

SFA Modernization Partner

United States Department of Education

Student Financial Assistance



EAI Core Architecture
EAI Build and Test Report
Release 2

Task Order #54

Deliverable #54.1.8

October 31, 2001

TABLE OF CONTENTS

TABLE OF CONTENTS	2
1 EXECUTIVE SUMMARY	8
1.1 OBJECTIVES	8
1.2 APPROACH	8
1.3 DESCRIPTION OF SECTIONS	8
1.4 SCOPE	9
1.5 INTENDED AUDIENCE	9
2 EAI TEST METHODOLOGY	10
2.1 TESTING PROCESS.....	10
2.1.1 Test Scenario Description.....	10
2.1.2 Test Scenario Detailed Design Description.....	10
2.1.3 Test Scenario Dependencies.....	11
2.1.4 Test Scenario Inputs	11
2.1.5 Test Scenario Expected Results.....	11
2.2 TEST ENVIRONMENT ARCHITECTURE DESIGN	12
3 EAI COMPONENT TESTS	13
3.1 EAI COMPONENT TEST FOR CENTRAL PROCESSING SYSTEM (CPS)	13
3.1.1 CPS Test Scenario Description.....	13
3.1.2 CPS Test Scenario Detailed Design Description.....	14
3.1.3 CPS Test Scenario Dependencies.....	16
3.1.4 CPS Test Scenario Inputs	17
3.1.5 CPS Test Scenario Expected Results.....	18
3.2 EAI COMPONENT TEST FOR DIRECT LOAN SERVICING SYSTEM (DLSS)	21
3.2.1 DLSS Test Scenario Description.....	21
3.2.2 DLSS Test Scenario Detailed Design Description	22
3.2.3 DLSS Test Scenario Dependencies	25
3.2.4 DLSS Test Scenario Inputs	28
3.2.5 DLSS Test Scenario Expected Results	28
3.3 EAI COMPONENT TEST FOR ELECTRONIC CAMPUS BASED SYSTEM (ECBS) - BATCH.	30
3.3.1 eCBS – Batch Test Scenario Description	30
3.3.2 eCBS – Batch Test Scenario Detailed Design Description	30
3.3.3 eCBS – Batch Test Scenario Dependencies	33

3.3.4	eCBS –Batch Test Scenario Inputs.....	33
3.3.4.1	Normal Test Cycle.....	33
3.3.4.2	Expected Error Test Cycle.....	34
3.3.4.3	Unexpected Error Test Cycle	34
3.3.4.4	Regression Test Cycle	35
3.3.5	eCBS – Batch Test Scenario Expected Results	35
3.3.5.1	Expected Results for Normal Test Cycle	35
3.3.5.2	Expected Results for Expected Error Test Cycle	36
3.3.5.3	Expected Results for Unexpected Error Test Cycle	36
3.3.5.4	Expected Results for Regression Test Cycle.....	38
3.4	EAI COMPONENT TEST FOR ELECTRONIC CAMPUS BASED SYSTEM (eCBS) – REAL TIME.....	40
3.4.1	eCBS – Real Time Test Scenario Description.....	40
3.4.2	eCBS – Real Time Test Scenario Detailed Design Description.....	40
3.4.3	eCBS – Real Time Test Scenario Dependencies	43
3.4.4	eCBS – Real Time Test Scenario Inputs	43
3.4.4.1	Normal Test Cycle.....	43
3.4.4.2	Expected Error Test Cycle.....	43
3.4.4.3	Unexpected Error Test Cycle	44
3.4.4.4	Regression Test Cycle	44
3.4.5	eCBS – Real Time Test Scenario Expected Results.....	44
3.4.5.1	Expected Results for Normal Test Cycle	44
3.4.5.2	Expected Results for Expected Error Test Cycle	45
3.4.5.3	Expected Results for Unexpected Error Test Cycle	45
3.4.5.4	Expected Results for Regression Test Cycle.....	46
3.5	EAI COMPONENT TEST FOR FINANCIAL MANAGEMENT SYSTEM (FMS) - BATCH.....	47
3.5.1	FMS – Batch Test Scenario Description	47
3.5.2	FMS – Batch Test Scenario Detailed Design Description.....	47
3.5.3	FMS – Batch Test Scenario Dependencies.....	50
3.5.4	FMS – Batch Test Scenario Inputs	50
3.5.4.1	Normal Test Cycle.....	50
3.5.4.2	Expected Error Test Cycle.....	50
3.5.4.3	Unexpected Error Test Cycle	51
3.5.4.4	Regression Test Cycle	51

3.5.5	FMS – BatchTest Scenario Expected Results	52
3.5.5.1	Expected Results for Normal Test Cycle	52
3.5.5.2	Expected Results for Expected Error Test Cycle	52
3.5.5.3	Expected Results for Unexpected Error Test Cycle	53
3.5.5.4	Expected Results for Regression Test Cycle.....	54
3.6	EAI COMPONENT TEST FOR FINANCIAL MANAGEMENT SYSTEM (FMS) - REAL-TIME	55
3.6.1	FMS – Real Time Test Scenario Description	55
3.6.2	FMS – Real Time Test Scenario Detailed Design Description	55
3.6.3	FMS – Real Time Test Scenario Dependencies	58
3.6.4	FMS – Real Time Test Scenario Inputs.....	58
3.6.4.1	Normal Test Cycle.....	58
3.6.4.2	Expected Error Test Cycle.....	58
3.6.4.3	Unexpected Error Test Cycle	59
3.6.4.4	Regression Test Cycle	59
3.6.5	FMS – Real Time Test Scenario Expected Results	59
3.6.5.1	Expected Results for Normal Test Cycle	59
3.6.5.2	Expected Results for Expected Error Test Cycle	60
3.6.5.3	Expected Results for Unexpected Error Test Cycle	61
3.6.5.4	Expected Results for Regression Test Cycle.....	63
3.7	EAI COMPONENT TEST FOR LO SYSTEM – ELECTRONIC MASTER PROMISSORY NOTE (EMPN).....	65
3.7.1	LO System –(eMPN) Test Scenario Description.....	65
3.7.2	LO System – (eMPN) Test Scenario Detailed Design Description.....	65
3.7.3	LO System – (eMPN) Test Scenario Dependencies.....	67
3.7.4	LO System – (eMPN) Test Scenario Inputs	67
3.7.4.1	Normal Test Cycle.....	67
3.7.4.2	Expected Error Test Cycle.....	68
3.7.4.3	Unexpected Error Test Cycle	68
3.7.4.4	Regression Test Cycle	69
3.7.5	LO System – (eMPN) Test Scenario Expected Results.....	69
3.7.5.1	Expected Results for Normal Test Cycle	69
3.7.5.2	Expected Results for Expected Error Test Cycle	71
3.7.5.3	Expected Results for Unexpected Error Test Cycle	71
3.7.5.4	Expected Results for Regression Test Cycle.....	71

3.8	EAI COMPONENT TEST FOR LO SYSTEM – PROMISSORY NOTE IMAGING (P-NOTE IMAGING)	73
3.8.1	LO System –(P-Note Imaging) Test Scenario Description	74
3.8.2	LO System – (P-Note Imaging) Test Scenario Detailed Design Description.....	74
3.8.3	LO System – (P-Note Imaging) Test Scenario Dependencies.....	76
3.8.4	LO System – (P-Note Imaging) Test Scenario Inputs	76
3.8.4.1	Normal Test Cycle.....	76
3.8.4.2	Expected Error Test Cycle.....	76
3.8.4.3	Unexpected Error Test Cycle	76
3.8.4.4	Regression Test	77
3.8.5	LO System – (P-Note Imaging) Test Scenario Expected Results	77
3.8.5.1	Expected Results for Normal Test Cycle	77
3.8.5.2	Expected Results for Expected Error Test Cycle	77
3.8.5.3	Expected Results for Unexpected Error Test Cycle	77
3.8.5.4	Expected Results for Regression Test Cycle.....	77
3.9	EAI COMPONENT TEST FOR NATIONAL STUDENT LOAN DATA SYSTEM (NSLDS) – BATCH.	78
3.9.1	NSLDS – Batch Test Scenario Description.....	78
3.9.2	NSLDS – Batch Test Scenario Detailed Design Description	79
3.9.3	NSLDS - Batch Test Scenario Dependencies.....	83
3.9.4	NSLDS – Batch Test Scenario Inputs	85
3.9.5	NSLDS - Batch Test Scenario Expected Results	86
3.10	EAI COMPONENT TEST FOR NATIONAL STUDENT LOAN DATA SYSTEM (NSLDS) – COOL:GEN	87
3.10.1	NSLDS – Cool:Gen Test Scenario Description	87
3.10.2	NSLDS – Cool:Gen Test Scenario Detailed Design Description.....	87
3.10.3	NSLDS – Cool:Gen Test Scenario Dependencies.....	91
3.10.4	NSLDS – Cool:Gen Test Scenario Inputs	91
3.10.5	NSLDS – Cool:Gen Test Scenario Expected Results.....	91
3.11	EAI COMPONENT TEST FOR POST-SECONDARY EDUCATION PARTICIPANTS SYSTEM (PEPS).	94
3.11.1	PEPS Test Scenario Description.....	94
3.11.2	PEPS Test Scenario Detailed Design Description.....	94
3.11.3	PEPS Test Scenario Dependencies.....	96
3.11.4	PEPS Test Scenario Inputs	97

3.11.5	PEPS Test Scenario Expected Results.....	98
3.12	EAI COMPONENT TEST FOR STUDENT AID INTERNET GATEWAY (SAIG/bTRADE) ...	100
3.12.1	SAIG/bTrade Test Scenario Description.....	100
3.12.2	SAIG/bTrade Test Scenario Detailed Design Description	100
3.12.3	SAIG/bTrade Test Scenario Dependencies	103
3.12.4	SAIG/bTrade Test Scenario Inputs.....	104
3.12.5	SAIG/bTrade Test Scenario Expected Results	105
4	EAI COMPONENT MIGRATION	107
4.1	EAI COMPONENT MIGRATION FOR CENTRAL PROCESSING SYSTEM (CPS) AND NATIONAL STUDENT LOAN DATA SYSTEM (NSLDS).....	107
4.1.1	System Installation	107
4.1.2	Networking	107
4.1.3	Configuration on the CPS System.....	107
4.1.4	Configuration on the NSLDS System	108
4.1.5	MQ Object Definitions	108
4.1.6	Other CPS and NSLDS Migration Considerations.....	108
4.2	EAI COMPONENT MIGRATION FOR DIRECT LOAN SERVICING SYSTEM (DLSS)	109
4.2.1	System Installation	109
4.2.2	Networking	109
4.2.3	Configuration.....	109
4.3	EAI COMPONENT MIGRATION FOR ELECTRONIC CAMPUS BASED SYSTEM (ECBS) ..	110
4.3.1	System Installation	110
4.3.2	Networking.....	110
4.3.3	Configuration.....	110
4.4	EAI COMPONENT MIGRATION FOR FINANCIAL MANAGEMENT SYSTEM (FMS)	111
4.4.1	System Installation	111
4.4.2	Networking.....	111
4.4.3	Configuration.....	111
4.5	EAI COMPONENT MIGRATION FOR LO SYSTEM – ELECTRONIC MASTER PROMISSORY NOTE (EMPN).....	112
4.5.1	System Installation	112
4.5.2	Networking.....	113
4.5.3	Configuration.....	113
4.6	EAI COMPONENT MIGRATION FOR LO SYSTEM – PROMISSORY NOTE IMAGING (P- NOTE IMAGING).....	114

4.6.1	System Installation	114
4.6.2	Networking	114
4.6.3	Configuration.....	114
4.7	EAI COMPONENT MIGRATION FOR NATIONAL STUDENT LOAN DATA SYSTEM (NSLDS)-COOL:GEN.....	115
4.7.1	System Installation	115
4.7.2	Networking	116
4.7.3	Configuration.....	116
4.8	EAI COMPONENT MIGRATION FOR POST-SECONDARY EDUCATION PARTICIPANTS SYSTEM (PEPS)	117
4.8.1	System Installation	117
4.8.2	Networking	117
4.8.3	Configuration.....	117
4.9	EAI COMPONENT MIGRATION FOR STUDENT AID INTERNET GATEWAY (SAIG/BTRADE)	118
4.9.1	System Installation	118
4.9.2	Networking	118
4.9.3	Configuration.....	118
5	SIEBEL ADAPTER ANALYSIS	119
5.1	MQSERIES/SIEBEL INTEGRATION OPTIONS	119
5.1.1	Siebel EAI	119
5.1.2	Neon (Sybase) Adapter for Siebel e-Business Applications	119
5.1.3	Propelis EAI (Enterprise Access Integrator)	120
5.1.4	SFA Custom Solution.....	120
5.1.5	MQSeries Adapter Offering (MQAO)	120
5.2	CONCLUSION	121

1 EXECUTIVE SUMMARY

The EAI Core Build and Test Report validates the architecture design, services, and interfaces provided by the Release 1 and 2 EAI Core Architecture to support the modernization effort of the Student Financial Assistance (SFA) Information Technology (IT) Enterprise.

1.1 Objectives

The objective of the EAI Core Build and Test Report deliverable is to provide the information necessary to execute and validate the EAI Core Architecture (Release 1 and 2), which will demonstrate that the EAI Core architecture provides the functional capabilities required for connecting the Release 1 and 2 legacy systems to the EAI Bus.

During the Build and Test phase all Release 1 and 2 architectural components shall be integrated into the EAI Core Architecture and shall be verified to execute properly.

The tests outlined in this report are based on the functional scenarios developed to validate the MQSeries Messaging and Transformation activities as designed in the EAI Technical Specifications Deliverable 54.1.6.

1.2 Approach

The following approach was used to develop the EAI Build and Test Report:

- Review of the EAI Core Architecture from an interface perspective.
- Review of the functional services defined for each Release 1 and 2 legacy system.
- Development of test scenarios that validate the MQSeries messaging and transformation architecture to connect each of the Release 1 and 2 legacy systems to the EAI Bus.

1.3 Description of Sections

This deliverable is divided into the following sections:

- Section 1 – Executive Summary
The Executive Summary provides an introduction and overview of the EAI Build and Test Report.
- Section 2 – EAI Test Methodology
The Build and Test procedures will focus on the validation of the architectural design of the Release 1 and 2 EAI Core Architecture. The test scenario descriptions will provide the objective and an overview of the test to be performed, function(s) exercised, and any other pertinent aspects of the test scenario. Test scenario inputs, expected results, and acceptance criteria are discussed.
- Section 3 – EAI Component Tests
The component tests for each legacy system are detailed and diagrams are used to explain the flow of data between the different EAI components. As messages flowed from queue to

queue the data was verified. The same component testing logic was applied to the MQSI message flows.

- Section 4 – EAI Component Migration

The migration of the Release 1 and 2 EAI Core components as designed and developed are dependent upon the specific legacy system requirements, required licensing, and legacy system owner approval for migrating each legacy system to production environment. For each legacy system the system installation pre-requisites, the networking dependencies, and the required configuration are defined and documented.

- Section 5 – Siebel Adapter Analysis

The analysis of five Siebel adapters is documented followed by a recommended solution.

1.4 Scope

MQSeries messaging and transformation activities were developed for the following Release 1 and 2 SFA legacy systems:

- Central Processing System (CPS)
- Direct Loan Servicing System (DLSS)
- Electronic Campus Based System (eCBS)
- Financial Management System (FMS)
- LO System – Electronic Master Promissory Note (eMPN)
- LO System – Promissory Note Imaging (P-Note Imaging)
- National Student Loan Data System (NSLDS)
- Post-Secondary Education Participants System (PEPS)
- Student Aid Internet Gateway (SAIG/bTrade)

The Build portion will ensure that all required components defined in the Release 1 and 2 EAI Core Architecture are installed, configured, and are operational.

The Test portion will ensure that the actual outputs produced conform to the expected outputs as defined by each test scenario.

1.5 Intended Audience

The EAI Build and Test Report document is intended for those who need to understand the validation of capabilities implemented at the completion of EAI Core Architecture Release 2.

2 EAI TEST METHODOLOGY

2.1 Testing Process

The following Build and Test procedures will focus on the validation of the architectural design of the Release 1 and 2 EAI Core Architecture.

To assist in the execution and demonstration of the EAI Core functionality for each Release 1 and 2 legacy system the EAI Core team developed a test driver application. This test driver application does not provide any business functionality, but provides a user interface for entering or retrieving message data from a file and sending messages to each legacy system for processing. Upon completion of each test execution, the results are returned to the test driver application for display and to be written to an output directory to save a hard copy of the test results. This test driver application is not meant to be a production ready application, but as an aid in the execution and demonstration of the Release 1 and 2 EAI Core Architecture validation process.

The testing process for the EAI Core Architecture Release 2 legacy systems will include four testing cycles: Normal testing, Expected error testing, Unexpected error testing, and Regression testing. The Normal Test Cycle will input data in the correct format and ensure successful validation. The Expected Error Test Cycle will input data created to generate errors the system is expected to process and handle. The Unexpected Error Test Cycle includes results that were unplanned. This cycle may also include unplanned or simulated instances of application or environment failures and outages. The Regression Test Cycle will retest approximately 30% of test scenarios from the preceding cycles, generally including scenarios from the Normal Test Cycle and Expected Error Test Cycle.

For each system tested, the following sections will be defined:

- Test Scenario Description
- Test Scenario Detailed Design Description
- Test Scenario Dependencies
- Test Scenario Inputs
- Test Scenario Expected Results

2.1.1 Test Scenario Description

This section provides the objective and an overview of the test to be performed and function(s) exercised relative to the MQSeries and MQSeries Integrator and legacy system tests.

2.1.2 Test Scenario Detailed Design Description

This section provides test scenario design detail via diagrams and text descriptions of what the diagrams represent. Each detail diagram depicts the test scenario's process flow by identifying each component and the interfaces involved. The text portion provides a description of what is occurring within each process module, what information is being shared, and how it is being transferred between products.

2.1.3 Test Scenario Dependencies

This section defines the system dependencies, both hardware and software that must be met prior to test execution.

2.1.4 Test Scenario Inputs

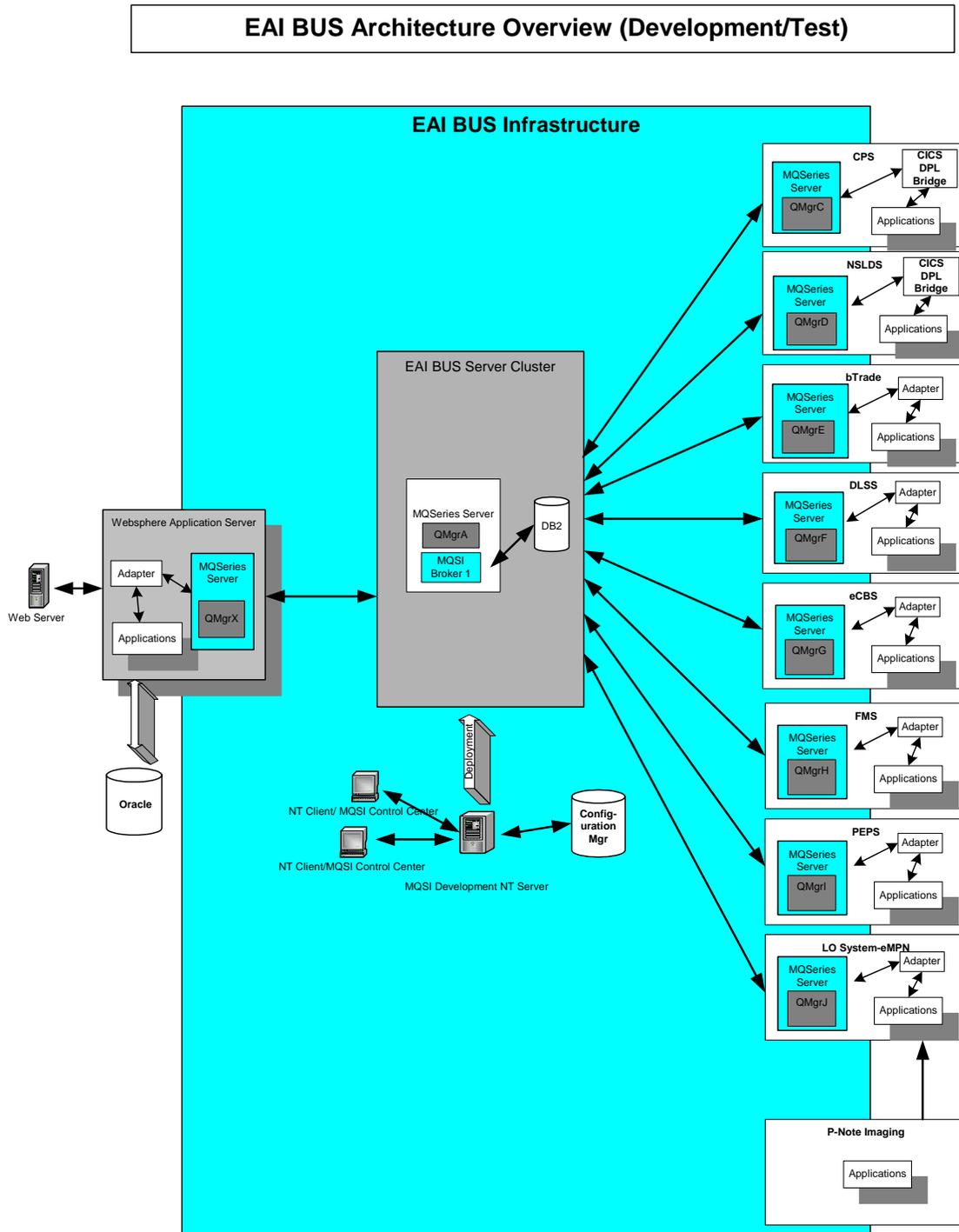
This section provides a description of the data required to execute the test scenario.

2.1.5 Test Scenario Expected Results

This section provides the expected results, or output, of the particular test scenario. The expected results for each test scenario are the same as if the transaction were executed on each system without using MQSeries as the message transport. Acceptance of the test is gained by demonstrating to Accenture and SFA that the transaction is executed successfully (i.e. the expected results are returned).

2.2 Test Environment Architecture Design

The intent of the diagram is to show the components of the EAI Bus Architecture implemented for Release 1 and 2 of the EAI Core. The location of MQSeries, MQSI, databases and adapters are shown.



3 EAI COMPONENT TESTS

3.1 EAI Component Test for Central Processing System (CPS)

The CPS system is comprised of CICS transactions developed to provide the required application functionality. EAI enablement of the CPS system required the installation and configuration of the MQSeries CICS Adapter to provide a real-time request/reply of system data through the EAI Middleware. The EAI Bus allows applications to send messages to the CPS system and utilize the existing CICS transactions to retrieve data in real-time from an external server, i.e. a web site.

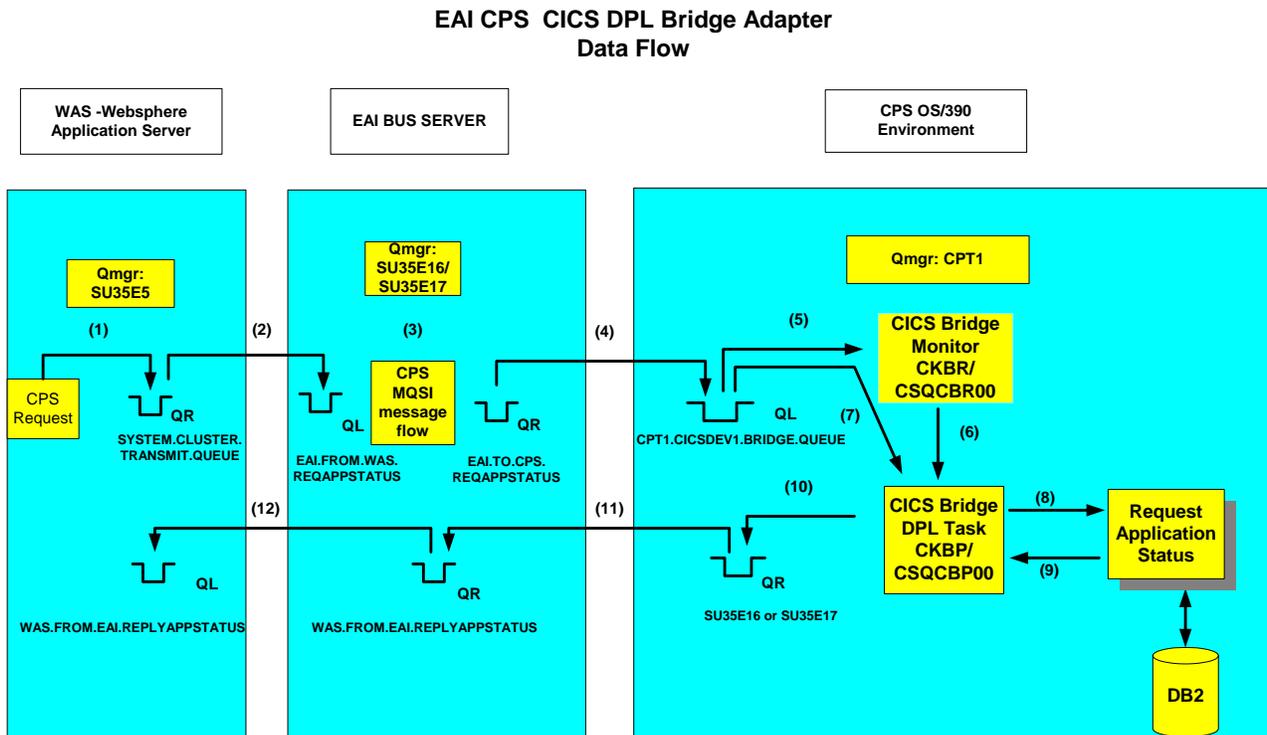
3.1.1 CPS Test Scenario Description

The application to be used for the EAI CPS component test is the CPS application status inquiry program. The CPS application status inquiry is a CICS program that provides the loan status for a specified input Social Security Number (SSN). A request message, consisting of a SSN and a name id, is originated from the test application. The message is routed through the MQSeries messaging infrastructure using the EAI bus infrastructure. The message is transformed by MQSI to append the required records to the message data. The message is then routed from the EAI Bus to the CPS system using the MQSeries CICS DPL Bridge. The CICS DPL Bridge enables the originating application to access information residing in the CPS OS/390 platform by invoking the CICS program running in the CICS environment to process the inquiry message and send a reply back.

The CPS application status inquiry program (C7392MQ1) will process the request message, access the database, format the reply message and return the result back to the originating source by way of the EAI bus.

