInBound).
Log message	May 20, 2012 13:23:12.248 EDT	000000c3	Successfully received Message of type null
Log message	May 20, 2012 13:23:12.258 EDT	000000c3	Creating Object Message
Log message	May 20, 2012 13:23:12.258 EDT	000000c3	Creation of Object Message Successful
Start JMS (SendMessage)	May 20, 2012 13:23:12.258 EDT	000000c3	Start of processing for JMS (SendMessage).
Start SIBus (Send)	May 20, 2012 13:23:12.258 EDT	000000c3	Start of processing for SIBus (Send).
End SIBus (Send)	May 20, 2012 13:23:12.679 EDT	000000c3	End of processing for SIBus (Send).
End JMS (SendMessage)	May 20, 2012 13:23:12.679 EDT	000000c3	End of processing for JMS (SendMessage).
Log message	May 20, 2012 13:23:12.699 EDT	000000c3	Object Message sent successfully: 1024
Start JMS (ReceiveInBound)	May 20, 2012 13:23:12.699 EDT	000000c3	Start of processing for JMS (ReceiveInBound).
Start SIBus (ReceiveNoWait)	May 20, 2012 13:23:12.699 EDT	000000c3	Start of processing for SIBus (ReceiveNoWait).
End SIBus (ReceiveNoWait)	May 20, 2012 13:23:12.729 EDT	000000c3	End of processing for SIBus (ReceiveNoWait).
End JMS (ReceiveInBound)	May 20, 2012 13:23:12.739 EDT	000000c3	End of processing for JMS (ReceiveInBound).

3. Since the data snapshot feature was enabled within the server logging settings, the following message data files were created in directory `log_directory/snapdata`:

- Message Send for Message Type Text (*.txt)
- Message Receive for Message Type Text (*.txt)
- Message Send for Message Type Map (*.map)
- Message Receive for the Message Type Map (*.map)

These files are available in lab directory `/WASv85Labs/XCT/Scenario4/snapdata`. You can open these files in a text editor to view the body and map information for the send and receive messages.

4. Close all log files (open in text editors and the log viewer).

IBM Tasks - Wrap-up

To prepare for the next scenario, this is what was done:

1. Used the deployment manager administrative console to stop server1.
    a. Displayed the following console screen: *Servers > Server Types > WebSphere Application Servers*.
    b. Selected (checked) **server1**.
    c. Clicked **Stop** and responded to any confirmation prompts as appropriate.

2. Deleted the `SystemOut.log` file for server1, for example,

Windows
```
C:\Program Files\IBM\WebSphere\AppServer\profiles\AppSrv01\logs\server1\SystemOut.log
```

UNIX or Linux
```
/opt/IBM/WebSphere/AppServer/profiles/AppSrv01/logs/server1/SystemOut.log
```

HTTP Instrumentation Analysis

Scenario 5: Basic HTTP Message

Note: This scenario requires a WebSphere Application Server V8.5 installation. For more information, see the Prerequisites section of this document.

You will first use the administrative console to enable HPEL (High Performance Extensible Logging) and set the log/trace string to an appropriate value. Next, you will enable XCT using the wsadmin command line interface. Then you will use the command line log viewer to display the latest instance of a request within the HPEL logs.

1. Open a command prompt and use the startServer command to start the deployment manager and the node agent, for example,

Windows
```
C:\Program Files\IBM\WebSphere\AppServer\profiles\Dmgr01\bin\startServer dmgr
C:\Program Files\IBM\WebSphere\AppServer\profiles\AppSrv01\bin\startServer nodeagent
```

UNIX or Linux
```
/opt/IBM/WebSphere/AppServer/profiles/Dmgr01/bin startServer.sh dmgr
/opt/IBM/WebSphere/AppServer/profiles/AppSrv01/bin startServer.sh nodeagent
```