3.1.2 CPS Test Scenario Detailed Design Description

The application to be used for the EAI CPS core test is the Application Status Query



- 1) A message consisting of SSN and NameId will be sent from the WebSphere Application Server (WAS) to the EAI bus. (Shown by 1 & 2 in diagram above)
- 2) In the MQSI message flow, the message will be transformed by modifying the MQSeries header. The MQSI message flow will also insert data into a database via the MQSI “DataInsert” node. (Shown by 3 in diagram above)
- 3) The message will then be sent to the CPS system where the CICS Bridge Monitor will query the message and start the CICS DPL task. (Shown in 5 & 6 in diagram above)
- 4) The CICS DPL task will read the message from the queue and call the Application Status Query application passing it the message as the parameter. The Application Status Query application will query the data from DB2 and return the response back to the CICS DPL task. (Shown by 7, 8 & 9 in diagram above)
- 5) The CICS DPL task will take the response data and write it on the queue to be sent back to the source system. (Shown by 10, 11 & 12 above)

Files

Following is a list of the data files used for test input. The data contains the program name to be executed by the CICS DPL bridge and the SSN and Name ID of the subject.

File Specification	Function
/www/dev/eai/input/cps/cpinp1.xml	Test input file residing on the SU35E5. Contains the data: C7392MQ1 265234899AX
/www/dev/eai/input/cps/cpinp2.xml	Test input file residing on the SU35E5. Contains the data: C7392MQ1 918264155WB
/www/dev/eai/input/cps/cpinp3.xml	Test input file residing on the SU35E5. Contains the data: C7392MQ1 215682344CP
/www/dev/eai/input/cps/cpinp4.xml	Test input file residing on the SU35E5. Contains the data: C7392MQ1 265312268TX
/www/dev/eai/input/cps/cpinp5.xml	Test input file residing on the SU35E5. Contains the data: C7392MQ1 556341287DC

Adapters

The MQSeries CICS Bridge Monitor and DPL Bridge are used on the CPS target system. These are out of the box MQSeries adapters and have been configured as part of the EAI Core Architecture implementation to validate the CICS Program execution.

To start the MQSeries CICS Bridge Monitor:

Log into the CPS CICS region and use the CICS Execution Command Interface (CECI) to START the MQSeries CICS Bridge Monitor CICS transaction “CKBR”. The following is the CECI command and options used:

START TR(CKBR) FR('Q=CPT1.CICSDEV2.BRIDGE.QUEUE,AUTH=LOCAL')

The MQSeries CICS Bridge Monitor

Inputs: This program queries the MQSeries queue that it is monitoring to determine its execution path.

Outputs: No output data is produced. It starts the DPL Bridge.

The MQSeries CICS DPL Bridge

Inputs: Message data from the queue. This data is application dependent. Maximum length of application data can be up to 32k.

Outputs: Application specific data is written to a MQSeries queue.

MQSI

The MQSI nodes and their function are documented below.

Node Name	Node Type	Function
CPS Applicant Request	MQInput	Read message from the input queue
Verify SSN	Filter	Verify SSN. If SSN = 'XXXXXXXXXX' goto false terminal If SSN <> 'XXXXXXXXXX' goto true terminal (<> means not equals)
Invalid DCN SSN	MQOutput	Reached from the failure terminal of the filter node if a failure is detected during computation. Message is written to the CORE.CPS.FAILURE queue.
Unkown filter error	MQOutput	Reached from the unknown terminal of the filter node if the expression evaluated is unknown. Message is written to the

Node Name	Node Type	Function
		CORE.CPS.UNKNOWN queue.
Build CPS Output Message	Compute	Take XML as input and create MRM. Input is taken from the true terminal of the filter node. This node appends required data fields to message. The out terminal outputs the transformed message and inserts data to a DB2 table using the datainsert1 node. The failure terminal is used to propagate the message if a failure is detected during computation.
Build Error Message	Compute	Reached via the failure terminal of the filter node. Build error message for invalid SSN. Out terminal goes to error reply. The failure terminal goes to trace1 node, which is used for debugging purposes.
DataInsert1	DataInsert	Insert the SSN to a DB2 table.
Error Reply	MQReply	Reached via the out terminal of the build error message node. Puts error message on reply to queue
CPS Queue	MQOutput	Reached via the out terminal of the build CPS output message. Puts message on queue for delivery to CPS
Trace on Input	Trace	Reached via the failure and catch terminal of the CPS Applicant Request node. The trace node is used for debugging purposes.
Trace1	Trace	Failure to build message goes to trace node. Trace node is used for debugging purposes.

3.1.3 CPS Test Scenario Dependencies

The following defines the dependencies and resource requirements for the sample CPS test.

- MQSeries running on logically referred to as SU35E5, SU35E16 or SU35E17, and CPT1.
- MQSI must be running on either logically referred to as SU35E16 or SU35E17.
- The DB2 database on CPS must be available.

Test Data

The test data must be in a specific format. From the initial entry on the WebSphere Server through MQSI and onto the adapter, each component is expecting the data a certain way. The data format is as follows:

The following table describes the Record layout required for input and output data for the CPS Application Status Query program.

Element Name	Start Position	End Position	Size	Input/Output Field	Permitted Value Override
Social Security Number	1	9	9	I/O	
Name Id	10	11	2	I/O	
Return Code	12	14	3	O	000 – Successful 100 - App not found on FE/W hold table 900 – Invalid Commarea length (must be 39)
Date Completed	15	22	8	O	Date – CCYYMMDD format
FEW Hold Flag	23	23	1	O	
DOB Prior	24	24	1	O	
Graduation Status	25	25	1	O	
Married Status	26	26	1	O	
Orphan	27	27	1	O	

Element Name	Start Position	End Position	Size	Input/Output Field	Permitted Value Override
Veteran	28	28	1	O	
Dependents	29	29	1	O	
Children	30	30	1	O	
Elec App Entry Src	31	31	1	O	
Misc Dates	32	40	8	O	Date – CCYYMMDD format

MQSeries Objects Used

Object Name	Object Type	Description
CPT1.DEAD.QUEUE	Local Queue	Local queue defined as the system dead letter queue. Message are placed on the queue when it is undeliverable.
CPT1.SU35E16	Channel	Sender Channel. Used to send messages to SU35E16.
SU35E16.CPT1	Channel	Receiver Channel. Used to receive messages from SU35E16.
SU35E16	Local Queue	Local queue defined as a transmission queue. Used when sending datafrom CPT1 to SU35E16.
CPT1.SU35E17	Channel	Sender Channel. Used to send messages to SU35E17.
SU35E17.CPT1	Channel	Receiver Channel. Used to receive messages from SU35E17.
SU35E17	Local Queue	Local queue defined as a transmission queue. Used when sending datafrom CPT1 to SU35E17.
CPT1.CICSDEV2.BRIDGE.QUEUE	Local Queue	Local queue used to receive data as input for the Application Status Query application..
SU35E5	Remote Queue	Remote queue used as a queue manager alias to be able to reroute the messages back to the SU35E5 server.

3.1.4 CPS Test Scenario Inputs

Following are 5 test input file contents for the CPS test scenario:

Input file cpinp1.xml

```
<?xml version = "1.0"?>
<cpsRoot>
  <ssn>100010109</ssn>
  <nameid>VP</nameid>
</cpsRoot>
```

Input file cpinp2.xml

```
<?xml version = "1.0"?>
<cpsRoot>
  <ssn>100010403</ssn>
  <nameid>TS</nameid>
</cpsRoot>
```

Input file cpinp3.xml

```
<?xml version = "1.0"?>
<cpsRoot>
```

```
<ssn>225400812</ssn>  
<nameid>ED</nameid>  
</cpsRoot>
```

Input file cpinp4.xml

```
<?xml version = "1.0"?>  
<cpsRoot>  
  <ssn>500902409</ssn>  
  <nameid> ED</nameid>  
</cpsRoot>
```

Input file cpinp5.xml

```
<?xml version = "1.0"?>  
<cpsRoot>  
  <ssn>555115021</ssn>  
  <nameid> FO</nameid>  
</cpsRoot>
```

3.1.5 CPS Test Scenario Expected Results

If the test scenario provides the output as detailed below or returns one of the return codes, then we can say the acceptance criteria has been met. If each record returned matches that exactly as shown below, then the expected results have been returned. Within each returned record are the social security number and nameid that were included in the input. Other results that may be returned are:

If the CPS application doesn't find the response information, the return code of 100 - App not found on FE/W hold table will be returned.

If the structure of the data is incorrect the return code of 900 - Invalid Commarea length (must be 39) will be returned

If the CPS application encounters any other errors the value of the SQL encountered will be returned.

If the CPS application can return a valid response the return code of 000 - Successful, data returned will be returned.

Response message definition

```
01 DFHCOMMAREA.  
  05 PARM-WEB-SSN PIC X(09).  
  05 PARM-WEB-NAME-ID PIC X(02).  
  05 PARM-RETURN-CODE PIC X(03).  
    88 SUCCESSFUL-LOOKUP VALUE '000'  
    88 RECORD-NOT-FOUND VALUE '100'  
    88 INVALID-CA-LENGTH VALUE '900'  
  05 PARM-DATE-COMPLETED PIC 9(08).  
  05 PARM-FEW-HOLD-FLAG PIC X(01).  
  05 PARM-DOB-PRIOR PIC X(01).  
  05 PARM-STAT-GRAD PIC X(01).  
  05 PARM-STAT-MARRIED PIC X(01).  
  05 PARM-ORPHAN PIC X(01).  
  05 PARM-VETERAN PIC X(01).  
  05 PARM-HAVE-DEP PIC X(01).  
  05 PARM-HAVE-CHILDREN PIC X(01).
```

05 PARM-ELEC-APP-ENTRY-SRC PIC X(01).
05 PARM-FEW-DATE-ADDED PIC 9(08).

Test Results using data file cpinp1.xml:

Web SSN: 100010109
Web Name ID: VP
Return Code: 000
Date Completed: 20001204
Hold Flag:
DOB Prior: 2
Graduation Status: 2
Marital Status: 1
Orphan Status: 2
Veteran Status: 1
Has Dependents: 2
Has Children: 2
Application Source: 7
Date Added: 20001204
100010109VP00020001204 2212122720001204

Test Results using data file cpinp2.xml:

Web SSN: 100010403
Web Name ID: TS
Return Code: 000
Date Completed: 20001204
Hold Flag: Y
DOB Prior: 2
Graduation Status: 2
Marital Status: 2
Orphan Status: 2
Veteran Status: 2
Has Dependents: 2
Has Children: 2
Application Source: 7
Date Added: 20001204
100010403TS00020001204Y2222222720001204

Test Results using data file cpinp3.xml

Web SSN: 225400812
Web Name ID: ED
Return Code: 000
Date Completed: 20010102
Hold Flag:
DOB Prior: 2
Graduation Status: 2
Marital Status: 1
Orphan Status: 2
Veteran Status: 2
Has Dependents: 2
Has Children: 2
Application Source: 7

Date Added: 20001019
225400812ED00020010102 2212222720001019

Test Results using data file cpinp4.xml

Web SSN: 500902409
Web Name ID: ED
Return Code: 000
Date Completed: 20010102
Hold Flag:
DOB Prior: 2
Graduation Status: 2
Marital Status: 2
Orphan Status: 2
Veteran Status: 1
Has Dependents: 2
Has Children: 2
Application Source: 7
Date Added: 20001019
500902409ED00020010102 2222122720001019

Test Results using data file cpinp5.xml

Web SSN: 555115021
Web Name ID: FO
Return Code: 000
Date Completed: 20001204
Hold Flag: Y
DOB Prior: 2
Graduation Status: 2
Marital Status: 2
Orphan Status: 2
Veteran Status: 2
Has Dependents: 2
Has Children: 2
Application Source: 8
Date Added: 20001204
555115021FO00020001204Y2222222820001204

- This CPS test scenario was executed by the Release 1 EAI Core team and the expected results were received and validated.

3.2 EAI Component Test for Direct Loan Servicing System (DLSS)

The DLSS system provides real-time and batch processing functionality on the Direct Loan Servicing System. The EAI Bus provides the capability to access real-time DLSS transactions as well as the processing of existing batch programs.

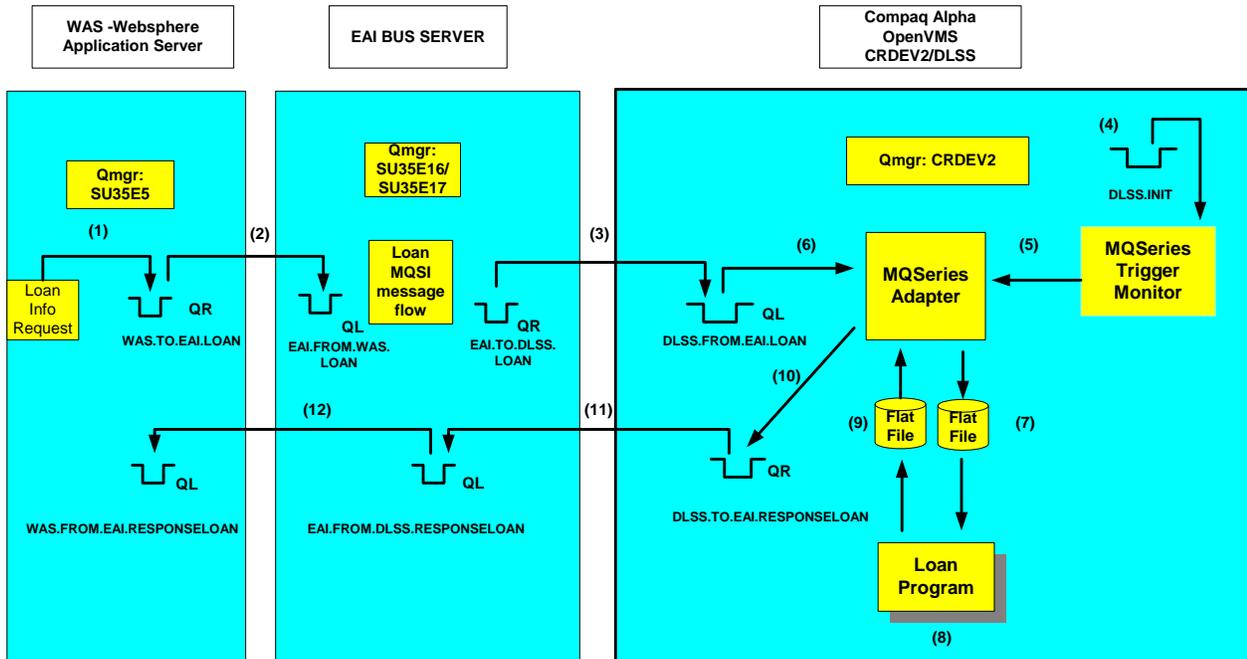
3.2.1 DLSS Test Scenario Description

This test will demonstrate the functionality of MQSeries messaging across disparate systems, the use of a MQSeries Integrator Message flow, and the use of MQSeries adapters written for DLSS.

The application to be used for the EAI DLSS core test is the loan payoff application. A message consisting of SSN will be sent from the WebSphere Application Server to the EAI bus. In the MQSI message flow, the message will be transformed by appending required records to the message. The message will then be sent to DLSS where an adapter will read from the queue, output the data to a flat file and then release the sequence of programs from the OpenVMS batch queue. The Loan payoff programs will process the file and create another file with the detail corresponding to each SSN initially sent to DLSS. Following the execution of the Loan payoff program, the adapter will be run to read from the file created by the loan payoff program and put the message(s) on the queue to be sent back to the Websphere Application Server. Once the batch jobs have been released, there is a five-minute waiting period for an output file to be generated. If no output file is found, an error message is returned to the Websphere Application Server.

3.2.2 DLSS Test Scenario Detailed Design Description

EAI DLSS Data Flow



The application to be used for the EAI DLSS core test is the loan payoff application.

1. A message consisting of SSN will be sent from the WebSphere Application Server (WAS) to the EAI bus. (Shown by 1 & 2 in diagram above.)
2. In the MQSI message flow, the message will be transformed by appending required records to the message (Shown by box in between (2) and (3) above.)
3. The message will then be sent to the DLSS system where an adapter will read from the queue and output the data to a flat file and then release the sequence of programs from the OpenVMS batch queue. (Shown by 3, 4, 5, 6 above.)
4. The Loan payoff application will process the file and in turn create another file with the detail corresponding to each SSN and Name initially sent to DLSS. (Shown by 7, 8, 9.)
5. Following the execution of the Loan payoff application the adapter will be run to read from the file created by the loan payoff application and put the message(s) on the queue to be sent back to the WebSphere Application Server (Shown by 10, 11, 12 above.).

Files

The table below contains the file specification and the function each file performs.

Device:[directory]filespec	Function
MQS1:[MQM]TRIG_CHANNEL.COM	Command file to to run the sender channel upon data arriving in the local queue
MQS1:[MQM]START_MQ_BATCH_JOBS.COM	Command file to be executed upon system startup
MQS1:[MQM]EAI.TST	Script file which contains definitions of the MQ objects
MQS_EXAMPLES:	Logical pointing to directory containing MQSeries sample programs.
MQS_INCLUDE	Logical pointing to directory containing Include files used in sample programs
TEST39:[TEST_39]MQLOAN.COM	Command procedure triggered upon data hitting the input queue. Receives file, releases batch jobs, sends file
DUA10:[TEST_39.STACEY]MQPUT.LOG	Log file of any errors on putting data to the queue
DUA10:[TEST_39.STACEY]MQGET.LOG	Log file of any errors on retrieving data from the queue.
DUA10:[TEST_39.STACEY]NOFILE.DAT	File returned to initiating system if no records were generated by the DLSS application within a 5 minute period.
DUA10:[TEST_39.STACEY]CI001S1.FDL	FDL file used to convert file from stream_lf to variable length, maximum of 600 byte file.
DUA10:[TEST_39.STACEY]REPLYTO.DAT	File which is used by both adapters. Contains the replyto queue manager and the replyto queue.
CIS_X_INPUT:CI001S1.INP	File spec of the input file for the DLSS application
CIS_X_XFER:CI024S1.DAT	Filespec of the output file created by the DLSS application
DUA10:[TEST_39]SCHED_DAILY_INTERFACE_MQSERIES.COM	File used to resubmit batch jobs.
CIS_LOG_D	Directory which contains log files created by the batch jobs.
MQS1:[MQM]	Directory containing MQSeries system files
MQS2:[MQM]	Directory containing MQSeries log files
MQS1:[MQM]EAI.TST	File containing the SFA MQSeries object definitions

Adapters

There are 2 adapters on the DLSS system. MQGET.C and MQPUT.C are C programs. The MQGET.C adapter reads messages from a MQSeries queue and writes the message to a file. The MQPUT.C adapter reads data from a file and puts the data as a message to a MQSeries queue.

To run the programs on the DLSS system:

\$MQPUT filename

where filename is the name of a file.

\$MQGET queuefilename filename

where queuefilename is a valid MQSeries queue

where filename is the name of a file

MQGET.C Adapter

Inputs: The program expects 2 parameters as input:

- (1) MQSeries queue to read the message(s) from.

Type of input parameter: MQCHAR48

Maximum length parameter: 48

Purpose of input parameter is to specify the queue to read messages from.

(2) Filename to write data to

Type of input parameter: char[500] – character array of size 500.

Maximum length of filename is 500 characters

Purpose of input parameter is to specify the file specification to write the message to.

Outputs: A file is generated as output. The filename is specified as input to the program.

MQPUT.C Adapter

Inputs: The program expects 1 parameter as input (1) filename to read data from.

Type of input parameter char[500] – character array of size 500.

Maximum length of filename is 500 characters

Purpose of input parameter: Tells the adapter the filespec to read data from.

Outputs: Message is written to a MQSeries queue.

To compile the MQGET.C program: `$cc / include_directory=mqs_include mqget.c`

To link the program: `$link mqget.obj,sys$input/options <ENTER>`
`sys$share:mqm/shareable`

To compile the MQPUT.C program: `$cc / include_directory=mqs_include mqput.c`

To link the program: `$link mqput.obj,sys$input/options <ENTER>`
`sys$share:mqm/shareable`

MQSI

The MQSI nodes and their function are documented below.

Node Name	Node Type	Function
DLSS Loan Input	MQInput	Read message from input queue
Verify DCN SSN	Filter	Verify SSN. If SSN = 'XXXXXXXXXX' goto false terminal If SSN <> 'XXXXXXXXXX' goto true terminal (<> means not equals)
Invalid DCN SSN	MQOutput	Reached from the failure terminal of the filter node if a failure is detected during computation. Message is written to the flow2.output queue.
Unkown filter error	MQOutput	Reached from the unknown terminal of the filter node if the expression evaluated is unknown. Message is written to the flow2.unkown queue.
Build DLSS Output Message	Compute	Take XML as input and create MRM. Input is taken from the true terminal of the filter node. This node appends required data fields to message. The out terminal outputs the transformed message. The failure terminal is used to propagate the message if a failure is detected during computation.

Build Error Message	Compute	Reached via the failure terminal of the Filter Node. Build error message for invalid SSN. Out terminal goes to error reply node. Failure terminal goes to trace1 node which is used for debugging purposes.
Error Reply	MQReply	Reached via the out terminal of the build error message node. Puts error message on reply to queue
DLSS Queue	MQOutput	Reached via the out terminal of the build DLSS output message. Puts message on queue for delivery to DLSS
Trace on Input	Trace	Reached via the failure and catch terminal of the DLSS loan input node. The trace node is used for debugging purposes.
Trace on Build	Trace	Failure to build message goes to trace node. Trace node is used for debugging purposes.

3.2.3 DLSS Test Scenario Dependencies

The following defines the dependencies and resource requirements for a successful DLSS test.

- MQSeries running on logically referred to as SU35E5, SU35E16 or SU35E17, and DLSS.
- MQSI must be running on either logically referred to as SU35E16 or SU35E17.
- Oracle Rdb database on DLSS must be available before test can be run.
- The TEST_39 test environment on DLSS.

Test Data

The DLSS user interface on the WebSphere Application Server accepts a social security number as input. The MQSI message flow prepends and appends data to the social security number. The DLSS Loan Application is expecting as input a record of the following format (shown below in the table). Each record type has a specific meaning related to it as described below.

- BAA - batch header record
- DAA - D transaction type header record
- DCN - Payoff Request
- DZZ - D transaction type trailer
- TZZ - Batch trailer record

The table below details the element names and other relevant data for each DLSS record.

BAA Record Type Element Names	Start Position	End Position	Size	Input Output Field	Permitted Value Override
Batch Identifier	1	1	1	I/O	D - Non-Financial Transaction
Transaction Type	2	3	2	I/O	CN
Transaction Sequence Number	4	7	4	I/O	This starts at 1 and is incremented for each DCN in the file
Loan ID	9	29	21	O	
Social Security Number	30	38	9	I/O	
Effective Date	39	46	8	I/O	

BAA Record Type Element Names	Start Position	End Position	Size	Input Output Field	Permitted Value Override
Daily Accrual Amount	47	57	11	O	
Payoff Amount	58	68	11	O	
Principal Balance Amount	69	79	11	O	
Interest Balance Amount	80	90	11	O	
Charges Balance Amount	91	101	11	O	
Fees Balance Amount	102	112	11	O	
Academic Completion Date	113	120	8	O	
Enrollment Status	121	121	1	O	
Effective Date	122	129	8	O	
Loan Status	130	131	2	O	
Change Date	132	139	8	O	
Interest Rate	140	145	6	O	
Interest Rate Category	146	146	1	O	
Institution OPE Number	147	154	8	O	
Loan Type Code	155	155	1	O	1 - Direct, Subsidized 2 - Direct, Unsubsidized 4 - Direct, PLUS 5 - Consolidation, Subsidized 6 - Consolidation, Unsubsidized 7 - Consolidation, PLUS
Application Receipt Date	156	163	8	O	
Incentive Indicator	164	164	1	I	Not yet in use
Current Repayment Plan	165	166	2	I	

DAA Record Type Element Names	Start Position	End Position	Size	Source	Permitted Value Override
Batch Identifier	1	1	1	I/O	B - Transmission Header
Transaction Type	2	3	2	I/O	AA
Reserved	4	8	5	I/O	Spaces
File Create Date	9	16	8	I/O	MMDDYYYY -
File Create Time	17	24	8	I/O	HHMMsscc
Interface ID	25	30	6	I/O	LO0101
Control Count	31	39	9	I/O	Spaces
Reserved	40	55	16	I/O	Spaces
Interface ID	56	61	6	I/O	Spaces
Reserved	62	103	42	I/O	Spaces

DCN Record Type Element Names	Start Position	End Pos	Size	Source	Permitted Value Override
Batch Identifier	1	1	1	I/O	D - Data Type Header
Transaction Type	2	3	2	I/O	AA
Reserved	4	8	5	I/O	Spaces
File Create Date	9	16	8	I/O	MMDDYYYY

DCN Record Type Element Names	Start Position	End Pos	Size	Source	Permitted Value Override
File Create Time	17	24	8	I/O	HHMMsscc
Batch Number	25	28	4	I/O	May use any number as long as it matches the DZZ
Control Count	29	37	9	I/O	Spaces
Reserved	38	103	66	I/O	Spaces

DZZ Record Type Element Names	Start Position	End Position	Size	Source	Permitted Value Override
Batch Identifier	1	1	1	I/O	D - Data Type Header
Transaction Type	2	3	2	I/O	ZZ
Reserved	4	8	5	I/O	Spaces
File Create Date	9	16	8	I/O	MMDDYYYY
File Create Time	17	24	8	I/O	HHMMsscc
Batch Number	25	28	4	I/O	Must match DAA
Control Count	29	37	9	I/O	Number of DCN records
Reserved	38	103	66	I/O	Spaces

TZZ Record Type Element Names	Start Position	End Position	Size	Source	Permitted Value Override
Batch Identifier	1	1	1	I/O	T - Transmission Trailer
Transaction Type	2	3	2	I/O	ZZ
Reserved	4	8	5	I/O	Spaces
File Create Date	9	16	8	I/O	MMDDYYYY
File Create Time	17	24	8	I/O	HHMMsscc
Interface ID	25	30	6	I/O	Input - LO0101
Control Count	31	39	9	I/O	Total records in file not including BAA & TZZ records but includes all other AA & ZZ records
Reserved	40	55	16	I/O	Spaces
Interface ID	56	61	6	I/O	Spaces
Reserved	62	103	42	I/O	Spaces

MQSeries Objects Used

Object Name	Object Type	Description
SU35E16	Local Queue	Local queue defined as transmit queue. Used when sending data from CRDEV2 to SU35E16.
SU35E17	Local Queue	Local queue defined as transmit queue. Used when sending data from CRDEV2 to SU35E17.
CRDEV2.DEAD.LETTER.QUEUE	Local Queue	Local queue defined as the system dead letter queue. Message is placed on the queue when it is undeliverable.
DLSS.INIT	Local Queue	Queue used as an initiation queue for the Loan application

- The generation of a report with output data as shown above or a message stating no output was available.
- The display of the report results via the browser is identical to the DLSS information retrieved directly from the Oracle Rdb database as shown in the expected results above.

The expected results flow is as follows. The input record passed to the DLSS loan application is a request for payoff information. The loan application will create an output file consisting of payoff balances. If a file does not appear after 5 minutes of initiating the programs, a message of “no data” is passed back to the front-end. Otherwise, the expected data results are passed back to the calling application.