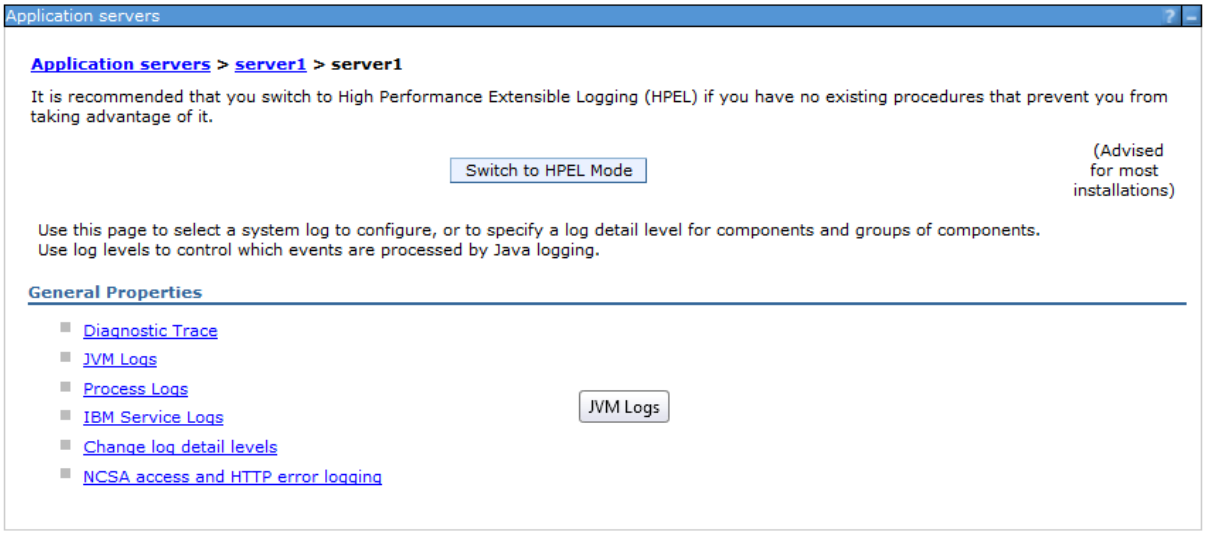
2. Use the operating system shell to launch the deployment manager administrative console. Next, log in to the console. Then use the console to enable HPEL and set the log/trace string to `*=info:HTTPChannel=all:GenericBNF=all`

a. Enable HPEL for server1.

i. Display the following console screen: *Servers* > *Server Types* > *WebSphere Application Servers*.

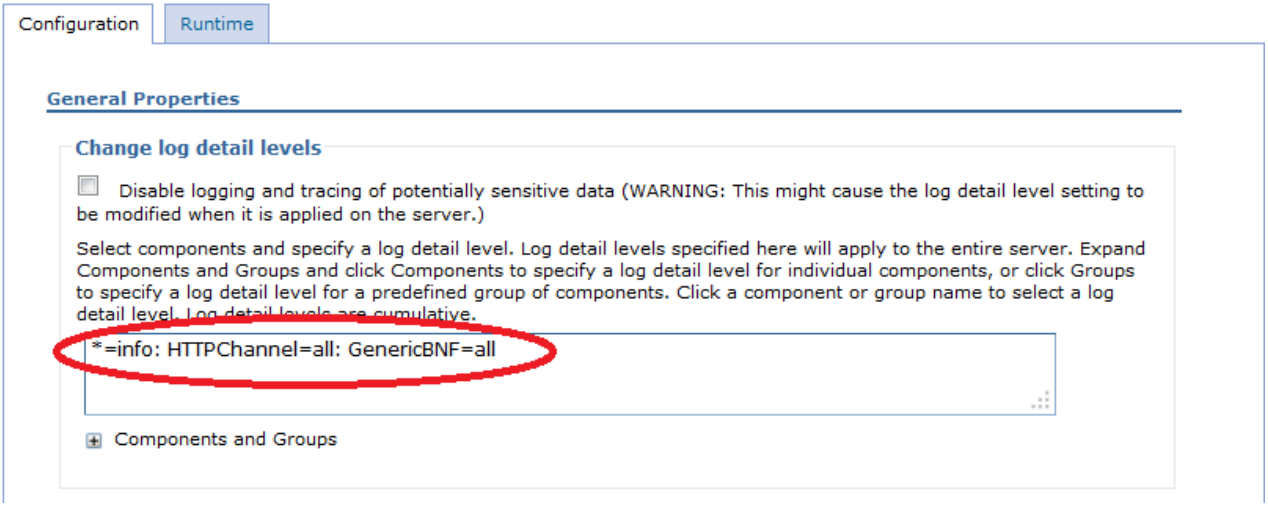
ii. Click **server1**.

iii. Click **Logging and Tracing** to display the following screen:



iv. Click **Switch to HPEL Mode**.

v. Click **Change log detail levels**. Then change the log/trace string to the following (as illustrated below): `*=info:HTTPChannel=all:GenericBNF=all`



vi. Click **OK**. Then click **Save** to save the changes to the master configuration.

c. Start server1.

i. Display the following console screen: *Servers* > *Server Types* > *WebSphere Application Servers*

ii. Select (check) **server1**.

iii. Click **Start** and respond to the message prompts as appropriate.