Output file layout with social security number 252494544

DCN0001
252494544S96G0154410125249454405262001000000000720000033194200000322827000000
091150000000000000000000000009171997 02252001RP02252001082500T001544001

DCN0002
252494544S97G0154410125249454405262001000000000720000033194400000322829000000
091150000000000000000000000009171997 02252001RP02252001082500T001544001

DCN0003
252494544S98G0154410125249454405262001000000000240000011064300000107605000000
030380000000000000000000000009171997 02252001RP02252001082500T001544001

Output file layout with social security number 402273152

DCN0001
402273152S99G8888700140227315204302001000000001330000059647000000595402000000
010680000000000000000000000003311999 08052000RP08052000081900T088887005

Output file layout with social security number 366821582

DCN0001
366821582S00G0232700136682158205202001000000000360000016469900000161264000000
034350000000000000000000000006212000 03182001RP03182001081900T002327001

DCN0002
366821582S95G0232710136682158205202001000000000960000043609600000426935000000
091610000000000000000000000004281995 03182001RP03182001082500T002327001

DCN0003
366821582S98G0232700236682158205202001000000000100000005317500000052066000000
011090000000000000000000000008181998 03182001RP03182001081900T002327001

DCN0004
366821582S99G0232700136682158205202001000000000440000020429600000200036000000
042600000000000000000000000006212000 03182001RP03182001081900T002327001

DCN0005
366821582U00G023270013668215820520200100000000090000004428400000043359000000
009250000000000000000000000006212000 03182001RP03182001081900T002327002

- This DLSS test scenario was executed by the Release 1 EAI Core team and the expected results were received and validated.

3.3 EAI Component Test for Electronic Campus Based System (eCBS) - Batch

The electronic Campus Based System contains data records about campus based programs. It is an application that runs within the WebSphere Application Server. The EAI batch enablement of the eCBS system will provide the capability to send and receive batch files to and from eCBS.

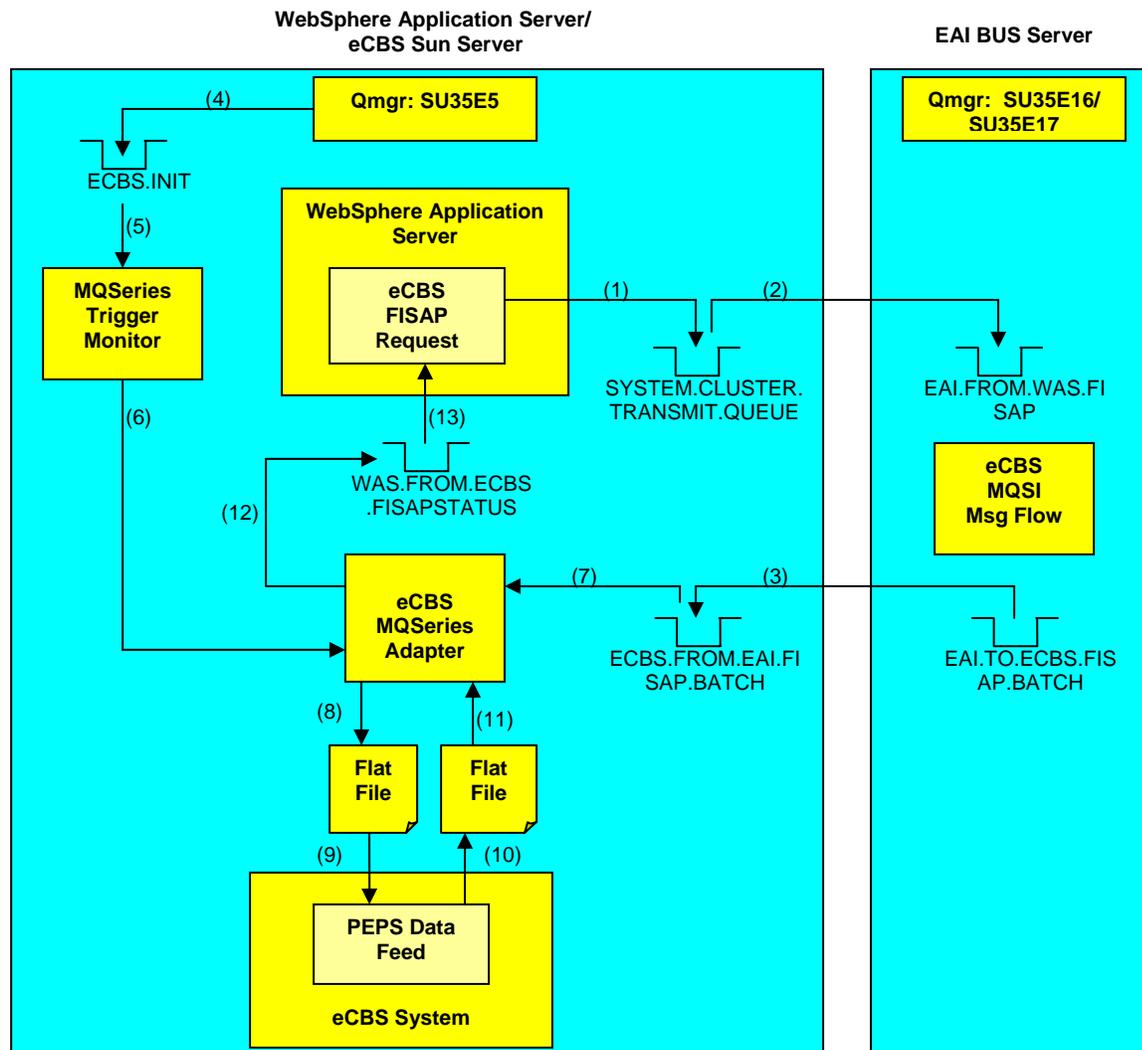
3.3.1 eCBS – Batch Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure to the eCBS system is based on a request for FISAP status. The eCBS development team provided a shell script that runs every hour that retrieves FISAP status based on PEPS data. A MQSeries Java adapter has been developed to connect the eCBS system to the EAI Bus.

3.3.2 eCBS – Batch Test Scenario Detailed Design Description

The sample function selected for the eCBS system validates the ability of a user to send PEPS data from the test application. A XML string that contains the PEPS data and a flag specifying batch processing, is placed into a MQSeries message that is sent to the EAI Bus for transformation by MQSI. MQSI performs the defined message flow transformation and routes the transformed message data to the target system, eCBS. Upon receipt by the eCBS server, the custom adapter retrieves the message from the queue, creates a file for input to eCBS that contains PEPS data and waits for an output file that contains FISAP status from eCBS. The results are formatted in XML and placed in a MQSeries message to be sent back to the test application for display.

The figure below describes the message flow for the batch eCBS custom adapter.



The flow of a MQSeries Request type message through the EAI eCBS batch design is as follows:

- 1) A MQSeries Request type message is placed on the SYSTEM.CLUSTER.TRANSMIT.QUEUE, that is destined for a cluster queue located on the EAI Bus (SU35E16 or SU35E17).
- 2) The MQSeries Queue Manager (SU35E5) on the WAS moves the message to the cluster queue EAI.FROM.WAS.FISAP. The message is pulled from the EAI.FROM.WAS.FISAP and processed through the eCBS MQSI Message Flow. The eCBS MQSI Message Flow will determine whether the data should be processed as a real-time transaction or as a batch process by eCBS.
- 3) Messages that will be processed by eCBS as a real-time transaction is put to the Remote Queue EAI.TO.ECBS.FISAP.REAL by the eCBS MQSI Message Flow. The MQSeries Queue Manager (SU35E16 or SU35E17) on the EAI Bus server moves the message to the Local Queue ECBS.FROM.EAI.FISAP.REAL and based on the queue attributes, the MQSeries Queue Manager (SU35E5) puts a trigger message on the ECBS.INIT queue.
- 4) Messages that will be processed by eCBS as a batch process is put to the Remote Queue EAI.TO.ECBS.FISAP.BATCH by the eCBS MQSI Message Flow. The MQSeries Queue Manager (SU35E16 or SU35E17) on the EAI Bus server moves the message to the Local Queue

- ECBS.FROM.EAI.FISAP.BATCH and based on the queue attributes, the MQSeries Queue Manager (SU35E5) puts a trigger message on the ECBS.INIT queue.
- 5) The MQSeries Trigger Monitor application pulls the trigger message off of the ECBS.INIT queue.
 - 6) The MQSeries Trigger Monitor application starts the eCBS Adapter.
 - 7) The eCBS Adapter reads the message(s) from the ECBS.FROM.EAI.FISAP.BATCH queue.
 - 8) The eCBS Adapter stores the message(s) into a flat file.
 - 9) eCBS monitors a directory for files created by the eCBS Adapter and then processes that data.
 - 10) When a response is to be sent back from eCBS, eCBS will store that information into another file in a directory which the eCBS Adapter monitors.
 - 11) The eCBS Adapter monitors a directory for files created by eCBS and then reads in the data from the file.
 - 12) The eCBS Adapter creates an MQSeries message based on the data in the file and places it onto the WAS.FROM.ECBS.FISAPSATUS Local queue on SU35E5.
 - 13) The MQSeries Reply message containing any eCBS response is retrieved by WAS.

Files

The following input files have been defined on the WAS server for access by the EAI Core test application in execution of this test scenario:

File Specification	Function
batchRequestData.txt	Valid PEPS data that returns successfully
batchErrorData1.txt	Invalid PEPS data that returns an error from MQSI
batchErrorData2.txt	Invalid PEPS data that returns an error from the eCBS custom adapter
batchMQSIOutputData.txt	Results from the successful processing of batchRequestData.txt by MQSI. This is the data that is sent to the eCBS custom adapter.
batchResponseData.txt	Results from the successful processing of batchMQSIOutputData.txt by the custom adapter. This is the data that is sent back to the test application.
batchErrorMQSIOutputData1.txt	Results from the erroneous processing of batchErrorData1.txt by MQSI. This is the data that is sent back to the test application.
batchErrorMQSIOutputData2.txt	Results from the successful processing of batchErrorData2.txt by MQSI. This is data that is sent to the eCBS custom adapter.
batchErrorResponseData2.txt	Results from the erroneous processing of batchErrorMQSIOutputData2.txt by the eCBS custom adapter. This is the data sent back to the test application.
SU35E5.msg	A file used to hold messages that cannot be sent to MQSeries, in the event MQSeries is not operational.
batchUnexpectedErrorResponseData2.txt	Results from the erroneous processing of batchMQSIOutputData.txt by the eCBS custom adapter. This is the data sent back to the test application.

Adapters

A custom MQ Adapter and a script file were developed for the eCBS system. The custom adapter, called MQECBS, is written in Java. The script file, called ECBSAdapter, is used to set environment variables and to execute the adapter.

- To start the adapter use the following command:
 ECBSAdapter batch

MQSI

The MQSI nodes and their function are documented below. The same MQSI message flow is used for both the batch and real-time adapter.

Node	Type	Description/Function
Input Message Queue From WAS	MQInput	Gets message from queue EAL.FROM.WAS.FISAP
Trace1	Trace	Traces input message
Determine If Batch Request Type	Filter	Checks if requesttype = 1 If True then Build eCBS Batch Request If False goto Build eCBS Real-Time Request
Build eCBS Batch Request	Compute	Builds an output message for eCBS batch requests
Build eCBS Real-Time Request	Compute	Builds an output message for eCBS real-time requests
Build Error Message	Compute	Builds an error message
Send Error Back To Requestor	MQReply	Sends a message back to the requestor based on the reply to information
Trace2	Trace	Traces output message
Output to Batch FMS Queue	MQOutput	Puts message to queue EAL.TO.FMS.FISSAP.BATCH
Output to Real-Time FMS Queue	MQOutput	Puts message to queue EAL.TO.FMS.FISSAP.REAL

3.3.3 eCBS – Batch Test Scenario Dependencies

To execute the eCBS System test scenario the following dependencies must be met:

- MQSeries Messaging and queue managers on each of the following systems are operational, logically referred to as SU35E5 and SU35E16 or SU35E17.
- MQSI must be operational on either of the following systems, logically referred to as SU35E16 or SU35E17.
- The adapter has access to read and write files on the eCBS system.
- The eCBS shell script that will process the batch file has been created and is operational.

3.3.4 eCBS –Batch Test Scenario Inputs

From the initial entry on the WebSphere Application Server through MQSI and onto the adapter, each component is expecting the data a certain format. To validate each component, 4 test cycles were used for testing: Normal testing, expected error testing, unexpected error testing, and regression testing. The results of each test cycle are shown in Section 3.3.5.

3.3.4.1 Normal Test Cycle

Test Description

The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. The data format sent to MQSI is as follows.

Test application output and MQSI input: batchRequestData.txt

```
<eCBSRequest>
  <Type>1</Type>
  <Transaction>0000000000P200109132001.00038
  0100103000NBishop State Community College           Bishop State
  Community College           925 Dauphin Island Parkway
  Mobile           AL097           366053299
  YCY051219970606199710231998032520010723051QH2 0401   8222990206   19660216
  NN04
  0100102400NUniversity of West Alabama           University of West
  Alabama           205 North Washington
```

```
Livingston      AL119      354700000
YCY051220010502200105072001052320070331081QH5 0407      19651219
NN04
</Transaction>
</eCBSRequest>
```

3.3.4.2 Expected Error Test Cycle

Test Description

Two sets of test data were created to generate expected errors. The first data format generates an error in MQSI and the second data format generates an error in the custom adapter. Both data formats are as follows:

Test application output and MQSI input: batchErrorData1.txt

```
<eCBSRequest>
  <Type>1</Type>
</eCBSRequest>
```

Test application output and MQSI input: batchErrorData2.txt

```
<eCBSRequest>
  <Type>1</Type>
  <Transaction>PEPS Data </Transaction>
</eCBSRequest>
```

3.3.4.3 Unexpected Error Test Cycle

Test Description

The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. While data is being processed by the custom adapter, two unexpected errors are simulated. The first simulates a MQSeries failure that makes all MQSeries objects unavailable after the data has been processed within eCBS but before a response has been sent to the source application, and the second simulates an eCBS failure that does not provide a response back to the adapter. The data format sent to MQSI is as follows:

Test application output and MQSI input: batchRequestData.txt

```
<eCBSRequest>
  <Type>1</Type>
  <Transaction>0000000000P200109132001.00038
0100103000NBishop State Community College      Bishop State
Community College      925 Dauphin Island Parkway
Mobile      AL097      366053299
YCY051219970606199710231998032520010723051QH2 0401      8222990206      19660216
NN04
0100102400NUniversity of West Alabama      University of West
Alabama      205 North Washington
Livingston      AL119      354700000
YCY051220010502200105072001052320070331081QH5 0407      19651219
NN04
```

</Transaction>
</eCBSRequest>

3.3.4.4 Regression Test Cycle

Test Description

This test re-executes the test executed in the normal test cycle and expected error test cycle. The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. The data format sent to MQSI is as follows:

Input file: batchRequestData.txt

```
<eCBSRequest>
  <Type>1</Type>
  <Transaction>0000000000P200109132001.00038
  0100103000NBishop State Community College           Bishop State
  Community College           925 Dauphin Island Parkway
  Mobile           AL097           366053299
  YCY051219970606199710231998032520010723051QH2 0401   8222990206   19660216
  NN04
  0100102400NUniversity of West Alabama           University of West
  Alabama           205 North Washington
  Livingston           AL119           354700000
  YCY051220010502200105072001052320070331081QH5 0407   19651219
  NN04
  </Transaction>
</eCBSRequest>
```

3.3.5 eCBS – Batch Test Scenario Expected Results

The execution of the test scenario returns an XML document that contains the result from MQSI and/or the custom adapter.

3.3.5.1 Expected Results for Normal Test Cycle

The results from successfully processing by MQSI and the custom adapter are listed below.

MQSI Output sent to eCBS custom adapter as input: batchMQSIOutputData.txt

```
<eCBSAdapterRequest>
  /www/dev/cbs/EAI/indata/PEPS.DAT;0000000000P200109132001.00038
  0100100304NFaulkner University           Faulkner University - Mobile
  Center           808 Western America Drive           Mobile
  AL           366094100           YCY           082SH5 04
  19681202   NN04
  0100100400NUniversity of Montevallo           University of Montevallo
  Palmer Circle           Montevallo           AL117
  351156000           YCY051220000621200007202000081020060930081SH5 0406
  8221010106   19651201   NN04
  ;/www/dev/cbs/EAI/outdata/STATUS.DAT;
</eCBSAdapterRequest>
```

Custom adapter output sent back to the test application: batchResponseData.txt

```
<eCBSAdapterResponse>
  <School>
    <OPEID>00103000</OPEID>
    <Year>2001</Year>
    <NewSchoolSW>N</NewSchoolSW>
    <WorkCollegeSW>N</WorkCollegeSW>
    <TradCalendarSW>Y</TradCalendarSW>
    <MultiCampusSW>N</MultiCampusSW>
    <FISAPStatusInd>1</FISAPStatusInd>
  </School>
  <School>
    <OPEID>00102400</OPEID>
    <Year>2001</Year>
    <NewSchoolSW>N</NewSchoolSW>
    <WorkCollegeSW>N</WorkCollegeSW>
    <TradCalendarSW>Y</TradCalendarSW>
    <MultiCampusSW>N</MultiCampusSW>
    <FISAPStatusInd>1</FISAPStatusInd>
  </School>
</eCBSAdapterResponse>
```

3.3.5.2 Expected Results for Expected Error Test Cycle

The results of invalid input data by MQSI is listed below.

MQSI Output sent to the test application: batchErrorMQSIOutputData1.txt

```
<eCBSAdapterResponse>
  An Error Occurred In The ECBS Message Flow
</eCBSAdapterResponse>
```

The results of valid input data sent to MQSI but invalid data being sent to the custom adapter is listed below.

MQSI output sent to custom adapter as input: batchErrorMQSIOutputData2.txt

```
<eCBSAdapterRequest>
  /www/dev/cbs/EAI/indata/PEPS.DAT;PEPS Data;/www/dev/cbs/EAI/outdata/STATUS.DAT;
</eCBSAdapterRequest>
```

Custom adapter output sent back to the test application: batchErrorResponseData2.txt

```
<eCBSAdapterResponse></eCBSAdapterResponse>
```

3.3.5.3 Expected Results for Unexpected Error Test Cycle

The results from successfully processing by MQSI and unsuccessfully processing by the custom adapter are listed below. The unsuccessful processing is caused by a MQSeries failure that makes all MQSeries objects unavailable. Since a response has been sent back from eCBS to the adapter, but the adapter

cannot send a response back to the source application because MQSeries is not operational, messages are written to a log file. This removes the problem of repeat processing of the same transaction and loss of unprocessed MQSeries messages. Messages found in the log file can be resent once the MQSeries is operational.

MQSI Output sent to eCBS custom adapter as input: batchMQSIOutputData.txt

```
<eCBSAdapterRequest>
  /www/dev/cbs/EAI/indata/PEPS.DAT;0000000000P200109132001.00038
  0100100304NFaulkner University                               Faulkner University - Mobile
  Center                               808 Western America Drive           Mobile
  AL                                   366094100                               YCY                               082SH5 04
  19681202   NN04
  0100100400NUniversity of Montevallo                       University of Montevallo
  Palmer Circle                               Montevallo           AL117
  351156000                               YCY051220000621200007202000081020060930081SH5 0406
  8221010106   19651201   NN04
  ;/www/dev/cbs/EAI/outdata/STATUS.DAT;
</eCBSAdapterRequest>
```

Log file used to store messages temporarily until MQSeries is operational: SU35E5.msg

```
<eCBSAdapterResponse>
  <School>
    <OPEID>00103000</OPEID>
    <Year>2001</Year>
    <NewSchoolSW>N</NewSchoolSW>
    <WorkCollegeSW>N</WorkCollegeSW>
    <TradCalendarSW>Y</TradCalendarSW>
    <MultiCampusSW>N</MultiCampusSW>
    <FISAPStatusInd>1</FISAPStatusInd>
  </School>
  <School>
    <OPEID>00102400</OPEID>
    <Year>2001</Year>
    <NewSchoolSW>N</NewSchoolSW>
    <WorkCollegeSW>N</WorkCollegeSW>
    <TradCalendarSW>Y</TradCalendarSW>
    <MultiCampusSW>N</MultiCampusSW>
    <FISAPStatusInd>1</FISAPStatusInd>
  </School>
</eCBSAdapterResponse>      EOM
```

Custom adapter output sent back to the test application: batchResponseData.txt

```
<eCBSAdapterResponse>
  <School>
    <OPEID>00103000</OPEID>
    <Year>2001</Year>
    <NewSchoolSW>N</NewSchoolSW>
    <WorkCollegeSW>N</WorkCollegeSW>
    <TradCalendarSW>Y</TradCalendarSW>
```

```
<MultiCampusSW>N</MultiCampusSW>
<FISAPStatusInd>1</FISAPStatusInd>
</School>
<School>
  <OPEID>00102400</OPEID>
  <Year>2001</Year>
  <NewSchoolSW>N</NewSchoolSW>
  <WorkCollegeSW>N</WorkCollegeSW>
  <TradCalendarSW>Y</TradCalendarSW>
  <MultiCampusSW>N</MultiCampusSW>
  <FISAPStatusInd>1</FISAPStatusInd>
</School>
</eCBSAdapterResponse>
```

The results from successfully processing by MQSI and unsuccessfully processing by the custom adapter are listed below. The unsuccessful processing is caused by a simulated eCBS failure, that does not provide a response back to the custom adapter.

MQSI Output sent to eCBS custom adapter as input: batchMQSIOutputData.txt

```
<eCBSAdapterRequest>
/www/dev/cbs/EAI/indata/PEPS.DAT;0000000000P200109132001.00038
0100100304NFaulkner University Faulkner University - Mobile
Center 808 Western America Drive Mobile
AL 366094100 YCY 082SH5 04
19681202 NN04
0100100400NUniversity of Montevallo University of Montevallo
Palmer Circle Montevallo AL117
351156000 YCY051220000621200007202000081020060930081SH5 0406
8221010106 19651201 NN04
;/www/dev/cbs/EAI/outdata/STATUS.DAT;
</eCBSAdapterRequest>
```

Custom adapter output sent back to the test application: batchUnexpectedErrorResponseData2.txt

```
<eCBSAdapterResponse>
Adapter Timeout - No Response From eCBS. Data Sent Successfully to eCBS. There Is A
Problem With eCBS Sending A Response Back To The Adapter.
</eCBSAdapterResponse>
```

3.3.5.4 Expected Results for Regression Test Cycle

The results from successfully processing by MQSI and the custom adapter are listed below.

MQSI Output sent to eCBS custom adapter as input: batchMQSIOutputData.txt

```
<eCBSAdapterRequest>
/www/dev/cbs/EAI/indata/PEPS.DAT;0000000000P200109132001.00038
0100100304NFaulkner University Faulkner University - Mobile
Center 808 Western America Drive Mobile
AL 366094100 YCY 082SH5 04
19681202 NN04
0100100400NUniversity of Montevallo University of Montevallo
Palmer Circle Montevallo AL117
351156000 YCY051220000621200007202000081020060930081SH5 0406
8221010106 19651201 NN04
;/www/dev/cbs/EAI/outdata/STATUS.DAT;
</eCBSAdapterRequest>
```

```
AL          366094100          YCY          082SH5 04
19681202   NN04
0100100400NUniversity of Montevallo          University of Montevallo
Palmer Circle          Montevallo          AL117
351156000          YCY051220000621200007202000081020060930081SH5 0406
8221010106   19651201   NN04
;/www/dev/cbs/EAI/outdata/STATUS.DAT;
</eCBSAdapterRequest>
```

Custom adapter output sent back to the test application: batchResponseData.txt

```
<eCBSAdapterResponse>
  <School>
    <OPEID>00103000</OPEID>
    <Year>2001</Year>
    <NewSchoolSW>N</NewSchoolSW>
    <WorkCollegeSW>N</WorkCollegeSW>
    <TradCalendarSW>Y</TradCalendarSW>
    <MultiCampusSW>N</MultiCampusSW>
    <FISAPStatusInd>1</FISAPStatusInd>
  </School>
  <School>
    <OPEID>00102400</OPEID>
    <Year>2001</Year>
    <NewSchoolSW>N</NewSchoolSW>
    <WorkCollegeSW>N</WorkCollegeSW>
    <TradCalendarSW>Y</TradCalendarSW>
    <MultiCampusSW>N</MultiCampusSW>
    <FISAPStatusInd>1</FISAPStatusInd>
  </School>
</eCBSAdapterResponse>
```

- This eCBS batch test scenario was executed by the Release 2 EAI Core team and the expected results were received and validated.

3.4 EAI Component Test for Electronic Campus Based System (eCBS) – Real Time

The electronic Campus Based System contains data records about campus based programs. It is an application that runs within the WebSphere Application Server. The EAI batch enablement of the eCBS system will provide the capability to execute a stored procedure in eCBS.

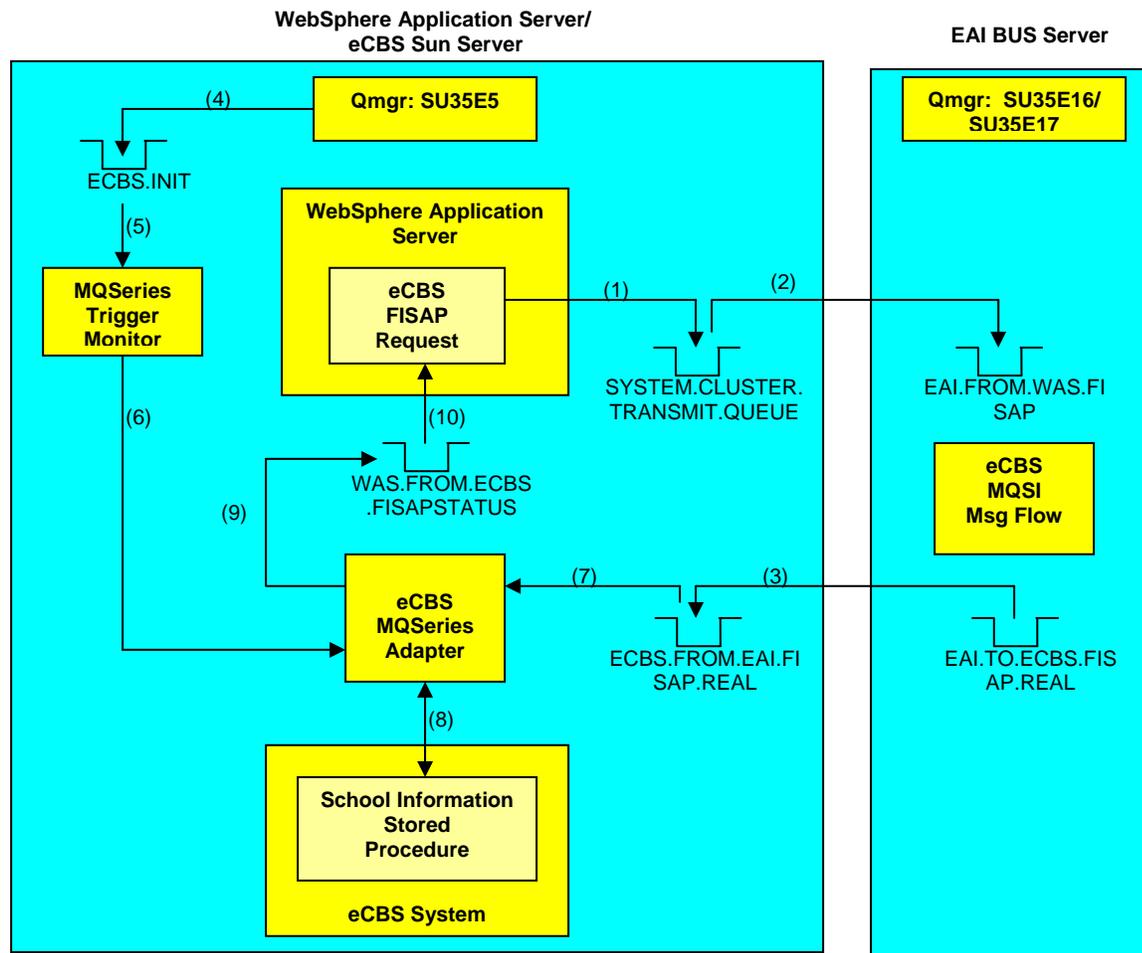
3.4.1 eCBS – Real Time Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure to the eCBS system is based on a request for FISAP status. The eCBS development team provided a stored procedure that runs in the eCBS database. A MQSeries Java adapter has been developed to connect the eCBS system to the EAI Bus and provides the capability to execute the stored procedure.