3. Start the wsadmin command line interface.

a. Open a command prompt and change to the deployment manager profile bin directory, for example,

Windows: `C:\Program Files\IBM\WebSphere\AppServer\profiles\Dmgr01\bin`
UNIX or Linux: `/opt/IBM/WebSphere/AppServer/profiles/Dmgr01/bin`

b. Run the following command to start wsadmin:

Windows: `wsadmin -lang jython -userName was -password was`
UNIX or Linux: `wsadmin.sh -lang jython -userName was -password was`

4. Run the following wsadmin commands to configure XCT for server1:

Note: If you receive an “AdminControl service is not available” error message after running the first command, wait one minute and try again.

a. Assign the name of the HPEL Control Service MBean for server1 to variable HPELControlMBean.

```
HPELControlMBean=AdminControl.queryNames('cell=DmgrCell01,node=AppSrv01Node,
type=HPELControlService,process=server1,*')
```

Output
None

b. Print the value of the variable containing the name of the HPEL Control Service MBean for server1.

```
print HPELControlMBean
```

Output
```
WebSphere:name=HPELControlService,process=server1,platform=proxy,node=
AppSrv01Node,version=8.5.0.0,type=HPELControlService,mbeanIdentifier=c
ells/DmgrCell01/nodes/AppSrv01Node/servers/server1/server.xml#RASLoggi
ngService_1334846165273,cell=DmgrCell01,spec=1.0
```

c. Print the current properties of the HPEL Control Service MBean for server1.

```
print AdminControl.getAttributes(HPELControlMBean)
```

Output
```
[[traceSpecification *=info] [rawTraceFilterEnabled false]
[correlationEnabled false] [xctLevel REQUESTID] ]
```

d. Enable XCT by setting the `correlationEnabled` property of the HPEL Control Service MBean for server1 to `true`.

```
AdminControl.setAttribute(HPELControlMBean, "correlationEnabled", "true")
```

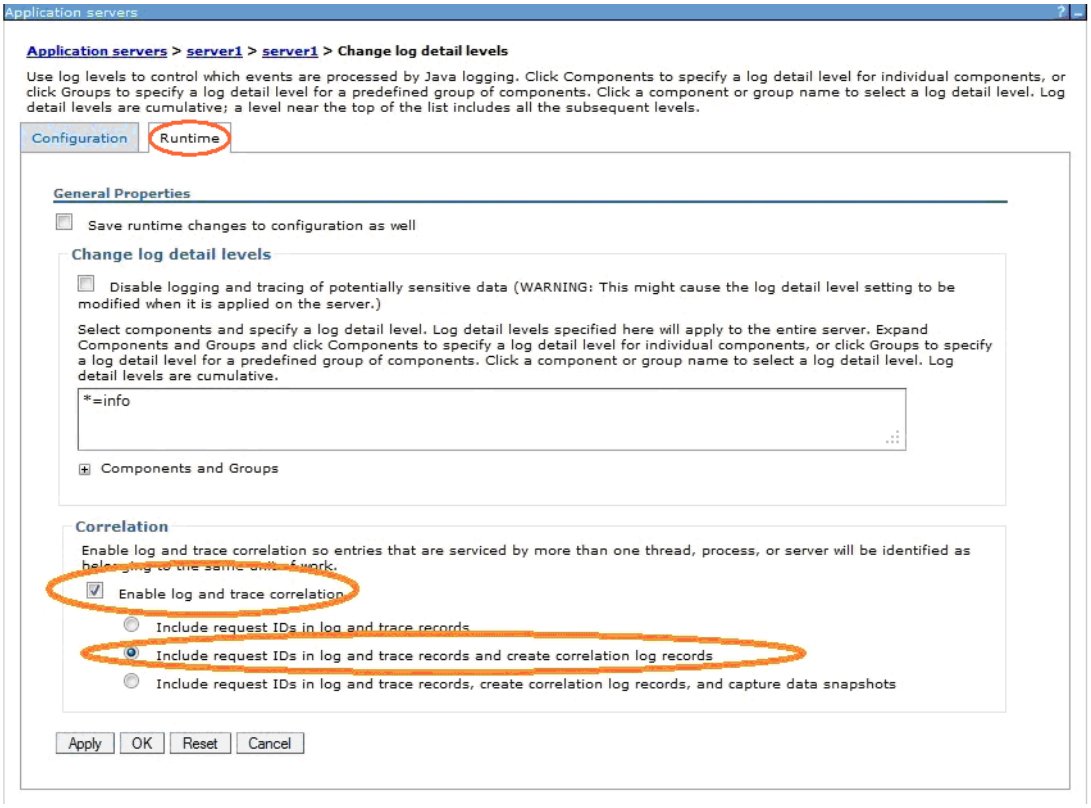
Output
```
''
```

e. Print the updated properties of the HPEL Control Service MBean for server1 to verify the change.

```
print AdminControl.getAttributes(HPELControlMBean)
```

Output
```
[[traceSpecification *=info] [rawTraceFilterEnabled false]
[correlationEnabled true] [xctLevel REQUESTID] ]
```

The equivalent administrative console settings for XCT are illustrated below.