3.4.2 eCBS – Real Time Test Scenario Detailed Design Description

The sample function selected for the eCBS system validates the ability of a user to send an OPEID and application year from the test application. A XML string that contains the OPEID, application year, and a flag specifying real-time processing is placed into a MQSeries message that is sent to the EAI Bus for transformation by MQSI. MQSI performs the defined message flow transformation and routes the transformed message data to the target system, eCBS. Upon receipt by the eCBS server, the custom adapter retrieves the message from the queue, and invokes a stored procedure. The results from the stored procedure are formatted in XML and placed in a MQSeries message that is sent back to the test application for display.

The figure below describes the message flow for the real-time eCBS custom adapter.



The flow of a MQSeries Request type message through the EAI eCBS real-time design is as follows:

- 1) A MQSeries Request type message is placed on the SYSTEM.CLUSTER.TRANSMIT.QUEUE, that is destined for a cluster queue located on the EAI Bus (SU35E16 or SU35E17).
- 2) The MQSeries Queue Manager (SU35E5) on the WAS moves the message to the cluster queue EAI.FROM.WAS.FI SAP. The message is pulled from the EAI.FROM.WAS.FI SAP and processed through the eCBS MQSI Message Flow. The eCBS MQSI Message Flow will determine whether the data should be processed as a real-time transaction or as a batch process by eCBS.
- 3) Messages that will be processed by eCBS as a real-time transaction is put to the Remote Queue EAI.TO.ECBS.FI SAP.REAL by the eCBS MQSI Message Flow. The MQSeries Queue Manager (SU35E16 or SU35E17) on the EAI Bus server moves the message to the Local Queue ECBS.FROM.EAI.FI SAP.REAL and based on the queue attributes, the MQSeries Queue Manager (SU35E5) puts a trigger message on the ECBS.INIT queue.
- 4) Messages that will be processed by eCBS as a batch process is put to the Remote Queue EAI.TO.ECBS.FI SAP.BATCH by the eCBS MQSI Message Flow. The MQSeries Queue Manager (SU35E16 or SU35E17) on the EAI Bus server moves the message to the Local Queue ECBS.FROM.EAI.FI SAP.BATCH and based on the queue attributes, the MQSeries Queue Manager (SU35E5) puts a trigger message on the ECBS.INIT queue.
- 5) The MQSeries Trigger Monitor application pulls the trigger message off of the ECBS.INIT queue.

- 6) The MQSeries Trigger Monitor application starts the eCBS Adapter.
- 7) The eCBS Adapter reads the message(s) from the ECBS.FROM.EAI.FISAP.REAL queue.
- 8) The eCBS Adapter invokes a stored procedure on eCBS.
- 9) The information that is returned from the stored procedure is placed in a MQSeries Message that will be placed on WAS.FROM.ECBS.FISAPSTATUS queue.
- 10) The MQSeries Reply message containing any eCBS response is retrieved by WAS.

Files

The following input files have been defined on the WAS server for access by the EAI Core test application in execution of this test scenario:

File Specification	Function
realRequestData.txt	Valid ReCBS data that returns successfully
realErrorData1.txt	Invalid ReCBS data that returns an error from MQSI
realErrorData2.txt	Invalid ReCBS data that returns an error from the eCBS custom adapter
realMQSIOutputData.txt	Results from the successful processing of realRequestData.txt by MQSI. This is the data that is sent to the eCBS custom adapter.
realResponseData.txt	Results from the successful processing of realMQSIOutputData.txt by the custom adapter. This is the data that is sent back to the test application.
realErrorMQSIOutputData1.txt	Results from the erroneous processing of realErrorData1.txt by MQSI. This is the data that is sent back to the test application.
realErrorMQSIOutputData2.txt	Results from the successful processing of realErrorData2.txt by MQSI. This is data that is sent to the eCBS custom adapter.
realErrorResponseData2.txt	Results from the erroneous processing of realErrorMQSIOutputData2.txt by the eCBS custom adapter. This is the data sent back to the test application.
SU35E5.msg	A file used to hold messages that cannot be sent to MQSeries, in the event MQSeries is not operational.
realUnexpectedErrorResponseData2.txt	Results from the erroneous processing of realMQSIOutputData.txt by the eCBS custom adapter. This is the data sent back to the test application.

Adapters

A custom MQ Adapter and a script file were developed for the eCBS system. The custom adapter, called MQECBS, is written in Java. The script file, called ECBSAdapter, is used to set environment variables and to execute the adapter.

- To start the adapter use the following command:
ECBSAdapter real

MQSI

The MQSI nodes and their function are documented below. The same MQSI message flow is used for both the batch and real-time adapter.

Node	Type	Description/Function
Input Message Queue From WAS	MQInput	Gets message from queue EAI.FROM.WAS.ReCBS
Trace1	Trace	Traces input message
Determine If Batch Request Type	Filter	Checks if requesttype = 1 If True then Build eCBS Batch Request If False goto Build eCBS Real-Time Request
Build eCBS Batch Request	Compute	Builds an output message for eCBS batch requests
Build eCBS Real-Time Request	Compute	Builds an output message for eCBS real-time requests
Build Error Message	Compute	Builds an error message
Send Error Back To Requestor	MQReply	Sends a message back to the requestor based on the

		reply to information
Trace2	Trace	Traces output message
Output to Batch eCBS Queue	MQOutput	Puts message to queue EAI.TO.eCBS.ReCBS.BATCH
Output to Real-Time eCBS Queue	MQOutput	Puts message to queue EAI.TO.eCBS.ReCBS.REAL

3.4.3 eCBS – Real Time Test Scenario Dependencies

To execute the eCBS System test scenario the following dependencies must be met,

- MQSeries Messaging and queue managers on each of the following systems are operational, logically referred to as SU35E5 and SU35E16 or SU35E17.
- MQSI must be operational on either of the following systems, logically referred to as SU35E16 or SU35E17.
- A stored procedure is created and operational.

3.4.4 eCBS – Real Time Test Scenario Inputs

From the initial entry on the WebSphere Application Server through MQSI and onto the adapter, each component is expecting the data a certain format. To validate each component, 4 test cycles were used for testing: Normal testing, expected error testing, unexpected error testing, and regression testing. The results of each test cycle are shown in Section 3.4.5.

3.4.4.1 Normal Test Cycle

Test Description

The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. The data format sent to MQSI is as follows:

Test application output and MQSI input: realRequestData.txt

```
<eCBSRequest>
  <Type>2</Type>
  <OPEID>00213000</OPEID>
  <Year>2001</Year>
</eCBSRequest>
```

3.4.4.2 Expected Error Test Cycle

Test Description

Two sets of test data were created to generate expected errors. The first data format generates an error in MQSI and the second data format generates an error in the custom adapter. Both data formats are as follows:

Test application output and MQSI input: realErrorData1.txt

```
<eCBSRequest>
  <Type>2</Type>
  <Year>2001</Year>
</eCBSRequest>
```

Test application output and MQSI input: realErrorData2.txt

```
<eCBSRequest>
  <Type>2</Type>
```

```
<OPEID>00213000</OPEID>
<Year>Two Thousand</Year>
</eCBSRequest>
```

3.4.4.3 Unexpected Error Test Cycle

Test Description

The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. While data is being processed by the custom adapter, two unexpected errors are simulated. The first simulates a MQSeries failure that makes all MQSeries objects unavailable after the data has been processed within eCBS but before a response has been sent to the source application, and the second simulates an eCBS failure that does not provide a response back to the adapter. The data format sent to MQSI is as follows:

Test application output and MQSI input: realRequestData.txt

```
<eCBSRequest>
  <Type>2</Type>
  <OPEID>00213000</OPEID>
  <Year>2001</Year>
</eCBSRequest>
```

3.4.4.4 Regression Test Cycle

Test Description

This test re-executes the test executed in the normal test cycle and expected error test cycle. The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. The data format sent to MQSI is as follows:

Input file: realRequestData.txt

```
<eCBSRequest>
  <Type>2</Type>
  <OPEID>00213000</OPEID>
  <Year>2001</Year>
</eCBSRequest>
```

3.4.5 eCBS – Real Time Test Scenario Expected Results

The execution of the test scenario returns an XML document that contains the result from MQSI and/or the custom adapter.

3.4.5.1 Expected Results for Normal Test Cycle

The results from successfully processing of input data by MQSI and the custom adapter are listed below.

MQSI Output sent to eCBS custom adapter as input: realMQSIOutputData.txt

```
<eCBSAdapterRequest>
  jdbc/CBSDataSource;eaiuser;oktryth13;{call EAIGeneralInfoProc('00213000','2001',?,?,?,?)};
</eCBSAdapterRequest>
```

eCBS Output sent back to the test application: realResponseData.txt

```
<eCBSAdapterResponse>
  <NewSchoolSw>N</NewSchoolSw>
  <WorkCollegeSw>N</WorkCollegeSw>
  <TradCalendarSw>Y</TradCalendarSw>
  <MultiCampusSw>N</MultiCampusSw>
  <FISAPStatusInd>2</FISAPStatusInd>
</eCBSAdapterResponse>
```

3.4.5.2 Expected Results for Expected Error Test Cycle

The results of invalid input data by MQSI is listed below.

MQSI Output sent to the test application: realErrorMQSIOutputData1.txt

```
<eCBSAdapterResponse>
  An Error Occurred In The ECBS Message Flow
</eCBSAdapterResponse>
```

The results of valid input data sent to MQSI but invalid data being sent to the custom adapter is listed below.

MQSI output sent to custom adapter as input: realErrorMQSIOutputData2.txt

```
<eCBSAdapterRequest>
  jdbc/CBSDataSource;eaiuser;oktryth13;{call EAIGeneralInfoProc('00213000','Two
  Thousand',?,?,?,?)};
</eCBSAdapterRequest>
```

eCBS Output sent back to the test application: realErrorResponseData2.txt

```
<eCBSAdapterResponse>
  Problem executing stored procedure. Error message is java.sql.SQLException: ORA-01403: no
  data found.ORA-06512: at "CBSDBA.EAIGENERALINFOPROC", line 15.ORA-06512: at
  line1.
</eCBSAdapterResponse>
```

3.4.5.3 Expected Results for Unexpected Error Test Cycle

The results from successfully processing by MQSI and unsuccessfully processing by the custom adapter are listed below. The unsuccessful processing is caused by a MQSeries failure that makes all MQSeries objects unavailable. Since a response has been sent back from eCBS to the adapter, but the adapter cannot send a response back to the source application because MQSeries is not operational, messages are written to a log file. This removes the problem of repeat processing of the same transaction and loss of unprocessed MQSeries messages. Messages found in the log file can be resent once the MQSeries is operational.

MQSI Output sent to eCBS custom adapter as input: realMQSIOutputData.txt

```
<eCBSAdapterRequest>
  jdbc/CBSDataSource;eaiuser;oktryth13;{call EAIGeneralInfoProc('00213000','2001',?,?,?,?)};
</eCBSAdapterRequest>
```

Log file used to store messages temporarily until MQSeries is operational: SU35E5.msg

```
<eCBSAdapterResponse>
  <NewSchoolSw>N</NewSchoolSw>
  <WorkCollegeSw>N</WorkCollegeSw>
  <TradCalendarSw>Y</TradCalendarSw>
  <MultiCampusSw>N</MultiCampusSw>
  <FISAPStatusInd>2</FISAPStatusInd>
</eCBSAdapterResponse>      EOM
```

eCBS Output sent back to the test application: realResponseData.txt

```
<eCBSAdapterResponse>
  <NewSchoolSw>N</NewSchoolSw>
  <WorkCollegeSw>N</WorkCollegeSw>
  <TradCalendarSw>Y</TradCalendarSw>
  <MultiCampusSw>N</MultiCampusSw>
  <FISAPStatusInd>2</FISAPStatusInd>
</eCBSAdapterResponse>
```

The results from successfully processing by MQSI and unsuccessfully processing by the custom adapter are listed below. The unsuccessful processing is caused by a simulated eCBS failure, that does not provide a response back to the custom adapter.

MQSI Output sent to eCBS custom adapter as input: realMQSIOutputData.txt

```
<eCBSAdapterRequest>
  jdbc/CBSDataSource;eaiuser;oktryth13;{ call EAIGeneralInfoProc('00213000','2001',?,?,?,?)};
</eCBSAdapterRequest>
```

eCBS Output sent back to the test application: realUnexpectedErrorResponseData2.txt

```
<eCBSAdapterResponse>
  Adapter Timeout - No Response From eCBS. Data Sent Successfully to eCBS. There Is A
  Problem With eCBS Sending A Response Back To The Adapter.
</eCBSAdapterResponse>
```

3.4.5.4 Expected Results for Regression Test Cycle

The results from successfully processing by MQSI and the custom adapter are listed below.

MQSI Output sent to eCBS custom adapter as input: realMQSIOutputData.txt

```
<eCBSRequestData>
  jdbc/CBSDataSource;eaiuser;oktryth13;{ call EAIGeneralInfoProc('00213000','2001',?,?,?,?)};
</eCBSRequestData>
```

eCBS Output sent back to the test application: realResponseData.txt

```
<eCBSAdapterResponse>
  <NewSchoolSw>N</NewSchoolSw>
  <WorkCollegeSw>N</WorkCollegeSw>
  <TradCalendarSw>Y</TradCalendarSw>
  <MultiCampusSw>N</MultiCampusSw>
  <FISAPStatusInd>2</FISAPStatusInd>
```

</eCBSAdapterResponse>

- This eCBS real-time test scenario was executed by the Release 2 EAI Core team and the expected results were received and validated.

3.5 EAI Component Test for Financial Management System (FMS) - Batch

The Financial Management System contains data records about disbursing funds to schools. It is an Oracle Financials COTS product running on HP-UX system. The EAI batch enablement of the FMS system will provide the capability to send and receive batch files to and from FMS.

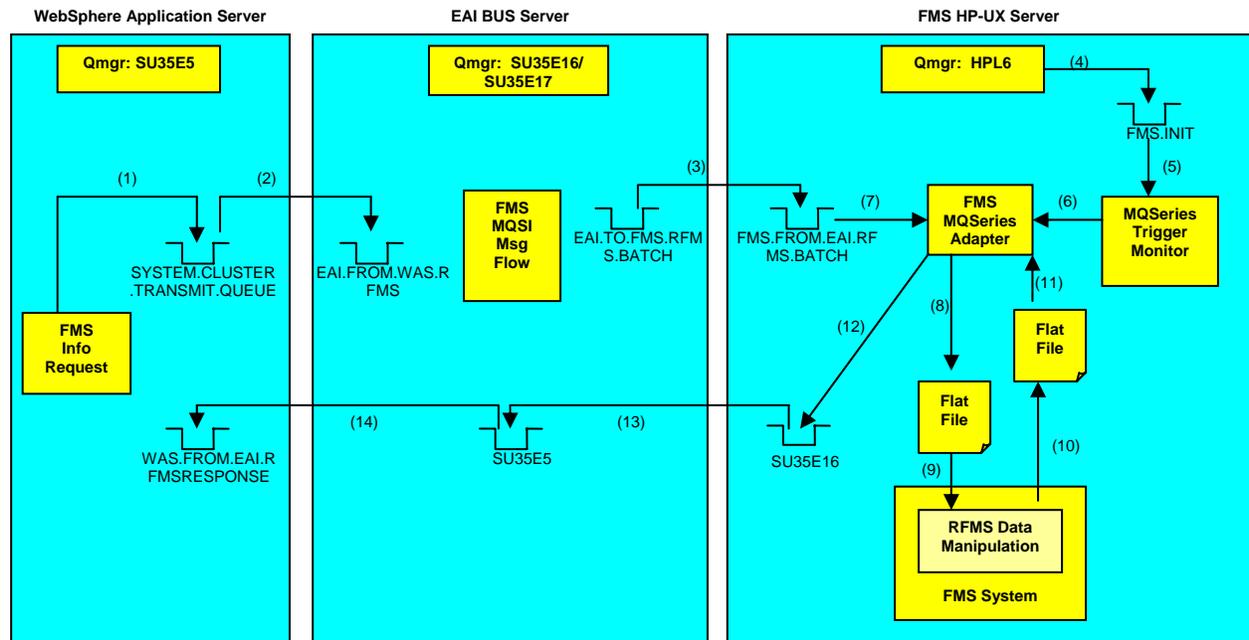
3.5.1 FMS – Batch Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure to the FMS system is based on processing RFMS data. The FMS development team provided a shell script that does some manipulation of the RFMS data. This test scenario detailed here does not include the processing of the data by FMS and only ensures that the adapter can read from and write to files on the FMS system. However, a test of the batch adapter that included processing of the data by FMS was performed successfully but is not documented in this section. A MQSeries Java adapter has been developed to connect the FMS system to the EAI Bus.

3.5.2 FMS – Batch Test Scenario Detailed Design Description

The sample function selected for the FMS system validates the ability of a user to send RFMS data from the test application. A XML string that contains the RFMS data and a flag to specify batch processing, is placed into a MQSeries message that is sent to the EAI Bus for transformation by MQSI. MQSI performs the defined message flow transformation and routes the transformed message data to the target system, FMS. Upon receipt by the FMS server, the custom adapter retrieves the message from the queue, creates a file for input to FMS that contains RFMS data and waits an output file that contains the manipulated RFMS data from FMS. The results are formatted in XML and placed in a MQSeries message that is sent back to the test application for display.

The figure below describes the message flow for the batch FMS custom adapter.



The flow of a MQSeries Request type message through the EAI FMS batch design is as follows:

- 1) A MQSeries Request type message is placed on the SYSTEM.CLUSTER.TRANSMIT.QUEUE, that is destined for a cluster queue located on the EAI Bus (SU35E16 or SU35E17).
- 2) The MQSeries Queue Manager (SU35E5) on the WAS moves the message to the cluster queue EAI.FROM.WAS.RFMS. The message is pulled from the EAI.FROM.WAS.RFMS and processed through the FMS MQSI Message Flow. The FMS MQSI Message Flow will determine whether the data should be processed as a real-time transaction or as a batch process by FMS.
- 3) Messages that will be processed by FMS as a real-time transaction are put to the Remote Queue EAI.TO.FMS.RFMS.BATCH by the FMS MQSI Message Flow. The MQSeries Queue Manager (SU35E16 or SU35E17) on the EAI Bus server moves the message to the Local Queue FMS.FROM.EAI.RFMS.BATCH on HPL6.
- 4) Based on the queue attributes for FMS.FROM.EAI.RFMS.BATCH, the MQSeries Queue Manager (HPL6) puts a trigger message on the FMS.INIT queue.
- 5) The MQSeries Trigger Monitor application pulls the trigger message off of the FMS.INIT queue.
- 6) The MQSeries Trigger Monitor application starts the FMS Adapter.
- 7) The FMS Adapter reads the message(s) from the FMS.FROM.EAI.RFMS.BATCH queue.
- 8) The FMS Adapter stores the message(s) into a flat file.
- 9) FMS monitors a directory for files created by the FMS Adapter and then processes that data.
- 10) When a response is to be sent back from FMS, FMS will store that information into another file in a directory which the FMS Adapter monitors.
- 11) The FMS Adapter monitors a directory for files created by FMS and then reads in the data from the file.
- 12) The FMS Adapter creates an MQSeries message based on the data in the file and places it onto the transmission queue for SU35E16.
- 13) The MQSeries Queue Manager (HPL6) on the FMS server moves the message to the Transmission Queue SU35E5.

14) The MQSeries Queue Manager (SU35E16) on the EAI Bus server moves the message to the Local Queue WAS.FROM.EAI.RFMSRESPONSE, specified as the reply-to queue by the FMS Info Request application.

Files

The following input files have been defined on the WAS server for access by the EAI Core test application in execution of this test scenario:

File Specification	Function
batchRequestData.txt	Valid RFMS data that returns successfully
batchErrorData1.txt	Invalid RFMS data that returns an error from MQSI
batchErrorData2.txt	Invalid RFMS data that returns an error from the FMS custom adapter
batchMQSIOutputData.txt	Results from the successful processing of batchRequestData.txt by MQSI. This is the data that is sent to the FMS custom adapter.
batchResponseData.txt	Results from the successful processing of batchMQSIOutputData.txt by the custom adapter. This is the data that is sent back to the test application.
batchErrorMQSIOutputData1.txt	Results from the erroneous processing of batchErrorData1.txt by MQSI. This is the data that is sent back to the test application.
batchErrorMQSIOutputData2.txt	Results from the successfully processing of batchErrorData2.txt by MQSI. This is data that is sent to the FMS custom adapter.
batchErrorResponseData2.txt	Results from the erroneous processing of batchErrorMQSIOutputData2.txt by the FMS custom adapter. This is the data sent back to the test application.
SU35E5.msg	A file used to hold messages that cannot be sent to MQSeries, in the event MQSeries is not operational.
batchUnexpectedErrorResponseData2.txt	Results from the erroneous processing of batchMQSIOutputData.txt by the FMS custom adapter. This is the data sent back to the test application.

Adapters

A custom MQ Adapter and a script file were developed for the FMS system. The custom adapter, called MQFMS, is written in Java. The script file, called FMSAdapter, is used to set environment variables and to execute the adapter.

- To start the adapter use the following command:
 FMSAdapter batch

MQSI

The MQSI nodes and their function are documented below. The same MQSI message flow is used for both the batch and real-time adapter.

Node	Type	Description/Function
Input Message Queue From WAS	MQInput	Gets message from queue EAI.FROM.WAS.RFMS
Trace1	Trace	Traces input message
Determine If Batch Request Type	Filter	Checks if requesttype = 1 If True then Build FMS Batch Request If False goto Build FMS Real-Time Request
Build FMS Batch Request	Compute	Builds an output message for FMS batch requests
Build FMS Real-Time Request	Compute	Builds an output message for FMS real-time requests
Build Error Message	Compute	Builds an error message

Send Error Back To Requestor	MQReply	Sends a message back to the requestor based on the reply to information
Trace2	Trace	Traces output message
Output to Batch FMS Queue	MQOutput	Puts message to queue EAI.TO.FMS.RFMS.BATCH
Output to Real-Time FMS Queue	MQOutput	Puts message to queue EAI.TO.FMS.RFMS.REAL

3.5.3 FMS – Batch Test Scenario Dependencies

To execute the FMS System test scenario the following dependencies must be met:

- MQSeries Messaging and queue managers on each of the following systems are operational, logically referred to as SU35E5, SU35E16 or SU35E17, and HPL6.
- MQSI must be operational on either of the following systems, logically referred to as SU35E16 or SU35E17.
- The adapter has access to read and write files on the FMS system.
- The FMS shell script that will process the batch file has been created and is operational.

3.5.4 FMS – Batch Test Scenario Inputs

From the initial entry on the WebSphere Application Server through MQSI and onto the adapter, each component is expecting the data a certain format. To validate each component, 4 test cycles were used for testing: Normal testing, expected error testing, unexpected error testing, and regression testing. The results of each test cycle are shown in Section 3.5.5.

3.5.4.1 Normal Test Cycle

Test Description

The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. The data format sent to MQSI is as follows:

Test application output and MQSI input: batchRequestData.txt

```
<FMSRequest>
  <Type>1</Type>
  <Transaction>H026BA    20010731140747
  OB20010731P063J 1999988519994101LEN  H12000063100000000565656040651853
  120010718      PL199907012005093020000000N
  PY20010731P063J 1999988519994101LEN  H12000063100000000565656040651853
  120010718      PL199907012005093020000000N          T026BA
  2001073114074700002200000000002780148
  ZTRANSTRAIL2001073114074700016100000000016731681      </Transaction>
</FMSRequest>
```

3.5.4.2 Expected Error Test Cycle

Test Description

Two sets of test data were created to generate expected errors. The first data format generates an error in MQSI and the second data format generates an error in the custom adapter. Both data formats are as follows:

Test application output and MQSI input: batchErrorData1.txt

```
<FMSRequest>
```

```
<Type>1</Type>
</FMSRequest>
```

Test application output and MQSI input: batchErrorData2.txt

```
<FMSRequest>
  <Type>1</Type>
  <Transaction>ABCDEFG</Transaction>
</FMSRequest>
```

3.5.4.3 Unexpected Error Test Cycle

Test Description

The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. While data is being processed by the custom adapter, two unexpected errors are simulated. The first simulates a MQSeries failure that makes all MQSeries objects unavailable after the data has been processed within FMS but before a response has been sent to the source application, and the second simulates an FMS failure that does not provide a response back to the adapter. The data format sent to MQSI is as follows:

Test application output and MQSI input: batchRequestData.txt

```
<FMSRequest>
  <Type>1</Type>
  <Transaction>H026BA    20010731140747
  OB20010731P063J 1999988519994101LEN  H12000063100000000565656040651853
  120010718      PL199907012005093020000000N
  PY20010731P063J 1999988519994101LEN  H12000063100000000565656040651853
  120010718      PL199907012005093020000000N          T026BA
  2001073114074700002200000000002780148
  ZTRANSTRAIL2001073114074700016100000000016731681      </Transaction>
</FMSRequest>
```

3.5.4.4 Regression Test Cycle

Test Description

This test re-executes the test executed in the normal test cycle and expected error test cycle. The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. The data format sent to MQSI is as follows:

Input file: batchRequestData.txt

```
<FMSRequest>
  <Type>1</Type>
  <Transaction>H026BA    20010731140747
  OB20010731P063J 1999988519994101LEN  H12000063100000000565656040651853
  120010718      PL199907012005093020000000N
  PY20010731P063J 1999988519994101LEN  H12000063100000000565656040651853
  120010718      PL199907012005093020000000N          T026BA
  2001073114074700002200000000002780148
  ZTRANSTRAIL2001073114074700016100000000016731681      </Transaction>
```

</FMSRequest>

3.5.5 FMS – BatchTest Scenario Expected Results

The execution of the test scenario returns an XML document that contains the result from MQSI and/or the custom adapter.

3.5.5.1 Expected Results for Normal Test Cycle

The results from successfully processing by MQSI and the custom adapter are listed below.

MQSI Output sent to FMS custom adapter as input: batchMQSIOutputData.txt

<FMSAdapterRequest>

```
/home/mqm/batch.txt;H026BA 20010731140747
OB20010731P063J 1999988519994101LEN H12000063100000000565656040651853
120010718 PL199907012005093020000000N
PY20010731P063J 1999988519994101LEN H12000063100000000565656040651853
120010718 PL199907012005093020000000N T026BA
2001073114074700002200000000002780148
ZTRANSTRAIL2001073114074700016100000000016731681
;/home/mqm/batch_mod.txt;
```

</FMSAdapterRequest>

FMS Output sent back to the test application: batchResponseData.txt

<FMSAdapterResponse>

```
H*26BA 2**1*73114*747
OB2**1*731P*63J 19999885199941*1LEN H12****631*****565656*4*651853
12**1*718 PL1999*7*12**5*93*2*****N
PY2**1*731P*63J 19999885199941*1LEN H12****631*****565656*4*651853
12**1*718 PL1999*7*12**5*93*2*****N T*26BA
2**1*73114*747****22*****278*148
ZTRANSTRAIL2**1*73114*747***161*****16731681
```

</FMSAdapterResponse>

3.5.5.2 Expected Results for Expected Error Test Cycle

The results of invalid input data by MQSI is listed below.

MQSI Output sent to the test application: batchErrorMQSIOutputData1.txt

<FMSAdapterResponse>

An Error Occurred In The FMS Message Flow

</FMSAdapterResponse>

The results of valid input data sent to MQSI but invalid data being sent to the custom adapter is listed below.

MQSI Output sent to FMS custom adapter as input: batchErrorMQSIOutputData2.txt

<FMSAdapterRequest>

```
/home/mqm/batch.txt;ABCDEFGG;/home/mqm/batch_mod.txt;
```

</FMSAdapterRequest>

FMS Output sent back to the test application: batchErrorResponseData2.txt

<FMSAdapterResponse>
ABCDEF
</FMSAdapterResponse>

3.5.5.3 Expected Results for Unexpected Error Test Cycle

The results from successfully processing by MQSI and unsuccessfully processing by the custom adapter are listed below. The unsuccessful processing is caused by a MQSeries failure that makes all MQSeries objects unavailable. Since a response has been sent back from FMS to the adapter, but the adapter cannot send a response back to the source application because MQSeries is not operational, messages are written to a log file. This removes the problem of repeat processing of the same transaction and loss of unprocessed MQSeries messages. Messages found in the log file can be resent once the MQSeries is operational.

MQSI Output sent to FMS custom adapter as input: batchMQSIOutputData.txt

<FMSAdapterRequest>
/home/mqm/batch.txt;H026BA 20010731140747
OB20010731P063J 1999988519994101LEN H12000063100000000565656040651853
120010718 PL199907012005093020000000N
PY20010731P063J 1999988519994101LEN H12000063100000000565656040651853
120010718 PL199907012005093020000000N T026BA
2001073114074700002200000000002780148
ZTRANSTRAIL2001073114074700016100000000016731681
;/home/mqm/batch_mod.txt;
</FMSAdapterRequest>

Log file used to store messages temporarily until MQSeries is operational: SU35E5.msg

<FMSAdapterResponse>
H*26BA 2**1*73114*747
OB2**1*731P*63J 19999885199941*1LEN H12***631*****565656*4*651853
12**1*718 PL1999*7*12**5*93*2*****N
PY2**1*731P*63J 19999885199941*1LEN H12***631*****565656*4*651853
12**1*718 PL1999*7*12**5*93*2*****N T*26BA
2**1*73114*747***22*****278*148
ZTRANSTRAIL2**1*73114*747***161*****16731681
</FMSAdapterResponse> EOM

FMS Output sent back to the test application: batchResponseData.txt

<FMSAdapterResponse>
H*26BA 2**1*73114*747
OB2**1*731P*63J 19999885199941*1LEN H12***631*****565656*4*651853
12**1*718 PL1999*7*12**5*93*2*****N
PY2**1*731P*63J 19999885199941*1LEN H12***631*****565656*4*651853
12**1*718 PL1999*7*12**5*93*2*****N T*26BA

```
2**1*73114*747****22*****278*148
ZTRANSTRAIL2**1*73114*747***161*****16731681
</FMSAdapterResponse>
```

The results from successfully processing by MQSI and unsuccessfully processing by the custom adapter are listed below. The unsuccessful processing is caused by a simulated FMS failure that does not provide a response back to the custom adapter.

MQSI Output sent to FMS custom adapter as input: batchMQSIOutputData.txt

```
<FMSAdapterRequest>
/home/mqm/batch.txt;H026BA    20010731140747
OB20010731P063J 1999988519994101LEN  H12000063100000000565656040651853
120010718      PL199907012005093020000000N
PY20010731P063J 1999988519994101LEN  H12000063100000000565656040651853
120010718      PL199907012005093020000000N          T026BA
2001073114074700002200000000002780148
ZTRANSTRAIL200107311407470001610000000016731681
;/home/mqm/batch_mod.txt;
</FMSAdapterRequest>
```

FMS Output sent back to the test application: batchUnexpectedErrorResponseData2.txt

```
<FMSAdapterResponse>
Adapter Timeout - No Response From FMS. Data Sent Successfully to FMS. There Is A
Problem With FMS Sending A Response Back To The Adapter.
</FMSAdapterResponse>
```

3.5.5.4 Expected Results for Regression Test Cycle

The results from successfully processing by MQSI and the custom adapter are listed below.

MQSI Output sent to FMS custom adapter as input: batchMQSIOutputData.txt

```
<FMSAdapterRequest>
/home/mqm/batch.txt;H026BA    20010731140747
OB20010731P063J 1999988519994101LEN  H12000063100000000565656040651853
120010718      PL199907012005093020000000N
PY20010731P063J 1999988519994101LEN  H12000063100000000565656040651853
120010718      PL199907012005093020000000N          T026BA
2001073114074700002200000000002780148
ZTRANSTRAIL200107311407470001610000000016731681
;/home/mqm/batch_mod.txt;
</FMSAdapterRequest>
```

FMS Output sent back to the test application: batchResponseData.txt

```
<FMSAdapterResponse>
H*26BA    2**1*73114*747
OB2**1*731P*63J 19999885199941*1LEN  H12****631*****565656*4*651853
12**1*718      PL1999*7*12**5*93*2*****N
PY2**1*731P*63J 19999885199941*1LEN  H12****631*****565656*4*651853
12**1*718      PL1999*7*12**5*93*2*****N          T*26BA
```

```
2**1*73114*747****22*****278*148  
ZTRANSTRAIL2**1*73114*747***161*****16731681  
</FMSAdapterResponse>
```

- This FMS batch test scenario was executed by the Release 2 EAI Core team and the expected results were received and validated.

3.6 EAI Component Test for Financial Management System (FMS) - Real-Time

The Financial Management System contains data records about disbursing funds to schools. It is an Oracle Financials COTS product running on HP-UX system. The EAI real-time enablement of the FMS system will provide the capability to send real-time transactions to FMS by placing these transactions in a database staging table.

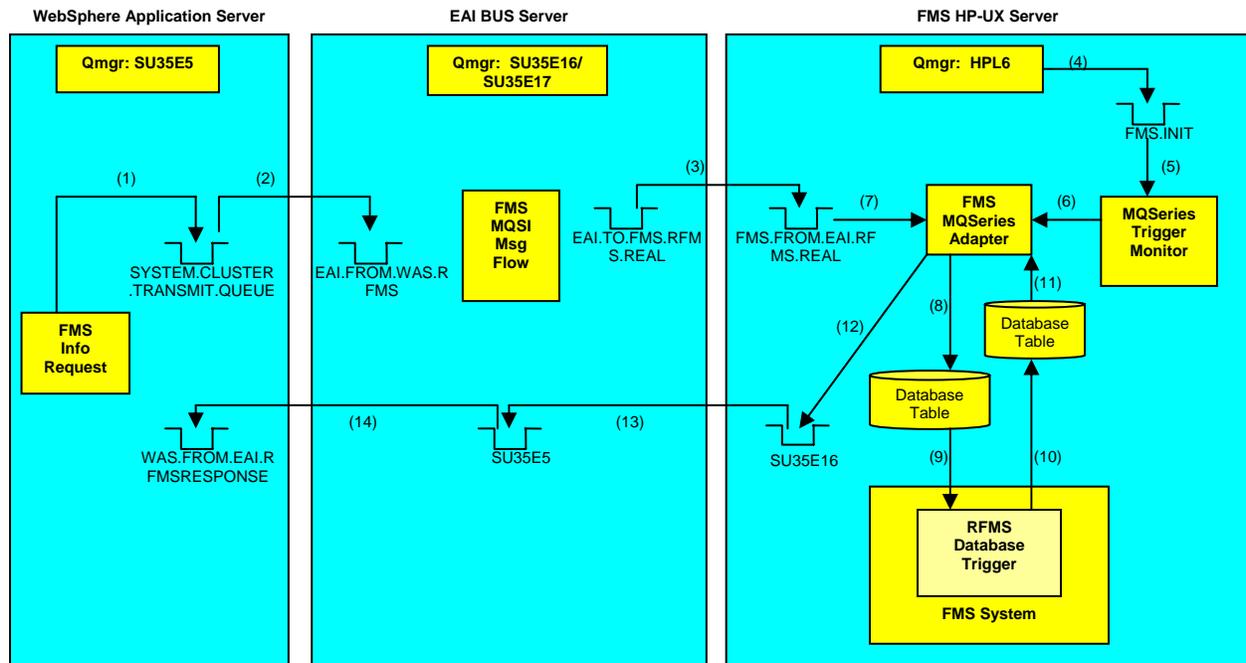
3.6.1 FMS – Real Time Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure to the FMS system is based on a RFMS request. The FMS development team provided access to two database tables, and a database trigger that moves data found in the input table to the output table that simulates processing by FMS. This test scenario does not include the processing of the data by FMS, but only ensures that the adapter can read from and write to the two database tables on the FMS system. A MQSeries Java adapter has been developed to connect the FMS system to the EAI Bus.

3.6.2 FMS – Real Time Test Scenario Detailed Design Description

The sample function selected for the FMS system validates the ability of a user to enter a Grantee DUNS number and transaction code from the test application. A XML string that contains the Grantee DUNS number, transaction code and a flag to specify real-time processing, is placed into a MQSeries message that is sent to the EAI Bus for transformation by MQSI. MQSI performs the defined message flow transformation and routes the transformed message data to the target system, FMS. Upon receipt by the FMS server the custom adapter retrieves the message from the queue, executes an insert statement into the input database table, waits for data to be populated into the output database table, and executes a select statement on the output database table to retrieve the results. The results are formatted in XML and placed in a MQSeries message that is sent back to the test application for display.

The figure below describes the message flow for the real-time FMS custom adapter.



The flow of a MQSeries Request type message through the EAI FMS real-time design is as follows:

- 1) A MQSeries Request type message is placed on the SYSTEM.CLUSTER.TRANSMIT.QUEUE, that is destined for a cluster queues located on the EAI Bus (SU35E16 or SU35E17).
- 2) The MQSeries Queue Manager (SU35E5) on the WAS moves the message to the cluster queue EAI.FROM.WAS.RFMS. The message is pulled from the EAI.FROM.WAS.RFMS and processed through the FMS MQSI Message Flow. The FMS MQSI Message Flow will determine whether the data should be processed as a real-time transaction or as a batch process by FMS.
- 3) Messages that will be processed by FMS as a real-time transaction are put to the Remote Queue EAI.TO.FMS.RFMS.REAL by the FMS MQSI Message Flow. The MQSeries Queue Manager (SU35E16 or SU35E17) on the EAI Bus server moves the message to the Local Queue FMS.FROM.EAI.RFMS.REAL on HPL6.
- 4) Based on the queue attributes for FMS.FROM.EAI.RFMS.REAL, the MQSeries Queue Manager (HPL6) puts a trigger message on the FMS.INIT queue.
- 5) The MQSeries Trigger Monitor application pulls the trigger message off of the FMS.INIT queue.
- 6) The MQSeries Trigger Monitor application starts the FMS Adapter.
- 7) The FMS Adapter reads the message(s) from the FMS.FROM.EAI.RFMS.REAL queue.
- 8) The FMS Adapter stores the message(s) into an input database table.
- 9) FMS monitors the input database table and processes the data when new information is stored into the database table.
- 10) When a real-time response is to be sent back from FMS, FMS will store that information into the output database table, which the real-time component/program of the FMS Adapter monitors.
- 11) The FMS Adapter monitors an output database table for responses back from FMS.
- 12) The FMS Adapter creates an MQSeries message based on the rows in a database and places it onto the transmission queue for SU35E16.
- 13) The MQSeries Queue Manager (HPL6) on the FMS server moves the message to the Transmission Queue SU35E5.

14) The MQSeries Queue Manager (SU35E16) on the EAI Bus server moves the message to the Local Queue WAS.FROM.EAI.RFMSRESPONSE, specified as the reply-to queue by the FMS Info Request application.

Files

The following input files have been defined on the WAS server for access by the EAI Core test application in execution of this test scenario:

File Specification	Function
realRequestData.txt	Valid RFMS data that returns successfully
realErrorData1.txt	Invalid RFMS data that returns an error from MQSI
realErrorData2.txt	Invalid RFMS data that returns an error from the FMS custom adapter
realMQSIOutputData.txt	Results from the successful processing of realRequestData.txt by MQSI. This is the data that is sent to the FMS custom adapter.
realResponseData.txt	Results from the successful processing of realMQSIOutputData.txt by the custom adapter. This is the data that is sent back to the test application.
realErrorMQSIOutputData1.txt	Results from the erroneous processing of realErrorData1.txt by MQSI. This is the data that is sent back to the test application.
realErrorMQSIOutputData2.txt	Results from the successful processing of realErrorData2.txt by MQSI. This is data that is sent to the FMS custom adapter.
realErrorResponseData2.txt	Results from the erroneous processing of realErrorMQSIOutputData2.txt by the FMS custom adapter. This is the data sent back to the test application.
SU35E5.msg	A file used to hold messages that cannot be sent to MQSeries, in the event MQSeries is not operational.
realUnexpectedErrorResponseData2.txt	Results from the erroneous processing of realMQSIOutputData.txt by the FMS custom adapter. This is the data sent back to the test application.

Adapters

A custom MQ Adapter and a script file were developed for the FMS system. The custom adapter, called MQFMS, is written in Java. The script file, called FMSAdapter, is used to set environment variables and to execute the adapter.

- To start the adapter use the following command:
 FMSAdapter real

MQSI

The MQSI nodes and their function are documented below. The same MQSI message flow is used for both the batch and real-time adapter.

Node	Type	Description/Function
Input Message Queue From WAS	MQInput	Gets message from queue EAI.FROM.WAS.RFMS
Trace1	Trace	Traces input message
Determine If Batch Request Type	Filter	Checks if requesttype = 1 If True then Build FMS Batch Request If False goto Build FMS Real-Time Request
Build FMS Batch Request	Compute	Builds an output message for FMS batch requests
Build FMS Real-Time Request	Compute	Builds an output message for FMS real-time requests
Build Error Message	Compute	Builds an error message
Send Error Back To Requestor	MQReply	Sends a message back to the requestor based on the

		reply to information
Trace2	Trace	Traces output message
Output to Batch FMS Queue	MQOutput	Puts message to queue EAI.TO.FMS.RFMS.BATCH
Output to Real-Time FMS Queue	MQOutput	Puts message to queue EAI.TO.FMS.RFMS.REAL

3.6.3 FMS – Real Time Test Scenario Dependencies

To execute the FMS System test scenario the following dependencies must be met,

- MQSeries Messaging and queue managers on each of the following systems are operational, logically referred to as SU35E5, SU35E16 or SU35E17, and HPL6.
- MQSI must be operational on either of the following system, logically referred to as SU35E16 or SU35E17.
- Two FMS database tables are created.
- A database trigger is created and operational.

3.6.4 FMS – Real Time Test Scenario Inputs

From the initial entry on the WebSphere Application Server through MQSI and onto the adapter, each component is expecting the data a certain format. To validate each component, 4 test cycles were used for testing: Normal testing, expected error testing, unexpected error testing, and regression testing. The results of each test cycle are shown in Section 3.6.5.

3.6.4.1 Normal Test Cycle

Test Description

The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. The data format sent to MQSI is as follows:

Test application output and MQSI input: realRequestData.txt

```
<FMSRequest>
  <Type>2</Type>
  <TransactionCode>OB</TransactionCode>
  <GranteeDUNSNNo>12345678901</GranteeDUNSNNo>
</FMSRequest>
```

3.6.4.2 Expected Error Test Cycle

Test Description

Two sets of test data were created to generate expected errors. The first data format generates an error in MQSI and the second data format generates an error in the custom adapter. Both data formats are as follows:

Test application output and MQSI input: realErrorData1.txt

```
<FMSRequest>
  <GranteeDUNSNNo>12345678901</GranteeDUNSNNo>
</FMSRequest>
```

Test application output and MQSI input: realErrorData2.txt

```
<FMSRequest>
  <Type>2</Type>
  <TransactionCode>OB</TransactionCode>
  <GranteeDUNSNo> 12345678901234</GranteeDUNSNo>
</FMSRequest>
```

3.6.4.3 Unexpected Error Test Cycle

Test Description

The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. While data is being processed by the custom adapter, two unexpected errors are simulated. The first simulates a MQSeries failure that makes all MQSeries objects unavailable after the data has been processed within eCBS but before a response has been sent to the source application, and the second simulates an FMS failure that does not provide a response back to the adapter. The data format sent to MQSI is as follows:

Test application output and MQSI input: realRequestData.txt

```
<FMSRequest>
  <Type>2</Type>
  <TransactionCode>OB</TransactionCode>
  <GranteeDUNSNo>12345678901</GranteeDUNSNo>
</FMSRequest>
```

3.6.4.4 Regression Test Cycle

Test Description

This test re-executes the test executed in the normal test cycle and expected error test cycle. The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. The data format sent to MQSI is as follows:

Input file: realRequestData.txt

```
<FMSRequest>
  <Type>2</Type>
  <TransactionCode>OB</TransactionCode>
  <GranteeDUNSNo>12345678901</GranteeDUNSNo>
</FMSRequest>
```

3.6.5 FMS – Real Time Test Scenario Expected Results

The execution of the test scenario returns an XML document that contains the result from MQSI and/or the custom adapter.

3.6.5.1 Expected Results for Normal Test Cycle

The results from successfully processing by MQSI and the custom adapter are listed below.

MQSI Output sent to FMS custom adapter as input: realMQSIOutputData.txt

```
<FMSAdapterRequest>
```



```
<GAPS_AWARD_NUMBER>1023120011234</GAPS_AWARD_NUMBER>
<REVERSE_CODE>1</REVERSE_CODE>
<AMOUNT>66666666666666</AMOUNT>
<GAPS_TRANSACTION_DATE>20010910</GAPS_TRANSACTION_DATE>
<GRANT_AWARD_PHASE>CLOSE</GRANT_AWARD_PHASE>
<FUND_CODE>55555</FUND_CODE>
<FUNDING_FISCAL_YEAR>4444</FUNDING_FISCAL_YEAR>
<CATEGORY>1</CATEGORY>
<BUDGET_FISCAL_YEAR>4444</BUDGET_FISCAL_YEAR>
<ORGANIZATION_CODE>88888888</ORGANIZATION_CODE>
<LIMITATION>333</LIMITATION>
<OBJECT_CLASS>55555</OBJECT_CLASS>
<ACTIVITY>333</ACTIVITY>
<CFDA>333</CFDA>
<COHORT_YEAR>4444</COHORT_YEAR>
<SECTOR>1</SECTOR>
<FUND_CONTROL_CODE>55555</FUND_CONTROL_CODE>
<FUTURE_USE>666666</FUTURE_USE>
<OFFICE_FINANCE_CODE>121212121212</OFFICE_FINANCE_CODE>
<SF215_SCHEDULE_NUMBER>14141414141414</SF215_SCHEDULE_NUMBER>
<CONFIRMATION_DATE>88888888</CONFIRMATION_DATE>
<TREASURY_5515_NUMBER>666666</TREASURY_5515_NUMBER>
<ERROR_CODE>22</ERROR_CODE>
<ERROR_TRANSACTION_TYPE>22</ERROR_TRANSACTION_TYPE>
<PAY_CONTROL_NUMBER>1234567891011</PAY_CONTROL_NUMBER>
<GRANTEE_DUNS_NUMBER>12345678901</GRANTEE_DUNS_NUMBER>
<GRANTEE_DUNS_TYPE_CODE>1</GRANTEE_DUNS_TYPE_CODE>
<PARTICIPATION_LEVEL>22</PARTICIPATION_LEVEL>
<GAPS_DEBIT_CREDIT_DETAIL_NUM>33333333333333333333333333333333</GAPS_DEB
IT_CREDIT_DETAIL_NUM>
</FMSAdapterResponse>
```

FMS Output sent back to the test application: realResponseData.txt

```
<FMSAdapterResponse>
  <TRANSACTION_CODE>OB</TRANSACTION_CODE>
  <BATCH_NUMBER>200</BATCH_NUMBER>
  <RUN_DATE>20010921</RUN_DATE>
  <GAPS_AWARD_NUMBER>1023120011234</GAPS_AWARD_NUMBER>
  <REVERSE_CODE>1</REVERSE_CODE>
  <AMOUNT>66666666666666</AMOUNT>
  <GAPS_TRANSACTION_DATE>20010910</GAPS_TRANSACTION_DATE>
  <GRANT_AWARD_PHASE>CLOSE</GRANT_AWARD_PHASE>
  <FUND_CODE>55555</FUND_CODE>
  <FUNDING_FISCAL_YEAR>4444</FUNDING_FISCAL_YEAR>
  <CATEGORY>1</CATEGORY>
  <BUDGET_FISCAL_YEAR>4444</BUDGET_FISCAL_YEAR>
  <ORGANIZATION_CODE>88888888</ORGANIZATION_CODE>
  <LIMITATION>333</LIMITATION>
```


3.7 EAI Component Test for LO System – Electronic Master Promissory Note (eMPN)

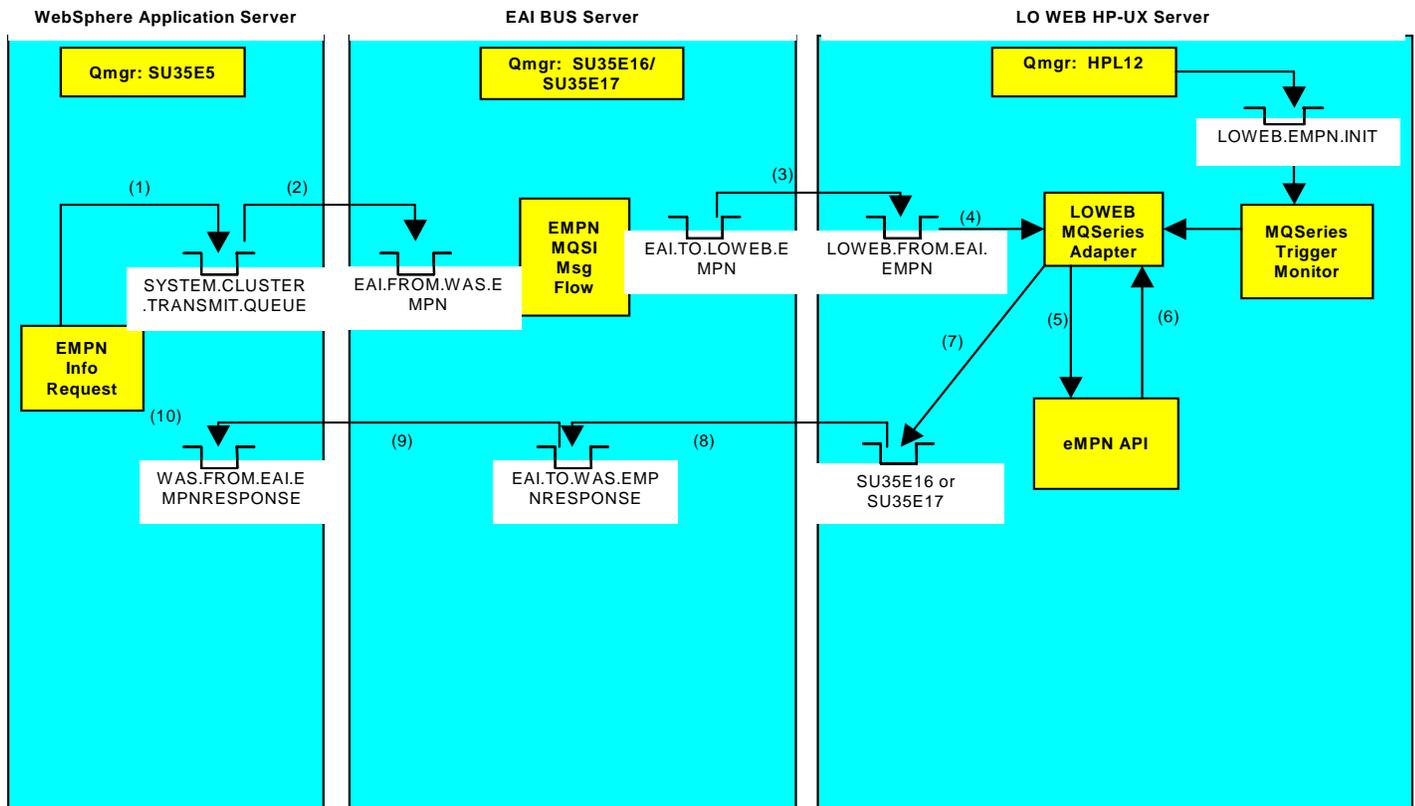
The eMPN system contains information relating to electronic master promissory notes. It is a custom system that runs on the Loan Origination Web Server. The EAI enablement of the eMPN system will provide the capability to retrieve eMPN reference data and transmit it to other systems.

3.7.1 LO System –(eMPN) Test Scenario Description

This test will demonstrate the functionality of MQSeries messaging across disparate systems, the use of a MQSeries Integrator Message flow, and the use of an MQSeries adapter written for eMPN.

The application to be used for the EAI eMPN test is a retrieval of eMPN reference data. An XML message containing Social Security Number, Date of Birth, and Signed Date will be sent from the WebSphere Application Server to the EAI bus. In the MQSI message flow, the message will be transformed by adding the host name and port number of the socket program that will provide the eMPN data. The message will then be sent to the LO System, where an adapter will read from the queue. The adapter will then parse the incoming message for the API host and port information and then invoke an API to retrieve eMPN record(s). The adapter will take the data returned from the API and put the message on the queue to be sent back to the WebSphere Application Server. If the adapter encounters an error, such as a timeout by the API, it will construct an XML message explaining the error and put that on the queue for return to the WebSphere Application Server.