5. Using a web browser, load the snoop servlet in order to send to cause several trace entries to be stored on the same server (server1), for example,

`http://localhost:9080/snoop`

6. Use the command line log viewer to display the latest instance of a request within the HPEL logs.

a. Open a command prompt and change to the profile bin directory for server1, for example,

Windows: `C:\Program Files\IBM\WebSphere\AppServer\profiles\AppSrv01\bin`
UNIX or Linux: `/opt/IBM/WebSphere/AppServer/profiles/AppSrv01/bin`

b. Run the following command to display the latest instance of a request within the HPEL logs:

Windows: `logViewer -latestInstance`
UNIX or Linux: `logViewer.sh -latestInstance`

The resulting log viewer output should look similar to the following.

```
[5/9/12 18:23:25:647 EDT] 0000002e BNFHeadersImp 3   Releasing marshall buffer: PooledWsByteBuf
ferImpl: ID: 121 java.nio.DirectByteBuffer[pos=92 lim=92 cap=1024] Owner Count: 1 From pool: WS
ByteBufferPool: com.ibm.ws.buffermgmt.impl.WsByteBufferPool@201698b3 buffer size: 1024
[5/9/12 18:23:25:647 EDT] 0000002e BNFHeadersImp <  clear Exit
[5/9/12 18:23:25:647 EDT] 0000002e HttpObjectFac 3   releaseRequest: com.ibm.ws.http.channel.im
pl.HttpRequestMessageImpl@49ec0b7a
[5/9/12 18:23:25:647 EDT] 0000002e HttpResponseM 1   Destroying this response: com.ibm.ws.http.
channel.impl.HttpResponseMessageImpl@6b0a59ef
[5/9/12 18:23:25:647 EDT] 0000002e BNFHeadersImp 3   Destroying these headers: com.ibm.ws.http.
channel.impl.HttpResponseMessageImpl@6b0a59ef
[5/9/12 18:23:25:648 EDT] 0000002e HttpResponseM 1   Clearing this response: com.ibm.ws.http.ch
annel.impl.HttpResponseMessageImpl@6b0a59ef
[5/9/12 18:23:25:648 EDT] 0000002e BNFHeadersImp >  clear Entry
[5/9/12 18:23:25:648 EDT] 0000002e BNFHeadersImp >  clearAllHeaders() Entry
[5/9/12 18:23:25:648 EDT] 0000002e HttpBaseMessa 1   Removing: Connection:null
[5/9/12 18:23:25:648 EDT] 0000002e HttpBaseMessa 1   Removing: Content-Length:null
[5/9/12 18:23:25:648 EDT] 0000002e BNFHeadersImp <  clearAllHeaders() Exit
[5/9/12 18:23:25:648 EDT] 0000002e BNFHeadersImp 3   Removing reference to parse buffer: Pooled
WsByteBufferImpl: ID: 187 java.nio.DirectByteBuffer[pos=220 lim=220 cap=8192] Owner Count: 0 Fr
om pool: WSByteBufferPool: com.ibm.ws.buffermgmt.impl.WsByteBufferPool@ab1f8875 buffer size: 81
92
[5/9/12 18:23:25:648 EDT] 0000002e BNFHeadersImp <  clear Exit
[5/9/12 18:23:25:648 EDT] 0000002e HttpObjectFac 3   releaseResponse: com.ibm.ws.http.channel.i
mpl.HttpResponseMessageImpl@6b0a59ef
[5/9/12 18:23:25:648 EDT] 0000002e HttpServiceCo <  destroy Exit
[5/9/12 18:23:25:648 EDT] 0000002e HttpOutboundC 3   Stopping channel with (0) : HTTP-out_CFINT
ERNAL_CHILD_0 com.ibm.ws.http.channel.outbound.impl.HttpOutboundChannel@cc3e4c4c
[5/9/12 18:23:25:649 EDT] 0000002e HttpOutboundC 3   Destroying channel: HTTP-out_CFINTERNAL_CH
ILD_0 com.ibm.ws.http.channel.outbound.impl.HttpOutboundChannel@cc3e4c4c
[5/9/12 18:23:25:675 EDT] 0000001e HttpInboundLi <  ready Exit
[5/9/12 18:23:25:677 EDT] 0000002a HttpInboundCh 1   Stop: Inbound channel Inbound3HTTP_CFINTER
NAL_CHILD_0 state=1
[5/9/12 18:23:30:677 EDT] 00000029 HttpInboundCh 1   Stop: Inbound channel Inbound3HTTP_CFINTER
NAL_CHILD_0 state=0
[5/9/12 18:23:32:679 EDT] 0000002a HttpInboundCh 1   destroy: Inbound channel Inbound3HTTP_CFIN
TERNAL_CHILD_0 state=0
Operation Complete
Processed 50,383 records in 30.484 seconds (1,652.769 records per second).
```

c. Run the following command to display the log messages in advanced format:

Windows: `logViewer -latestInstance -format advanced`
UNIX or Linux: `logViewer.sh -latestInstance -format advanced`

The resulting log viewer output should look similar to the following.

```
[5/9/12 19:27:20:873 EDT] 0000002b  > UOW= source=com.ibm.ws.genericbnf.impl.BNFHeadersImpl met
hod=marshallHeaders org=IBM prod=WebSphere component=Application Server thread=[WebContainer :
2] requestID=[AABZvPwW/cp-AAAAAAAAAAH]
         Entry
[5/9/12 19:27:20:873 EDT] 0000002b  1 UOW= source=com.ibm.ws.genericbnf.impl.BNFHeadersImpl org
=IBM prod=WebSphere component=Application Server thread=[WebContainer : 2] requestID=[AABZvPwW/
cp-AAAAAAAAAAH]
         Marshalling: Key: X-Powered-By Ordinal: 93 undefined: true [Servlet/3.0]
[5/9/12 19:27:20:874 EDT] 0000002b  1 UOW= source=com.ibm.ws.genericbnf.impl.BNFHeadersImpl org
=IBM prod=WebSphere component=Application Server thread=[WebContainer : 2] requestID=[AABZvPwW/
cp-AAAAAAAAAAH]
         Marshalling: Key: Content-Type Ordinal: 12 undefined: false [text/html;charset=ISO-88
59-1]
[5/9/12 19:27:20:874 EDT] 0000002b  1 UOW= source=com.ibm.ws.genericbnf.impl.BNFHeadersImpl org
=IBM prod=WebSphere component=Application Server thread=[WebContainer : 2] requestID=[AABZvPwW/
cp-AAAAAAAAAAH]
         Marshalling: Key: Content-Language Ordinal: 16 undefined: false [en-US]
[5/9/12 19:27:20:874 EDT] 0000002b  1 UOW= source=com.ibm.ws.genericbnf.impl.BNFHeadersImpl org
=IBM prod=WebSphere component=Application Server thread=[WebContainer : 2] requestID=[AABZvPwW/
cp-AAAAAAAAAAH]
         Marshalling: Key: Content-Length Ordinal: 11 undefined: false [9288]
[5/9/12 19:27:20:874 EDT] 0000002b  1 UOW= source=com.ibm.ws.genericbnf.impl.BNFHeadersImpl org
=IBM prod=WebSphere component=Application Server thread=[WebContainer : 2] requestID=[AABZvPwW/
cp-AAAAAAAAAAH]
```

You can use a request ID such as `requestID=[AABZvPwW/cp-AAAAAAAAAAA]` to correlate requests together. For more information, see Scenario 1.

In cases where a request spans two servers, you can launch the log viewer with the request ID (to display only messages related to the request ID), for example,

Windows: `logViewer -includeExtensions requestID=AABZvPwW/cp-AAAAAAAAAAA`
UNIX or Linux: `logViewer.sh -includeExtensions requestID=AABZvPwW/cp-AAAAAAAAAAA`

An HTTP request will include XCT begins and XCT ends.

Example HTTP XCT begin

```
BEGIN AABZvPwW/cp-AAAAAAAAAAB 00000000000-ccccccccc2 HTTPCF(InboundRequest
/ibm/console/com.ibm.ws.console.probdetermination/loggingSettingsComponents.
jsp RequestContext(-1770557153))
```

Notes

- The blue text is the type of request that is beginning.
- The orange text is the URI of the request.
- The red text is the Channel Framework request context ID.
- For more information about the components of the message, see Scenario 1.

Example HTTP XCT end

```
END AABZvPwW/cp-AAAAAAAAAAB 00000000000-ccccccccccc2 HTTPCF(InboundRequest
RC=200 RequestContext(-1770557153))
```

Notes

- The blue text is the type of request that is beginning.
- The orange text is the return code of the request.
- The red text is the channel Framework request context ID.
- For more information about the components of the message, see Scenario 1.