3.7.2 LO System – (eMPN) Test Scenario Detailed Design Description



1. A XML string, consisting of an applicant's SSN, date of birth, and signed date, is sent from the WebSphere Application Server (WAS) to the EAI bus. (Shown by 1 & 2 in diagram above.)
2. In the MQSI message flow, the message will be transformed by appending the host name and port of the socket program that is providing eMPN information. (Shown by box in between (2) and (3) above.)
3. The message will then be sent to the LO Web Server, where an adapter will read from the queue and invoke the eMPN API, which will return the requested eMPN information to the adapter. (Shown by 4, 5, 6 above)
4. The adapter will then build an output message from the information returned by the eMPN API and put the message on the queue to be sent back to the WebSphere Application Server. (Shown by 7, 8, 9, 10 above)

Files

The following are the files used in conjunction with the eMPN adapter.

File Specification	Function
hpl12:/dev2/users/bmalki01/empn/empn.sh	A shell script used to set up environment variables and then run the adapter.
hpl12:/dev2/users/bmalki01/empn/EMPN.class	The java class that contains the adapter.
hpl12:/dev2/users/bmalki01/empn/empn1.jar	A java repository that contains part of the eMPN API.
hpl12:/dev2/users/bmalki01/empn/common.jar	A java repository that contains part of the eMPN API.
Hpl12:/dev2/users/bmalki01/empn/empncore.xml	The MQSeries Application Messaging Interface (AMI) repository.
hpl12:/export/home/mqm/LOWEB.txt	A file containing the definitions of the MQSeries objects on the LO Web Server
su35e16:/export/home/mqm/SU35E16.txt	A file containing the definitions of the MQSeries objects on su35e16.
su35e17:/export/home/mqm/SU35E17.txt	A file containing the definitions of the MQSeries objects on su35e17.
hpl12:/dev2/users/bmalki01/empn/empnTestInput1.xml	An XML file containing test input to the eMPN adapter.
hpl12:/dev2/users/bmalki01/empn/empnTestInput2.xml	An XML file containing test input to the eMPN adapter.

Adapter

The adapter is invoked by calling /dev2/users/bmalki01/empn.sh. This shell script sets environment variables and then invokes the actual java adapter by calling 'java EMPN'. There are no parameters, as the input data is retrieved from MQSeries and the MQSeries objects are specified in the AMI repository.

The following are the MQSI objects used in conjunction with the eMPN adapter:

Node	Type	Description/Function
Input Message From WAS	MQInput	Gets message from queue EAI.FROM.WAS.EMPNN
Trace1	Trace	Traces input message
Verify Request Type	Filter	Validate message contents
Forward Request to LOWEB	MQOutput	Sends a message to LOWEB
Build Error Message	Compute	Builds an error message
Send Error Back To Requestor	MQOutput	Sends a message back to the requestor based on the reply to information

3.7.3 LO System – (eMPN) Test Scenario Dependencies

The following defines the dependencies and resource requirements for a successful eMPN test.

- MQSeries running on logically referred to as SU35E5, SU35E16 or SU35E17, and HPL12
- MQSI must be operational on either of the following systems, logically referred to as SU35E16 or SU35E17.
- EDS eMPN API is operational

3.7.4 LO System – (eMPN) Test Scenario Inputs

From the initial entry on the WebSphere Application Server through MQSI and onto the adapter, each component is expecting the data a certain format. To validate each component, 4 test cycles were used for testing: Normal testing, expected error testing, unexpected error testing, and regression testing. The results of each test cycle are shown in Section 3.7.5.

3.7.4.1 Normal Test Cycle

Test Description

The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. The data format sent to MQSI is as follows.

XML Input String #1 (Correct Information)

```
<?xml version="1.0" standalone="yes"?>
<EPNRequestRoot>
  <requestType>ref</requestType>
  <Request>
    <ssn>217165454</ssn>
    <dob>03/20/1962</dob>
    <signedDate>10/01/2001</signedDate>
  </Request>
</EPNRequestRoot>
```

XML Input String #2 (Incorrect Information)

```
<?xml version="1.0" standalone="yes"?>
<EPNRequestRoot>
  <requestType>ref</requestType>
```

```
<Request>
  <ssn>123456789</ssn>
  <dob>03/20/1962</dob>
  <signedDate>10/01/2001</signedDate>
</Request>
</EPNRequestRoot>
```

3.7.4.2 Expected Error Test Cycle

Test Description

Two sets of test data were created to generate expected errors. The first data is correctly formatted XML and the API calls an incorrect port number. The second data is the same XML string but the API calls an incorrect host ip address. The data formats are as follows:

Test #1

Simulate P-Note API server being down. Do this by altering MQSI message flow to supply an incorrect port number.

```
<?xml version="1.0" standalone="yes"?>
<EPNRequestRoot>
  <requestType>ref</requestType>
  <Request>
    <ssn>217165454</ssn>
    <dob>03/20/1962</dob>
    <signedDate>10/01/2001</signedDate>
  </Request>
</EPNRequestRoot>
```

Test #2

Adapter receives an invalid port number for the P-Note API Server. Simulate this by altering MQSI message flow to supply an incorrect host ip address.

```
<?xml version="1.0" standalone="yes"?>
<EPNRequestRoot>
  <requestType>ref</requestType>
  <Request>
    <ssn>217165454</ssn>
    <dob>03/20/1962</dob>
    <signedDate>10/01/2001</signedDate>
  </Request>
</EPNRequestRoot>
```

3.7.4.3 Unexpected Error Test Cycle

Test Description

The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. While data is being processed by the custom adapter, two unexpected errors are simulated. The first simulates a MQSeries failure that makes all MQSeries objects unavailable after the data has been processed within eMPN but before a response has been sent to the source application, and the second simulates an eMPN failure that does not provide a response back to the adapter. The data format sent to MQSI is as follows.

Test #1

Simulate an MQSeries failure by shutting down the queue manager on the LO Web Server.

```
<?xml version="1.0" standalone="yes"?>
<EPNRequestRoot>
  <requestType>ref</requestType>
  <Request>
    <ssn>217165454</ssn>
    <dob>03/20/1962</dob>
    <signedDate>10/01/2001</signedDate>
  </Request>
</EPNRequestRoot>
```

3.7.4.4 Regression Test Cycle

Test Description

This test re-executes the test executed in the normal test cycle and expected error test cycle. The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. The data format sent to MQSI is as follows.

XML Input String #1 (Correct Information)

```
<?xml version="1.0" standalone="yes"?>
<EPNRequestRoot>
  <requestType>ref</requestType>
  <Request>
    <ssn>217165454</ssn>
    <dob>03/20/1962</dob>
    <signedDate>10/01/2001</signedDate>
  </Request>
</EPNRequestRoot>
```

XML Input String #2 (Incorrect information)

```
<?xml version="1.0" standalone="yes"?>
<EPNRequestRoot>
  <requestType>ref</requestType>
  <Request>
    <ssn>123456789</ssn>
    <dob>03/20/1962</dob>
    <signedDate>10/01/2001</signedDate>
  </Request>
</EPNRequestRoot>
```

3.7.5 LO System – (eMPN) Test Scenario Expected Results

The execution of the test scenario returns an XML document that contains the result from MQSI and/or the custom adapter. The following XML strings are formatted with line breaks and indentions for easy viewing in this document. They are actually returned as single strings, without such formatting.

3.7.5.1 Expected Results for Normal Test Cycle

The results from successfully processing by MQSI and the custom adapter are listed below.

XML Input String #1

```
<?xml version="1.0"?>
<EPNResponseRoot>
  <result>Success</result>
  <request>Summary</request>
  <message/>
  <Response>
    <ssn>217165454</ssn>
    <dob>03/20/1962</dob>
    <mpnId>217165454N02G90139101</mpnId>
    <signedDate>10/01/2001</signedDate>
    <statusCode>0</statusCode>
    <ReferenceData>
      <internalId>74</internalId>
      <schoolCodeType>G</schoolCodeType>
      <schoolNumber>95154</schoolNumber>
      <Reference>
        <Name>
          <firstName>JIM</firstName>
          <middleInitial></middleInitial>
          <lastName>MOORE</lastName>
        </Name>
        <Address>
          <street1>333 REFERENCE AVENUE</street1>
          <street2>APT. R1</street2>
          <city>REFERENCE1VILLE</city>
          <state>NY</state>
          <zipcode>33333</zipcode>
        </Address>
        <phone></phone>
        <relationship></relationship>
      </Reference>
      <Reference>
        <Name>
          <firstName>JANE</firstName>
          <middleInitial></middleInitial>
          <lastName>VESTER</lastName>
        </Name>
        <Address>
          <street1>4444 REFERENCE AVENUE</street1>
          <street2>APT. R2</street2>
          <city>REFERENCE2VILLE</city>
          <state>NJ</state>
          <zipcode>44444</zipcode>
        </Address>
        <phone></phone>
        <relationship></relationship>
      </Reference>
    </ReferenceData>
  </Response>
</EPNResponseRoot>
```

XML Input String #2

```
<?xml version="1.0"?>
<EPNResponseRoot>
  <result>Partial Failure</result>
  <request>Summary</request>
  <message/>
  <Response>
    <ssn>217165454</ssn>
    <dob>03/20/1962</dob>
    <signedDate>10/01/2001</signedDate>
    <statusCode>100</statusCode>
    <message>No records found.</message>
  </Response>
</EPNResponseRoot>
```

3.7.5.2 Expected Results for Expected Error Test Cycle

Test #1

```
<EMPN_ERROR>
  Caught an IOException: Connection Refused
</EMPN_ERROR>
```

Test #2

```
<EMPN_ERROR>
  Caught an IOException: Connection Refused
</EMPN_ERROR>
```

3.7.5.3 Expected Results for Unexpected Error Test Cycle

The unsuccessful processing is caused by a MQSeries failure that makes all MQSeries objects unavailable. The adapter cannot send a response back to the source application because MQSeries is not operational, thus the application will receive a timeout.

3.7.5.4 Expected Results for Regression Test Cycle

The results from successfully processing by MQSI and the custom adapter are listed below.

XML Output String #1 (Normal Test Cycle)

```
<?xml version="1.0"?>
<EPNResponseRoot>
  <result>Success</result>
  <request>Summary</request>
  <message/>
  <Response>
    <ssn>217165454</ssn>
    <dob>03/20/1962</dob>
    <mpnId>217165454N02G90139101</mpnId>
    <signedDate>10/01/2001</signedDate>
    <statusCode>0</statusCode>
    <ReferenceData>
      <internalId>74</internalId>
      <schoolCodeType>G</schoolCodeType>
      <schoolNumber>95154</schoolNumber>
      <Reference>
        <Name>
          <firstName>JIM</firstName>
          <middleInitial></middleInitial>
          <lastName>MOORE</lastName>
        </Name>
        <Address>
          <street1>333 REFERENCE AVENUE</street1>
          <street2>APT. R1</street2>
          <city>REFERENCE1VILLE</city>
          <state>NY</state>
          <zipcode>33333</zipcode>
        </Address>
        <phone></phone>
        <relationship></relationship>
      </Reference>
      <Reference>
        <Name>
          <firstName>JANE</firstName>
          <middleInitial></middleInitial>
          <lastName>VESTER</lastName>
        </Name>
        <Address>
          <street1>4444 REFERENCE AVENUE</street1>
          <street2>APT. R2</street2>
          <city>REFERENCE2VILLE</city>
          <state>NJ</state>
          <zipcode>44444</zipcode>
        </Address>
        <phone></phone>
        <relationship></relationship>
      </Reference>
    </ReferenceData>
  </Response>
</EPNResponseRoot>
```

XML Output String #2 (Normal Test Cycle)

```
<?xml version="1.0"?>
<EPNResponseRoot>
  <result>Partial Failure</result>
  <request>Summary</request>
  <message/>
  <Response>
    <ssn>217165454</ssn>
    <dob>03/20/1962</dob>
    <signedDate>10/01/2001</signedDate>
    <statusCode>100</statusCode>
    <message>No records found.</message>
  </Response>
</EPNResponseRoot>
```

Test #1 (Expected Error Test Cycle)

```
<EMPN_ERROR>
  Caught an IOException: Connection Refused
</EMPN_ERROR>
```

Test #2 (Expected Error Test Cycle)

```
<EMPN_ERROR>
  Caught an IOException: Connection Refused
</EMPN_ERROR>
```

- This LO System – eMPN test scenario was executed by the Release 2 EAI Core team and the expected results were received and validated.

3.8 EAI Component Test for LO System – Promissory Note Imaging (P-Note Imaging)

The P-Note Imaging system contains information relating to master promissory notes that have been submitted in written form. It is a custom system that runs on the Loan Origination Imaging Server. An application running on the LO Imaging Server will invoke, via a socket connection, a program running on the LO System. This application in turn, will invoke the EAI P-Note Imaging adapter to transmit the

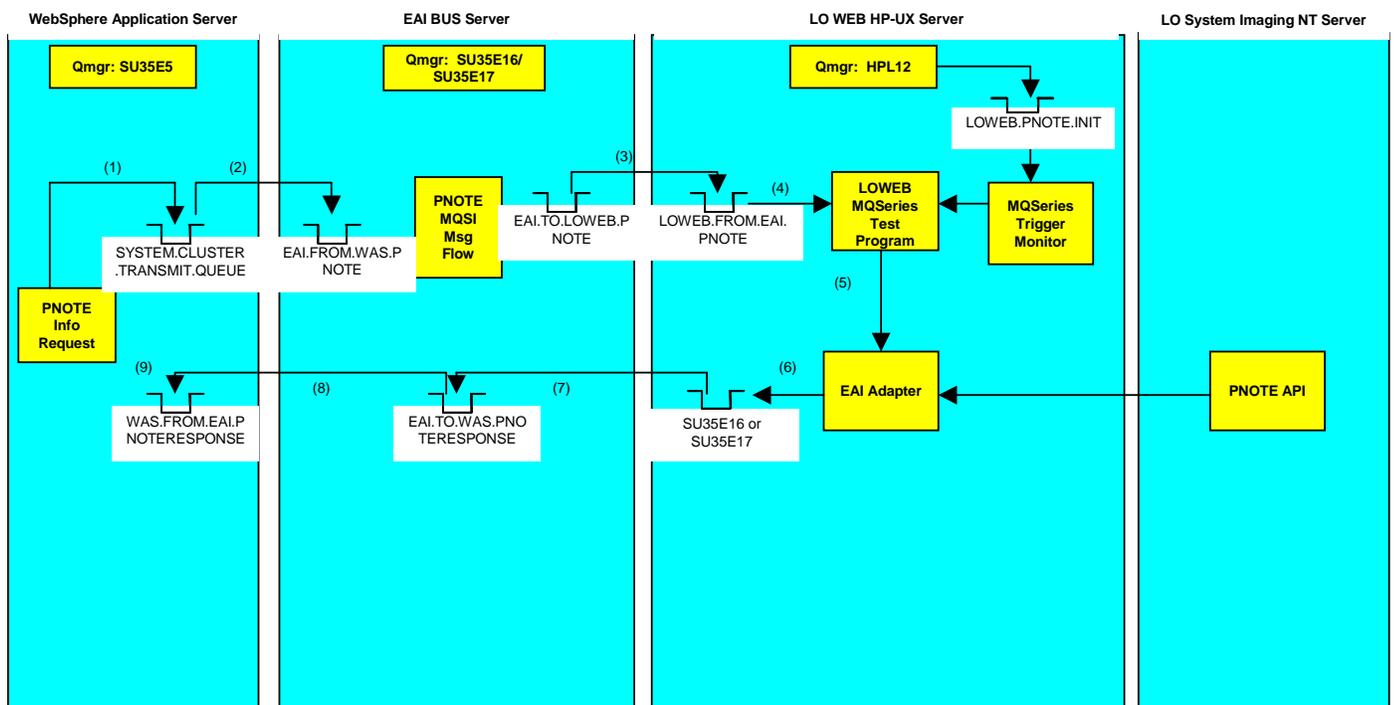
Imaging information. Since the applications that invoke the EAI adapter have not been completely developed, the EAI Core team has developed a test harness to run the EAI P-Note Imaging adapter.

3.8.1 LO System –(P-Note Imaging) Test Scenario Description

This test will demonstrate the functionality of MQSeries messaging across disparate systems, the use of a MQSeries Integrator Message flow, and the use of an MQSeries adapter written for P-Note Imaging.

The application to be used for the EAI P-Note Imaging test is a transmission of test data. A valid XML string will be sent from the WebSphere Application Server to the EAI bus. In the MQSI message flow, the message will be transformed by adding a host name and port number. The message will then be sent to the LO System, where a test program will read from the queue. The test program will take the input data, modify it slightly, and invoke the P-Note Imaging adapter put the message on the queue to be sent to the designated queue.

3.8.2 LO System – (P-Note Imaging) Test Scenario Detailed Design Description



1. A valid XML string is sent from the WebSphere Application Server (WAS) to the EAI bus. (Shown by 1 & 2 in diagram above.)
2. In the MQSI message flow, the message will be transformed by appending a host name and port number. (Shown by box in between (2) and (3) above.)
3. The message will then be sent to the LO Web Server, where a test program will read from the queue and invoke the EAI Adapter. (Shown by 4, 5 above)

- The adapter will then build an output message and put the message on the queue to be sent back to the WebSphere Application Server. (Shown by 6, 7, 8, 9 above)

Files

The following are the files used in conjunction with the P-Note Imaging adapter.

File Specification	Function
hpl12:/dev2/users/bmalki01/pnote/pnote.sh	A shell script used to set up environment variables and then run the adapter.
hpl12:/dev2/users/bmalki01/pnote/PNOTE.class	The java class that contains the adapter.
hpl12:/dev2/users/bmalki01/pnote/EAI.class	Part of the P-Note Imaging adapter.
hpl12:/dev2/users/bmalki01/pnote/EAILog.class	Part of the P-Note Imaging adapter.
hpl12:/dev2/users/bmalki01/pnote/EAIException.class	Part of the P-Note Imaging adapter.
hpl12:/dev2/users/bmalki01/pnote/IEAI.class	Part of the P-Note Imaging adapter.
Hpl12:/dev2/users/bmalki01/pnote/pnote.xml	The MQSeries Application Messaging Interface (AMI) repository.
hpl12:/export/home/mqm/LOWEB.txt	A file containing the definitions of the MQSeries objects on the LO Web Server
su35e16:/export/home/mqm/SU35E16.txt	A file containing the definitions of the MQSeries objects on su35e16.
su35e17:/export/home/mqm/SU35E17.txt	A file containing the definitions of the MQSeries objects on su35e17.
hpl12:/dev2/users/bmalki01/pnote/pnoteTestInput.xml	An XML file that is input to the P-Note test program.

Adapter

The adapter resides in the file /dev2/users/bmalki01/EAI.class. It is invoked when a calling java program creates an object of class EAI and then invokes the sendDatagram method with two parameters. The first parameter is the name of an MQSeries AMI service name that indicates the destination queue manager and queue that the message is to be sent to. The second parameter is a String that contains the message to be sent.

MQSI Objects

Node	Type	Description/Function
Input Message From WAS	MQInput	Gets message from queue EAI.FROM.WAS.PNOTE
Trace1	Trace	Traces input message
Verify Request Type	Compute	Validate message contents
Forward Request to LOWEB	MQOutput	Sends a message to LOWEB
Build Error Message	Compute	Builds an error message
Send Error Back To Requestor	MQOutput	Sends a message back to the requestor based on the reply to information

3.8.3 LO System – (P-Note Imaging) Test Scenario Dependencies

The following defines the dependencies and resource requirements for a successful P-Note Imaging test.

- MQSeries running on logically referred to as SU35E5, SU35E16 or SU35E17, and HPL12.
- MQSI must be operational on either of the following systems, logically referred to as SU35E16 or SU35E17.

3.8.4 LO System – (P-Note Imaging) Test Scenario Inputs

From the initial entry on the WebSphere Application Server through MQSI and onto the adapter, each component is expecting the data a certain format. To validate each component, 4 test cycles were used for testing: Normal testing, expected error testing, unexpected error testing, and regression testing. The results of each test cycle are shown in Section 3.8.5.

3.8.4.1 Normal Test Cycle

Test Description

The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. The data format sent to MQSI is as follows:

```
<PNOTE_TEST>  
    This is test data  
</PNOTE_TEST>
```

3.8.4.2 Expected Error Test Cycle

Since the purpose of this adapter is to send information in one direction, there can be no error result. This is due to the fact that if an error could be transmitted across the bus, the original information could have been transmitted instead and an error result will not be returned.

3.8.4.3 Unexpected Error Test Cycle

Test Description

The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. While data is being processed by the custom adapter, two unexpected errors are simulated. The first simulates a MQSeries failure that makes all MQSeries objects unavailable after the data has been processed within P-Note Imaging but before a response has been sent to the source application, and the second simulates an P-Note Imaging failure that does not provide a response back to the adapter. The data format sent to MQSI is as follows:

Simulate an MQSeries failure by shutting down the queue manager on the LO System.

```
<PNOTE_TEST>  
    This is test data  
</PNOTE_TEST>
```

3.8.4.4 Regression Test

Test Description

This test re-executes the test executed in the normal test cycle and expected error test cycle. The test data is in the specific format needed by MQSI to create the correct data format expected by the custom adapter. The data format sent to MQSI is as follows:

```
<PNOTE_TEST>  
    This is test data  
</PNOTE_TEST>
```

3.8.5 LO System – (P-Note Imaging) Test Scenario Expected Results

The execution of the test scenario returns an XML document that contains the result from MQSI and/or the custom adapter.

3.8.5.1 Expected Results for Normal Test Cycle

The results from successfully processing by MQSI and the custom adapter are listed below.

```
(<PNOTE_TEST>  
    This is test data  
</PNOTE_TEST>)
```

3.8.5.2 Expected Results for Expected Error Test Cycle

There are no expected errors.

3.8.5.3 Expected Results for Unexpected Error Test Cycle

Test #1

Data will not be received at the WAS Server. As this is a one way transmission, it is the job of the sending application to ensure that data has arrived.

3.8.5.4 Expected Results for Regression Test Cycle

The following is the result of the test:

```
(<PNOTE_TEST>  
    This is test data  
</PNOTE_TEST>)
```

- This LO System – P-Note Imaging test scenario was executed by the Release 2 EAI Core team and the expected results were received and validated.

3.9 EAI Component Test for National Student Loan Data System (NSLDS) – Batch

The NSLDS system serves as a data repository for all loan application data within SFA. The EAI enablement of the NSLDS system will allow applications to query the NSLDS system for real-time results, as well as provide the functionality to execute batch processes using MQSeries messaging as the transport mechanism of data from an external system to NSLDS.

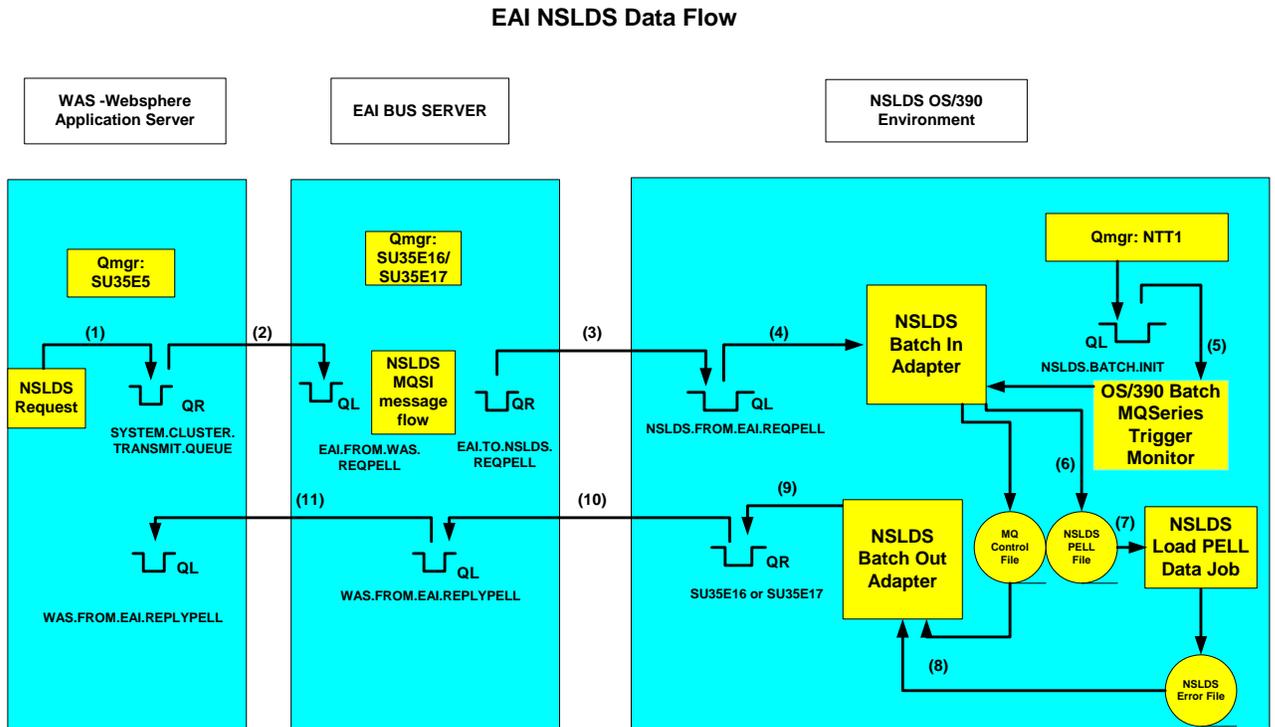
3.9.1 NSLDS – Batch Test Scenario Description

This test will demonstrate the functionality of the MQSeries messaging infrastructure between the test application and the NSLDS mainframe system, the use of a MQSeries Integrator Message flow, and the use of MQSeries adapters written for the NSLDS system.

The Pell Grant request application is used for the EAI NSLDS Batch core test. In this test scenario, an initial control file is sent through the SFA EAI Core architecture that defines the number of records that will be sent to execute the batch job. Following the control record, a data record is sent which contains the application data to be processed. Once all messages are received the MQ Adapters initiate the batch processing on the NSLDS system. The output of the sample batch program is an output file of invalid Pell Grant records, which are returned to the test application and displayed.

The control record is transformed through MQSI, which determines the message type, control or data records. After transformation, the message is sent to NSLDS Batch, the target system, where an adapter will read from the queue and output the data to a flat file. The flat file is input to an existing NSLDS Batch job ARB6200 that reads in the Pell Grant request records and processes the records through numerous application edits. The Pell Grant request records that fail an edit get written out to a flat file. The second MQ adapter puts the messages on the queue to be sent back to the originating system displaying the error file.

3.9.2 NSLDS – Batch Test Scenario Detailed Design Description



The application to be used for the EAI NSLDS Batch core test is the Pell Grant request.

5. A data file, consisting of control and data records, is sent from the WebSphere Application Server (WAS) to the EAI bus. (Shown by 1 & 2 in diagram above.)
6. In the MQSI message flow, the message will be transformed by evaluating the message data, control or data records. (Shown by box in between (2) and (3) above.)
7. The message will then be sent to the NSLDS Batch system where an adapter will read from the queue and output the data to a PELL flat file that can serve as input to an existing NSLDS Batch job (ARB62000). (shown by 4, 5, 6 above)
8. An existing NSLDS Batch job reads the PELL file, processing each PELL record thru edit routines, records with errors are written to an ERROR file. (shown by 7)
9. Another adapter will read the Error file, build an output message and put the message on the queue to be sent back to the WebSphere Application Server. (Shown by 8, 9, 10 and 11).

Files

Files used on the NSLDS system.

File Specification	Function / Description
MQADM1.A.JCLLIB(CMPJCL02)	Job to compile Adapter programs: NSBATCH1 and NSBATCH2
MQADM1.A.JCLLIB(JOBCARD)	Jobcard definition executed by Trigger monitor
MQADM1.A.JCLLIB(MQCKTIBA)	Trigger monitor job, submitted once and continues to run in the background and executes the batch process when notified by the initiation queue that a message has arrived on the queue
MQADM1.A.JCLLIB(MQCKTIEN)	Stop the Trigger monitor from running in the background
MQADM1.A.PROCLIB(PELL)	The proc for the EAI NSLDS batch process
MQADM1.A.PROCLIB(PRB62000)	The proc for the existing NSLDS batch process
MQADM1.A.SRCLIB(CKTIBAT2)	Source for Trigger monitor process
MQADM1.A.SRCLIB(CKTIEND)	Source for Trigger monitor end process
MQADM1.A.SRCLIB(NSBATCH1)	Source for first NSLDS batch adapter – reads from inbound queue
MQADM1.A.SRCLIB(NSBATCH2)	Source for second NSLDS batch adapter – writes to outbound queue
MQM.V5R2M0.*	Contains MQSeries installed product libraries (sample programs, copybooks etc)
MQADM2.PELLNS1.FILE	Input File to PELL Load process
MQADM2.CNTLNS1.FILE	Control file created by NSBATCH1
MQADM2.ERROR.FILE	Error file created by PELL Load process
MQADM2.ERROR.FILE	Error file input to NSBATCH2
MQADM2.CNTL.FILE	Control file input to NSBATCH2

Adapters

The EAI Core validation of the NSLDS batch functionality required the development of custom adapters to execute a batch process on NSLDS. The adapters developed were titles to reflect the NSLDS system and provide the functionality to build an input file, and process the NSLDS batch job, and then send an output file back to the calling application.

There are 2 adapters on the NSLDS system – NSLDS Batch. NSBATCH1 and NSBATCH2 are Cobol programs. The NSBATCH1 adapter reads messages from a MQSeries queue and writes the messages to a file. The NSBATCH2 adapter reads data from a file and puts the data as a message to a MQSeries queue.

NSBATCH1: This program reads messages off an inbound queue, parses the messages to extract PELL records and writes the PELL records to a flat file.

ARB62000: Executes Procedure: ARB62000 This is an existing multi-step process that does edit checks against the PELL data and creates a flat file of exception/error records for those Pell records that fail the edits.

NSBATCH2: This program reads in the exception/error file created in ARB62000 and builds outbound messages stringing multiple exception/error records and puts the messages to the outbound reply queue.

OS390 Batch Trigger Monitor

The Trigger Monitor allows a batch application process to be submitted automatically when a message arrives on an inbound application queue.

When the Pell message arrives from the test application the Queue manager, NTT1 puts the message to the inbound application queue, NSLDS.FROM.EAI.REQPELL.

Since the inbound queue is defined for triggering 'first' Y the Queue Manager will put the data defined in the Process definition: NSLDS.PELL.PROCESS as a message to the initiation queue, NSLDS.BATCH.INIT, when the first message arrives in the queue.

The Batch Trigger Monitor monitors the initiation queue and when a message arrives the Batch Trigger monitor executes the batch process and supplies the queue manager name and queue name to the adapter, NSBATCH1.

Process Definition supplies the following,

1. proc stmt to be applied to the jobcard info
2. placeholders for the queue manager to put the queue manager name and name of the application queue

Inbound queue defined for triggering,

1. Tells the queue manager to trigger a process
2. Supplies the name of the process definition to the queue manager

To run the developed test programs,

In the NSLDS Test TSO environment

Edit: MQADM1.A.JCLLIB(MQCKTIBA)

Submit

This will submit a job: MQCKTIBA (Batch Trigger Monitor) that will continue to run in the background and execute the NSLDS Batch process as explained above.

NSBATCH1

One control message is sent from WAS with each PELL file. The control message contains information to tell the program NSBATCH1 how to process the PELL records off the inbound queue.

CM-MSG-CNT – tells how many data messages to process.

CM-MAXMSG-LEN – tells the maximum size of a message. The application knows based on this value at what offset to stop processing data from the message buffer.

CM-TOTAL-REC-CNT– tells when to stop parsing PELL records from all messages.

CM-LRECL – tells the length of the records to be parsed.

NSBATCH1 performs an initial read to get the first message, i.e. the control message. Using the values from the control message, the adapter pulls the messages from the inbound queue until the msg-count equals CM-MSG-CNT. For each message, the adapter parses the PELL records from the message for a length of CM-LRECL and writes the records to the PELL output file until CM-TOTAL-REC-CNT or CM-MAXMSG-LEN is reached.

NSBATCH1 writes out the control message to a control file to be used by the NSBATCH2 program.

Inputs: The program expects 2 messages as input,

- 1) Control message:

- Record length: 177 bytes character format
- Purpose of input parameter is to tell the adapter,
 1. Number of data messages
 2. Max data message length

3. Number of records in data message
 4. Length of data records
- 2) Data message

Pell Grant request records
Record length: 300 bytes character format

Outputs: A flat file is generated as output. It's a file of Pell request records that failed the required edits.

NSBATCH2

The control file created in NSBATCH1 contains information to tell the program NSBATCH2 how to build the messages for the put to the outbound queue.

CM-MAXMSG-LEN – tells when to stop reading error records and stringing the record to out an outbound message.

CM-LRECL – tells the length of the records to be strung together

NSBATCH2 reads the control file once (only one record) and moves values to Working Storage. The program then reads the error file and moves each record out to the outbound message buffer by the length of CM-LRECL until CM-MAXMSG-LEN or end-of-file condition on the Error file. On completion of all record processing the message are put to the outbound replyto queue.

Prior to using the NSLDS batch adapters, NSBATCH1 and NSBATCH2, will need to be compiled. The following steps should be executed to perform the compilation:

MQADM2.A.JCLLIB(CMPJCL02) specify which program needs be compiled.

Inputs: The program expects 1 file as input.

File description: Errors generated from processing Pell requests through the existing NSLDS batch job: ARB62000.

The filename is limited to a maximum length of 109 characters.

The purpose of input file is to send back Pell requests that are in error back to the test application.

Outputs: Message is written to a MQSeries queue.

MQSI

The MQSI nodes and their function are documented below.

Node Name	Node Type	Function
NSLDS PELL file transfer process	MQInput	Read from input queue
Verify PELL record	Filter	Filter out control message to transform.
Non- control messages	MQOutput	Write to queue if message is not a control message.
Unkown filter error	MQOutput	Write to replyto queue if unknown filter error
Build PELL Output Message	Compute	Take XML as input and create MRM. Append required data fields to message Set data fields to default values
Build Error Message	Compute	Build error message for invalid PELL record
Error Reply	MQReply	Put error message on replyto queue
NSLDS Queue	MQOutput	Put message on queue for delivery to NSLDS
Trace on Input	Trace	Failure on queue input goes to trace node
Trace on Build	Trace	Failure to build message goes to trace node

3.9.3 NSLDS - Batch Test Scenario Dependencies

The following defines the dependencies and resource requirements for a successful NSLDS Batch test.

- MQSeries running on logically referred to as SU35E5, SU35E16 or SU35E17, and NTT1
- MQSI must be running on either logically referred to as SU35E16 or SU35E17.

Test Data

The NSLDS Batch Pell Request Application expects two (2) input records. The first record is a Control Message, which defines the number of records to expect, and the second record is the actual input data. The control record has the following format,

Detailed Record Layout/ Control Record:

```
01 NSLDS-CNTL-AREA .
    03 CM-MSG-CNT          PIC X(4) .
    03 CM-MAXMSG-LEN      PIC X(9) .
    03 CM-TOTAL-REC-CNT  PIC X(8) .
    03 CM-LRECL          PIC X(4) .
    03 CM-NAME-FILE-IN   PIC X(48) .
    03 CM-NAME-FILE-OUT  PIC X(48) .
    03 CM-NAME-JOB-FILE  PIC X(48) .
    03 CM-NAME-JOB-MEMBER PIC X(8) .
```

Following the control message is the Pell record. The Pell input record and ERROR output records are treated as unformatted data. The Pell record length equals 300 bytes (spaces are not visible) and the last characters at the end of record are '2001'.

The message queue does not manipulate the individual fields and only defines the field at the record level.

The following are the record structures for the Pell and error records,

01 PELL-RECORD.
03 PIC X(300).

01 ERROR-RECORD.
03 PIC X(109).

PELL Record:

01 NSLDS-PELL-REC.
03 PE-CURR-SSN PIC X(9).
03 PE-DOB PIC X(8).
03 PE-FNAME PIC X(12).
03 PE-RPTING-OPEID PIC X(8).
03 PE-ATTENDING-OPEID PIC X(8).
03 PE-NEW-CURR-SSN PIC X(9).
03 PE-NEW-DOB PIC X(8).
03 PE-NEW-FNAME-CD PIC X(12).
PE-NEW-RPTING-OPEID PIC X(8).
03 PE-NEW-ATTENDING-OPEID PIC X(8).
03 PE-LNAME PIC X(35).
03 PE-MI PIC X(1).
03 PE-ORIG-SSN PIC X(9).
03 PE-NAME-CD PIC X(2).
03 PE-PELL-BATCH-CD PIC X(26).
03 PE-DISBURS-REF-NUM PIC X(2).
03 PE-FILLER PIC X(4).
03 PE-EXP-FAMILY-CONTR PIC X(5).
03 PE-SEC-FAMILY-CONTR PIC X(5).
03 PE-ACC-SEC-EFC-USED-CD PIC X(1).
03 PE-ACC-COST-ATTEND PIC X(7).
03 PE-HIGHEST-COST-ATTEND PIC X(7).
03 PE-SCHED-FED-PELL-AMT PIC X(7).
03 PE-ACC-AMT-PAID-TO-DT PIC X(7).
03 PE-ACC-ORIGIN-AMT PIC X(7).
03 PE-ACC-FIRST-ENROLL PIC X(8).
03 PE-ACC-ACAD-CALEN PIC X(1).
03 PE-ACC-VERIF-STAT-CD PIC X(1).
03 PE-ACC-PMT-METHOD PIC X(1).
03 PE-ACC-ENROLL-STATUS-CD PIC X(1).
03 PE-HIGH-ACC-ENROLL-STAT PIC X(1).
03 PE-ACC-CR-HRS-CR-COMPL PIC X(4).
03 PE-ACC-HRS-CR-SCH-ACAD PIC X(4).
03 PE-ACC-WEEKS-ENROLL PIC X(2).
03 PE-ACC-WEEKS-IN-PGM PIC X(2).
03 PE-HIGH-ACC-INCAR-REC-CD PIC X(1).
03 PE-TOTL-ELIG-USED PIC X(5).
03 PE-SEG-ELIG-USED PIC X(5).

```

03 PE-SEG-POT-OVRAW-FLAG      PIC X(1) .
03 PE-SEG-POT-OVRAW-DIS      PIC X(7) .
03 PE-RECVD-DT                PIC X(8) .
03 PE-DISBURS-DT              PIC X(8) .
03 PE-UNUSED                  PIC X(18) .
03 PE-ACTION-CD                PIC X(1) .
03 PE-PELL-AWARD-YR           PIC X(4) .
  
```

Cobol Application Specifics

Prior to executing the test the NSLDS batch adapters must first be compiled. To compile the NSBATCH1 and NSBATCH2 programs,

```

In the NSLDS Test TSO environment
edit MQADM1.A.JCLLIB(CMPJCL02)
submit
  
```

MQSeries Objects Used

The MQSI nodes and their function are documented below.

Object Name	Object Type	Description
SU35E16	Local Queue	Local queue defined as transmit queue. Used when sending data from NTT1 to SU35E16.
SU35E17	Local Queue	Local queue defined as transmit queue. Used when sending data from NTT1 to SU35E17.
NSLDS.DEAD.LETTER.QUEUE	Local Queue	Local queue defined as the system dead letter queue. Message is placed on the queue when it is undeliverable.
NSLDS.BATCH.INIT	Local Queue	Queue used as an initiation queue for the PELL request application
NSLDS.FROM.EAI.REQPELL	Local Queue	Local queue used to receive data as input for PELL request application
NTT1.SU35E16	Channel	Sender Channel. Used to send messages to SU35E16
NTT1.SU35E17	Channel	Sender Channel. Used to send messages to SU35E17
SU35E16.NTT1	Channel	Receiver Channel used to receive messages from SU35E16
SU35E17.NTT1	Channel	Receiver Channel used to receive messages from SU35E17
NSLDS.PELL.PROCESS	Process	Process defined for PELL request application. This is the process which gets triggered when a message arrives on the NSLDS.FROM.EAI.REQPELL queue.
TRIGGER.CHANNEL	Process	Process defined to run the sender channel

3.9.4 NSLDS – Batch Test Scenario Inputs

The test data must be in a specific format. From the initial entry on the WebSphere Server through MQSI and onto the adapter, each component is expecting the data a certain way. The data format is as follows.

3.10 EAI Component Test for National Student Loan Data System (NSLDS) – Cool:Gen

The NSLDS system provides the capability to interface with the NSLDS web server and allow users to perform queries against the NSLDS mainframe system. This capability is provided via the Cool:Gen product and its use of a COM proxy server. The implementation of the EAI Bus to provide access from the NSLDS web server to the NSLDS mainframe system will provide messaging capabilities directly from the NSLDS web server to the NSLDS mainframe system using the EAI Bus instead of the Cool:Gen Com proxy server and SNA to TCP/IP conversion for each request.

3.10.1 NSLDS – Cool:Gen Test Scenario Description

The NSLDS Cool:Gen test demonstrates the functionality of using Cool:Gen with IBM's MQSeries messaging to perform the following:

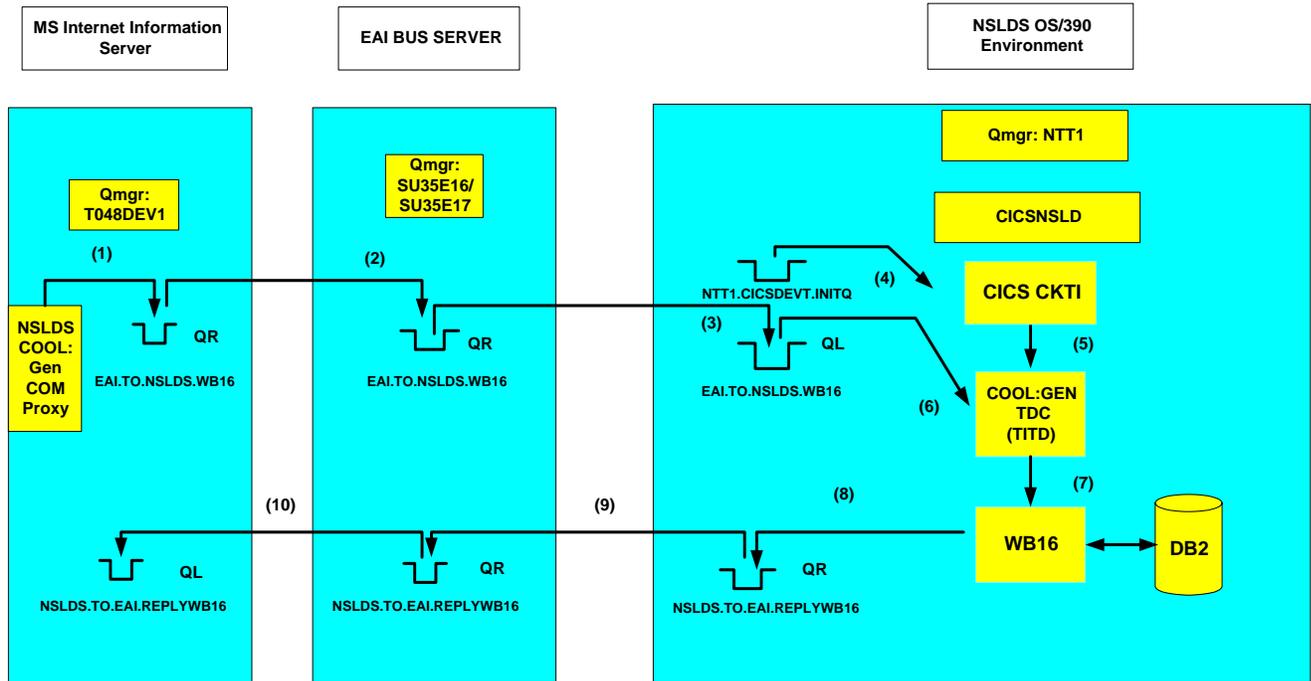
- Demonstrate the capability of the Cool:Gen adapters, residing on both the NSLDS web server and the NSLDS mainframe, and the Cool:Gen COM Proxy adapter on the NSLDS Web Server to communicate with the Cool:Gen Server adapter, on the NSLDS mainframe, via MQSeries messaging.
- Demonstrate the interoperability of MQSeries messaging to route Cool:Gen messages across disparate systems, the NSLDS NT web server and the NSLDS IBM 9672 Mainframe.
- Demonstrate the capability of the MQSeries CICS trigger monitor and the Cool:Gen Transaction Dispatcher for CICS (TDC) to automatically invoke Cool:Gen developed CICS transactions. The trigger monitor is shown in step 5 in the diagram below.

3.10.2 NSLDS – Cool:Gen Test Scenario Detailed Design Description

The sample representative NSLDS transaction used for the EAI Cool:Gen sample function test is the NSLDS Organization contact list inquiry (WB16).

The NSLDS message flow uses MQSeries messaging to connect the NSLDS web server, running Microsoft Internet Information Server (MS IIS), to the NSLDS OS/390 Mainframe via the EAI Bus Servers. The Cool:Gen COM Proxy residing on the NSLDS web server initiates the requested transaction to the Cool:Gen Server. The MQ Cool:Gen adapter on the NSLDS web server puts the message data into a message queue and the message is routed to the NSLDS mainframe system for processing. As soon as the message arrives on the NSLDS mainframe input queue, the CICS trigger monitor starts the Transaction Dispatcher for CICS (TDC) component of Cool:Gen on the NSLDS OS/390 mainframe. The TDC starts the WB16 transaction to process the request. This connectivity demonstrates the capability of the MQSeries CICS trigger monitor and the Cool:Gen TDC to automatically start and execute a test transaction (WB16) when a message request arrives onto the input queue on the NSLDS System partition where the queue manager resides. The following define the detailed steps in executing the NSLDS transaction sample test:

EAI NSLDS COOL:GEN
 Data Flow



- 1) A Cool:Gen COM Proxy request message, consisting of an Organization Type, Org Code and Org Sub code, is put onto the Remote Queue EAI.TO.NSLDS.WB16 from the NSLDS when server queue manager (T048DEV1).
- 2) The message is routed through the EAI BUS Servers – queue manager (logically referred to as SU35E16 or SU35E17) using the EAI.TO.NSLDS.WB16 message queue.
- 3) The Cool:Gen request message arrives on the EAI.TO.NSLDS.WB16 message queue on the NSLDS OS/390 System queue manager (NTT1). The EAI.TO.NSLDS.WB16 queue is defined and set for triggering.
- 4) For each message arrival, the queue manager (NTT1) creates a trigger message based on the information defined on the PROCESS definition and puts the data into the NTT1.CICSDEVT.INITQ.
- 5) The CICS trigger monitor in the CICS region, CICSNSLD, retrieves the trigger message, examines the message contents and initiates the defined Cool:Gen Transaction Dispatcher for CICS (TDC), passing the entire trigger message to the program.
- 6) The TDC, which opens the application queue, gets the request message.
- 7) The program WB1612DS is invoked which accesses the DB2 databases and formats the reply to be sent back to the COM Proxy on the NSLDS mainframe system. Communication between MQSeries and the CICS program WB1612DS is done thru the CICS COMMAREA.
- 8) The formatted reply message is put into the NSLDS.TO.EAI.REPLYWB16 queue.

- 9) The message is routed through the EAI BUS Servers – queue manager (logically referred to as SU35E16 or SU35E17) using the NSLDS.TO.EAI.REPLYWB16 queue.
- 10) The reply message arrives at the queue manager – T048DEV1 on the NSLDS web server. The Cool:Gen COM Proxy gets the reply message from the queue and display the result.

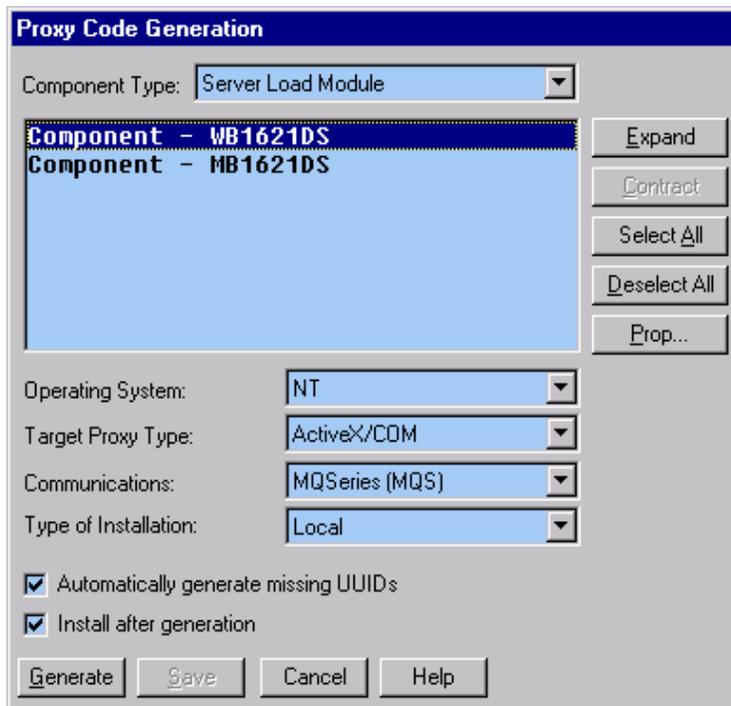
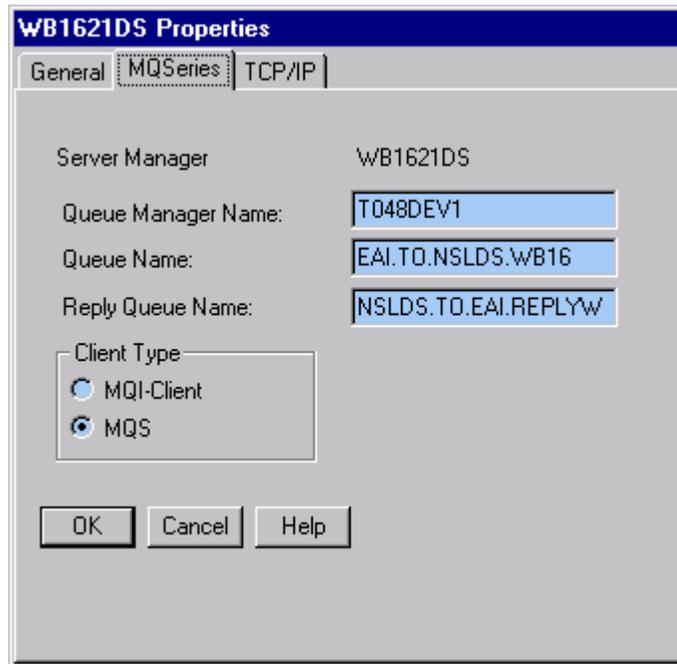
Files

Not applicable with this test.

Adapters

On the NSLDS web server and the NSLDS mainframe the developed MQ Cool:Gen Adapters were used. The NSLDS applications were developed using Cool:Gen v5.1. Both the Cool:Gen COM Proxy and Server are generated on the NSLDS web server. The COM Proxy executes on the NSLDS web server. The Cool:Gen Server is built, bound and deployed to the NSLDS OS/390 platform.

The Cool:Gen development tools were used to create the NSLDS Client (COM Proxy) adapter, the server manager name, the queue manager name, the queue name, the REPLYTO queue name and the client type. The following are screens from the Cool:Gen configuration. The first screen defines the MQSeries queue manager, and queues to the Cool:Gen application. The second screen maps the Cool:Gen CICS transaction to MQSeries for execution during operation of the application.



MQSI

Not applicable with this test; data transformation is not required.

3.10.3 NSLDS – Cool:Gen Test Scenario Dependencies

The following defines the dependencies and resource requirements for a successful NSLDS Batch test.

- MQSeries running on logically referred to as SU35E5, SU35E16 or SU35E17, and NTT1
- MQSI must be running on either logically referred to as SU35E16 or SU35E17.
- CICS 4.1 using the CICS Region CICSDEVT
- Cool:Gen MQSeries Transaction Dispatcher for CICS
- An IBM DB2 database on CICSDEVT must be available.

Test Data

The test data must be in a specific format, as defined in the Cool:Gen application. From the initial entry on the WebSphere Server through MQSI and onto the adapter, each component is expecting the data a certain way. The data format is defined in the following section.

3.10.4 NSLDS – Cool:Gen Test Scenario Inputs

The following data and format is required in the message request from the NSLDS web server.

- Organization Type:
 - SCH School (6 + 4 digits)
 - GA Guarantee Agency (3 digits)
 - LEN Lender (6 + 4 digits)
 - EDR Education Region (2 digits)
 - FDLP Servicer Branch (1 digit)
 - LBS Lender Branch Service (6 + 4 digits)
- Organization Code: Organization Code (6 + 4 digits)
- Organization Sub Code: School or Lender

3.10.5 NSLDS – Cool:Gen Test Scenario Expected Results

A message will be initiated from the Websphere Application Server and be routed through the EAI Bus. The message will be received via MQSeries on the NSLDS mainframe where the message will be extracted from the queue. The trigger monitor will pass the message data to the CICS TDC to execute the WB16 transaction. The results returned from the transactions will be placed on a message queue on the NSLDS mainframe and routed back to the NSLDS web server for display.

The Cool:Gen transaction dispatcher for CICS (TDC) starts the WB16 transaction, passing the request message from MQSeries via the CICS DFHCOMMAREA for processing. The result of the request is a formatted reply, which is wrapped in the MQ message and sent back via a queue

called NSLDS.TO.EAI REPLYWB16. The result is displayed by the Cool:Gen COM Proxy and it contains the organization name, code, type, status, address, city, state, and zip code.

Below is a result of a test with valid input data.

NSLDS ORG CONTACT LIST FIELDS	VALUE
ORG TYPE	SCH
ORG CODE	001002
ORG SUB CODE	00
ORG_NAME	ALABAMA AGRICULTURE & MECHANICAL UNIVERSITY
ORG_STREET_ADDRESS	4900 MERIDIAN STREET NW
ORG_CITY	NORMAL
ORG_STATE	ALABAMA
ORG_ZIP_CODE	35762
ORG_COUNTRY	U.S.A.

- This NSLDS Cool:Gen test scenario was executed by the Release 1 EAI Core team and the expected results were received and validated.

The following picture depicts the results of the execution of the sample transaction on the NSLDS system using the NSLDS web server and the SFA EAI Core Architecture.

Name: ALABAMA AGRICULTURAL & MECHANICAL UNIVERSITY
Code: 00100200 **Type:** School
Status: OPEN
Address: 4900 MERIDIAN STREET NW
 NORMAL, AL 357621357

Organization Contact List

Function	First Name / Last Name	Phone / Ext.	Email
1 IS TECHNICAL ISSUES	MY TEK	(540)555-1212 0001	
2 SSCR ISSUES	PRIMO FANTELLI	8594	
3 FAT/ISIR ISSUES	PRIMO FANTELLI	8594	
4 DEFAULT ISSUES	BRUCE TAYLOR		
5 OVERPAYMENT ISSUES	TEST ME	(111)111-1111 2212	
6 FEDERAL PERKINS ISSUES	PRIMO FANTELLI	8594	
7 FFEL ISSUES	A		
8 DIRECT LOAN ISSUES	PRIMO FANTELLI	8594	
9 PELL GRANT ISSUES	STUFF		
10 COHORT DEFLT RATE ISSUES	PRIMO FANTELLI	8594	
11 PERKINS DATA PROV CONTACT	PRIMO FANTELLI	8594	
12 GUARANTY AGENCY CONTACT	PRIMO FANTELLI	8594	
13 LENDER NSLDS CONTACT	PRIMO FANTELLI	0504	
14 CUSTOMER SVC(BORROWERS)	PRIMO FANTELLI	8594	

3.11 EAI Component Test for Post-Secondary Education Participants System (PEPS)

The PEPS system contains data records to support the Post Secondary Education Processing System. It is a custom developed system built upon the Oracle Forms COTS product. The EAI enablement of the PEPS system will provide the capability to execute real-time transactions against the PEPS database.

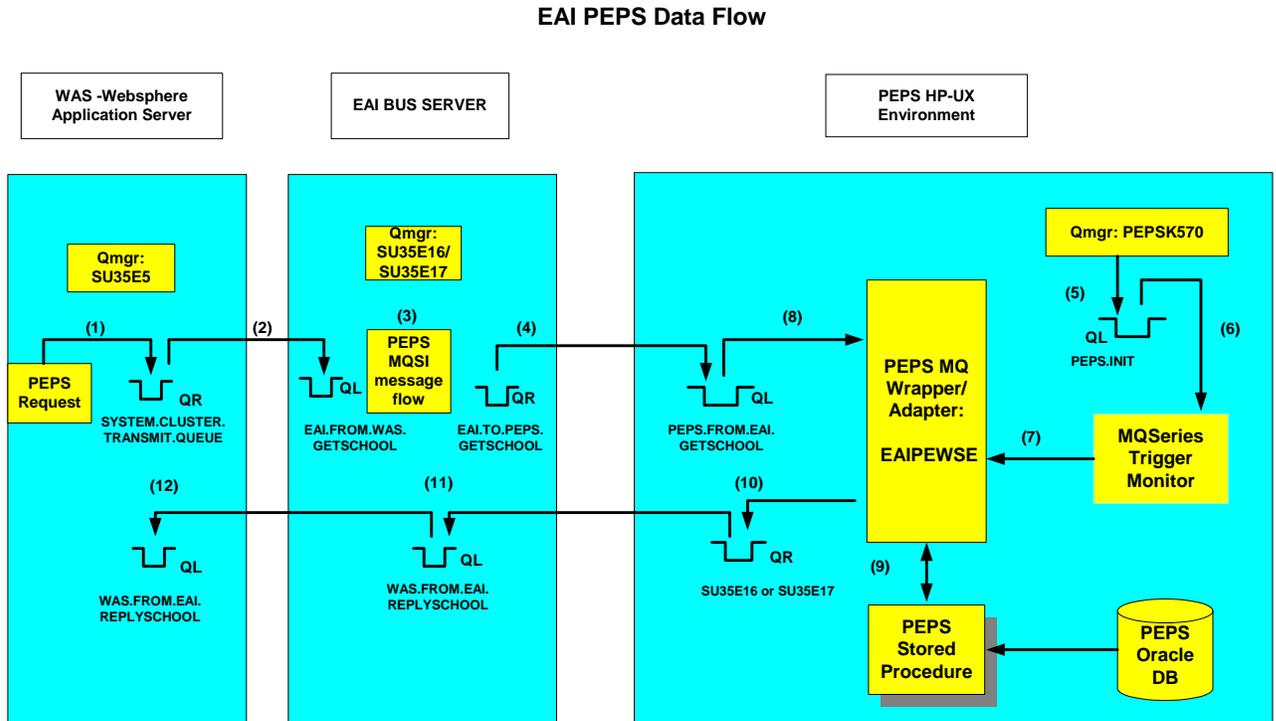
3.11.1 PEPS Test Scenario Description

The EAI Core Architecture test scenario to validate the EAI infrastructure to the PEPS system is based on a School Eligibility request. The PEPS development team provided access to the Oracle database with one stored procedure that demonstrates this typical business process. A MQSeries Java adapter has been developed to connect the PEPS system to the EAI Bus and provides the capability to send a request to the PEPS system to determine the eligibility of a school.

3.11.2 PEPS Test Scenario Detailed Design Description

The sample function selected for the PEPS system validates the ability of a user to enter a School ID as input data from the test application. The school ID input data is put into an MQSeries message and routed to the EAI Bus for transformation by MQSI. The input data is in XML format. MQSI performs the defined message flow transformation and routes the transformed message data to the target system, PEPS. Upon receipt by the PEPS server the custom developed MQ Adapter retrieves the message from the queue, executes the Oracle stored procedure, which performs a lookup in the PEPS Oracle database, and returns the school eligibility status to the test application for display.

The figure below describes the message flow through the PEPS custom adapter.



The flow of a MQSeries Request type message through the EAI PEPS Request Design is as follows:

- 1) A PEPS MQSeries Request type message is put to the Cluster Queue EAI.FROM.WAS.GETSCHOOL from the WAS box.
- 2) The MQSeries Queue Manager (logically referred to as SU35E5) on the WAS moves the message to the Local Queue EAI.FROM.WAS.GETSCHOOL.
- 3) The message is pulled from the EAI.FROM.WAS.GETSCHOOL and processed through the PEPS MQSI Message Flow.
- 4) The output message from the PEPS MQSI Message Flow is put to the Remote Queue EAI.TO.PEPS.GETSCHOOL.
- 5) The MQSeries Queue Manager (PEPSK570) on PEPS puts a trigger message on the initiation queue: PEPS.INIT.
- 6) The MQSeries Trigger Monitor application pulls the trigger message from the PEPS.INIT queue.
- 7) The MQSeries Trigger Monitor application starts the PEPS MQ Wrapper/Adapter application.

- 8) The PEPS MQ Wrapper/Adapter pulls the message from the PEPS.FROM.EAI.GETSCHOOL.
- 9) The PEPS MQ Wrapper/Adapter application calls the PEPS API to pull data from the PEPS Oracle database and pass back the data retrieved.
- 10) The PEPS MQ Wrapper/Adapter puts the PEPS message into the transmission Queue for the logically referred to as SU35E16 or SU35E17.
- 11) The MQSeries Queue Manager (PEPSK570) on PEPS moves the reply message to the Queue WAS.FROM.EAI.REPLYSCHOOL.
- 12) The MQSeries Queue Manager (SU35E16/SU35E17) on the EAI Bus server moves the reply message to the Local Queue WAS.FROM.EAI.REPLYSCHOOL.

Files

The following input files have been defined on the WAS server for access by the EAI Core test application in execution of this test scenario:

File Specification	Function
Peinp1.xml	Returns the eligibility status of the school
Peinp2.xml	Returns the eligibility status of the school
Peinp3.xml	Returns the eligibility status of the school
Peinvld1.xml	Invalid school ID, returns an error

Adapters

A custom MQ Adapter, written in Java, was developed for the PEPS system. The adapter is called MQPEPS.1.1.0.8.

MQSI

The MQSI nodes and their function are documented below.

Node	Type	Description/Function
Input Message Queue From WAS	MQInput	Gets message from queue EAI.FROM.WAS.GETSCHOOL
Trace1	Trace	Traces flow for debug
Determine Request Type	Filter	Checks if requesttype = 1 If tue goto format SQL request If false goto format Oracle call request
Format SQL Request	Compute	Builds message based on the input data and the required parameters for the stored procedure
Output Trace	Trace	Traces flow for debug
Format Oracle Call Request	Compute	Builds message per the format expected by the PEPS stored procedure
Output Queue to PEPS	MQOutput	Puts message to queue EAI.TO.PEPS.GETSCHOOL

3.11.3 PEPS Test Scenario Dependencies

To execute the PEPS System test scenario the following dependencies must be met,

- MQSeries Messaging and queue managers on each of the following systems operational, logically referred to as SU35E5, SU35E16 or SU35E17, and PEPS.

- MQSI must be operational on either logically referred to as SU35E16 or SU35E17.
- Valid school Ids defined in the PEPS database and a tested Oracle stored procedure to query the PEPS database to extract the database data.

3.11.4 PEPS Test Scenario Inputs

The PEPS test scenario included 4 test executions, three valid tests and one invalid test. The results of each test are shown in Section 3.6.5.

Test Data

The test data must be in a specific format. From the initial entry on the WebSphere Server through MQSI and onto the adapter, each component is expecting the data a certain way. The data format is as follows.

Input file: peinp1.xml

```
<?xml version=1.0?>
<pepsRoot>
  <OPEID>00103300</OPEID>
</pepsRoot>
```

Input: peinp2.xml

```
<?xml version=1.0?>
<pepsRoot>
  <OPEID>00102000</OPEID>
</pepsRoot>
```

Input: peinp3.xml

```
<?xml version=1.0?>
<pepsRoot>
  <OPEID>00716400</OPEID>
</pepsRoot>
```

Input: peinvld1.xml

```
<?xml version=1.0?>
<pepsRoot>
  <OPEID>00000000</OPEID>
</pepsRoot>
```


null1212null1212null1291null1291null1291null1291null1212null1212null1212null122nu
11212null1212null1291null1212null10

0 00716400 1 Bryan College of Court Reporting 2 2333 Beverly Boulevard
3 null 4 Los Angeles 5 CA 6 90057 7 2209 8 null 9 null 10 null 11 037
12 09 13

09 14 Y 15 Y 16 C 17 3 18 11 19 QH 20 5 21 ACICS 22 Certified 23
Certified 24 Certified 25 Certified 26 49 27 12 28 1999-10-14
00:00:00.0 29

1997-07-10 00:00:00.0 30 1997-08-01 00:00:00.0 31 2001-06-30 00:00:00.0
32 078799343 33 952312992 34 1998 35 6.40 36 DU 37 Y 38 null 39

007164

Corresponding Output for peinvld1.xml: Test Results of sendReceivePEPS Test Scenario

2oracle.jdbc.driver.OracleDriverjdbc:oracle:oci8:@pepsprodpt16200{call
P_GET_MQ_TEST_DATA(?,
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)}401120000000212null1212null1212null1212null12
12null1212null1212null1212null1212null1212null1212null1212null1212null121
2null1212null1212null1212null1212null1212null1212null1212null1212null1212
null1212null1212null1291null1291null1291null1291null1212null1212null1212null122nu
11212null1212null1291null1212null10

0 00000000 1 null 2 null 3 null 4 null 5 null 6 null 7 null 8 null 9
null 10 null 11 null 12 null 13 null 14 null 15 null 16 null 17 null 18
null 19 null 20 null 21 null 22
null 23 null 24 null 25 null 26 null 27 null 28 null 29 null 30 null 31
null 32 null 33 null 34 null 35 null 36 null 37 null 38 null 39 null

- This PEPS test scenario was executed by the Release 1 EAI Core team and the expected results were received and validated.

3.12 EAI Component Test for Student Aid Internet Gateway (SAIG/bTrade)

The SAIG/bTrade application is a replacement for the existing TIV/WAN interface. TIV/WAN provides the functionality for sending and retrieving messages/data via secure network, Value Added Network (VAN). As part of the SFA Modernization effort, SFA is moving towards a secure Internet file transfer capability to reduce the dependency on external services, i.e. VAN, and utilize the Internet. The SAIG/bTrade application was chosen for Release 1 and 2 of the Core to provide this validation and integration into the EAI Bus to provide this file transfer capability since it has impacts on all SFA systems since it will be a replacement of the TIV/WAN system.

3.12.1 SAIG/bTrade Test Scenario Description

SAIG/bTrade has provided a Java based application connector API to support the retrieval of messages from a SAIG/bTrade mailbox. The EAI Core Architecture team has developed an MQ Adapter to interface with the SAIG/bTrade application, through the SAIG/bTrade connector API, to extract data from a mailbox on the SAIG/bTrade server. The test scenario chosen for the EAI Build and Test will validate the ability to extract message data from a test mailbox configured for the EAI team per the SAIG/bTrade connector API specifications.

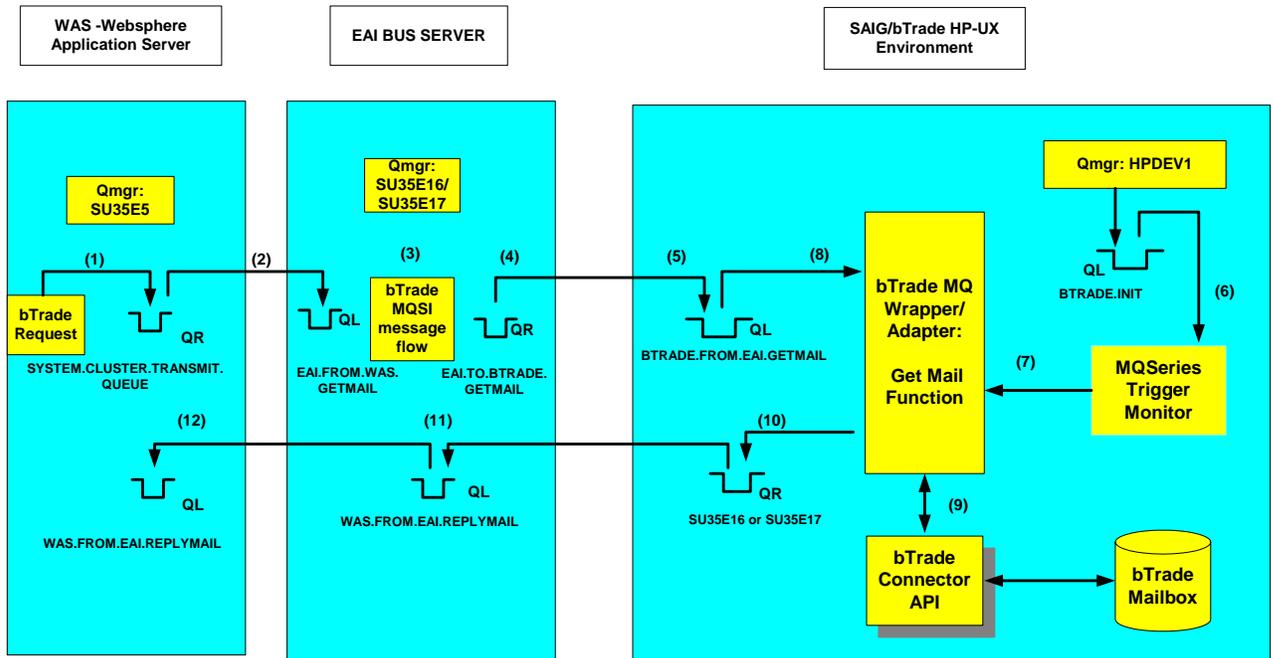
3.12.2 SAIG/bTrade Test Scenario Detailed Design Description

The SAIG/bTrade test scenario starts with a request for mailbox data from the Websphere Application Server. The message is put on a queue and is routed through the EAI Bus for transformation via MQSI. The message is then routed to the SAIG/bTrade system where the MQ adapter reads the message from the queue and calls the SAIG/bTrade connector API with the appropriate parameters. The connector API will check the status of the specified mailbox, retrieve the data from the mailbox and return the results back to the MQ Adapter. The results will be put on a message queue and sent back to the test application

Errors will be reported in the MQ SAIG/bTrade debug output (/home/mqm/btrade/debug.out) and the MQ bTrade XML reply files. (/home/mqm/btrade/temp.'Date'.out).

All application and MQ errors and exceptions are printed to the MQSeries SAIG/bTrade standard error and standard output streams. In addition, all application and MQ errors and exceptions are returned in the MQ reply message.

EAI SAIG/bTrade Data Flow



The flow of a MQSeries Request type message through the EAI SAIG/bTrade Request Design is as follows:

- 1) A bTrade MQSeries Request type message is put to the Cluster queue EAI.FROM.WAS.GETMAIL from the WAS box.
- 2) The MQSeries Queue Manager (logically referred to as SU35E5) on the WAS moves the message to the queue EAI.FROM.WAS.GETMAIL.
- 3) The message is pulled from the EAI.FROM.WAS.GETMAIL and processed through the bTrade MQSI Message Flow.
- 4) The output message from the bTrade MQSI Message Flow is put to the queue EAI.TO.BTRADE.GETMAIL.
- 5) The MQSeries Queue Manager (logically referred to as SU35E16/SU35E17) on the EAI Bus server moves the message to the queue BTRADE.FROM.EAI.GETMAIL and based on the attributes set up in the queue, the MQSeries Queue Manager (HPDEV1) on bTrade puts a trigger message on an initiation queue: BTRADE.INIT.
- 6) The MQSeries Trigger Monitor application pulls the trigger message on the BTRADE.INIT.
- 7) The MQSeries Trigger Monitor application starts the bTrade MQ Wrapper/Adapter application.
- 8) The bTrade MQ Wrapper/Adapter pulls the message from the BTRADE.FROM.EAI.GETMAIL.

- 9) The bTrade MQ Wrapper/Adapter application calls the bTrade Connector API to pull data from a bTrade mailbox and pass back the file/message retrieved.
- 10) The bTrade MQ Wrapper/Adapter puts the bTrade file/message into the Transmission Queue for logically referred to as SU35E16 or SU35E17.
- 11) The MQSeries Queue Manager (HPDEV1) on bTrade moves the reply message to the queue WAS.FROM.EAI.REPLYMAIL.
- 12) The MQSeries Queue Manager (logically referred to as SU35E16/SU35E17) on the EAI Bus server moves the reply message to the queue WAS.FROM.EAI.REPLYMAIL.

Files

The following files will be used to execute the Test Scenario for the bTrade system,

- Btinp1.xml – contains the mailbox name of LINDALOYD
- Btinp2.xml – contains the mailbox name of IBMMQ
- Btinvld.xml – contains an invalid mailbox name of FREDMQ.

Each is an XML file with the required mailbox name and parameters as specified in the bTrade specification document. These files are located in /www/dev/eai/input/ and are presented by the WAS Test application in a drop list on the bTrade web page.

Adapters

This scenario uses the WAS Test application and associated MQ Adapters, and the custom bTrade MQ Adapter, written in Java. The technical specifications for the bTrade adapter are defined in the EAI Technical Specifications–Release 2, Deliverable 54.1.6.

MQSI

The MQSI nodes and their function are documented below.

Node Name	Node Type	Function
Input Message Queue From WAS	MQInput	This node retrieves messages from the flow input queue EAI.FROM.WAS.GETMAIL
Trace1	Trace	This node provides a trace file showing the structure of input messages.
Determine Request Type	Filter	This node routes processing within the flow based on the RequestType. If RequestType = '1' then processing continues along the GetMail branch, otherwise PutMail is assumed.
Put Mail	Compute	This node builds the XML document which will be sent to bTrade, mapping input fields to output fields. The node set request = 2 which indicates PutMail, connectortname to the mailbox ID and request data to the input request data.
Output Queue to bTrade	MQOutput	This message puts messages to the remote queue EAI.TO.BTRADE.GETMAIL which will cause them to be routed to bTrade.
Output Trace	Trace	This node provides a trace file showing the structure of output messages.
Get Mail	Compute	This node builds the XML document which will be sent to bTrade, mapping input fields to output fields. The node set request = 1 which indicates GetMail and connectortname to the mailbox ID.

3.12.3 SAIG/bTrade Test Scenario Dependencies

- Execution of this test scenario requires MQSeries messaging running on all systems through which the messages must travel. In the case of the bTrade test scenario this includes: logically referred to as SU35E5, SU35E16 or SU35E17, bTrade server.
- In addition, MQSI must be running on either SU35E16 or SU35E17.

Test Data

The test data must be in a specific format. From the initial entry on the WebSphere Server through MQSI and onto the adapter, each component is expecting the data a certain way. The data format is as follows:

The following XML schema defines the input and output data layout. The input XML is defined above the dotted line. The output XML is defined above and below the dotted line (i.e. the output is represented in the entire XML schema).

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:annotation>
<xsd:documentation xml:lang="en">
  MQRBTRADE XML schema
</xsd:documentation>
</xsd:annotation>
<xsd:element name="mqrbrtrade" type="mqrbrtradeType"/>
<xsd:complexType name="mqrbrtradeType">
  <xsd:sequence>
    <xsd:element name="mqrequest" type="mqrequestType"/>
    <xsd:element name="mqstatus" type="mqstatusType"
      minOccurs="0" maxOccurs="1"/>
    <xsd:element name="mqdata" type="mqdataType"
      minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="mqrequestType">
  <xsd:sequence>
    <xsd:element name="request" type="xsd:integer"/>
    <xsd:element name="language" type="xsd:string"
      minOccurs="0" maxOccurs="1"/>
    <xsd:element name="customer" type="xsd:string"
      minOccurs="0" maxOccurs="1"/>
    <xsd:element name="genschema" type="xsd:string"
      minOccurs="0" maxOccurs="1"/>
    <xsd:element name="genusername" type="xsd:string"
      minOccurs="0" maxOccurs="1"/>
    <xsd:element name="genpassword" type="xsd:string"
      minOccurs="0" maxOccurs="1"/>
    <xsd:element name="gendsn" type="xsd:string"
      minOccurs="0" maxOccurs="1"/>
    <xsd:element name="gennetservicename" type="xsd:string"
      minOccurs="0" maxOccurs="1"/>
    <xsd:element name="environment" type="xsd:string"
      minOccurs="0" maxOccurs="1"/>
    <xsd:element name="messagepath" type="xsd:string"
      minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

```

    <xsd:element name="connectorname" type="xsd:string"
      minOccurs="0" maxOccurs="1"/>
    <xsd:element name="jdbcdriver" type="xsd:string"
      minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>

```

```

</xsd:complexType>
<xsd:complexType name="mqstatusType">
  <xsd:sequence>
    <xsd:element name="rc" type="xsd:integer"/>
    <xsd:element name="status" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="mqdataType">
  <xsd:sequence>
    <xsd:element name="data" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>

```

MQSeries Objects Used

Object Name	Object Type	Description
BTRADE.DEAD.QUEUE	Local Queue	Local queue defined as the system dead letter queue. Message is placed on the queue when it is undeliverable.
BTRADE.SU35E16	Channel	Sender Channel. Used to send messages to SU35E16.
SU35E16.BTRADE	Channel	Receiver Channel. Used to receive messages from SU35E16.
SU35E16	Local Queue	Local queue defined as a transmission queue. Used when sending data from bTrade to SU35E16.
BTRADE.SU35E17	Channel	Sender Channel. Used to send messages to SU35E17.
SU35E17.BTRADE	Channel	Receiver Channel. Used to receive messages from SU35E17.
SU35E17	Local Queue	Local queue defined as a transmission queue. Used when sending data from bTrade to SU35E17.
BTRADE.FROM.EAI.GETMAIL	Local Queue	Local queue used to receive data as input for the bTrade Adapter application.
BTRADE.INIT	Local Queue	Local queue defined as an initiation queue.
SU35E5	Remote Queue	Remote queue used as a queue manager alias to be able to reroute the messages back to the SU35E5 server.

3.12.4 SAIG/bTrade Test Scenario Inputs

Each input file must be of a specific format in order for it to be recognized by the message flow and the MQSeries adapter. The input files for the SAIG/bTrade system are as follows:

GETALL XML request tr1.xml – This file is an XML document, which requests all messages from the IBMTST mailbox.

```

<mqrbtrade>
  <mqrequest>
    <request>1</request>
    <language>EN-US</language>
    <customer>NCS</customer>

```

```
<genschema>btradev</genschema>
<genusername>mailbox</genusername>
<genpassword>btradev</genpassword>
<gendsn>btradev</gendsn>
<gennetservicename>btradev</gennetservicename>
<environment>DEVELOPMENT</environment>
<messagepath>/eaadmin/data</messagepath>
<connectorname>IBMTST</connectorname>
<jdbcdriver>jdbc:oracle:oci8:@</jdbcdriver>
</mqrequest>
</mqrbtrade>
```

3.12.5 SAIG/bTrade Test Scenario Expected Results

A message is sent to the EAI Bus where the message is transformed and then sent on to the bTrade system. The bTrade MQ Adapter is called which executes the bTrade API to retrieve the mailbox data. If the mailbox does not contain any data corresponding to status code 9, the application returns a result of "Mailbox Empty". When data is in the mailbox with the correct status code, the EAI bTrade test scenario returns the file where the data is located.

A positive result of this test is the successful retrieval of a message.

3.12.5.1.1.1.1 Tr1debug.out - This file shows the full program trace output from a GetMail request.

```
-- listing properties --
user.language=en
java.home=c:\jdk1.1.8\bin\..
java.vendor.url.bug=
awt.toolkit=sun.awt.windows.WToolkit
file.encoding.pkg=sun.io
java.version=1.1.8
file.separator=\
line.separator=

Debug=a
user.region=US
file.encoding=Cp1252
java.compiler=ibmjitc
java.vendor=IBM Corporation
user.timezone=GMT
user.name=nich
os.arch=x86
java.fullversion=JDK 1.1.8 IBM build n118p-19991124 (J...
os.name=Windows NT
java.vendor.url=http://www.ibm.com/
user.dir=C:\doe\btrade\MQRbTrade
java.class.path=C:\PROGRA~1\MQSeries\java\lib;C:\PROG...
java.class.version=45.3
os.version=4.0
path.separator=;
user.home=C:\
MQRbTrade DEBUG: Main Started
MQRbTrade DEBUG: qmgr = queue = INBOUND.QUEUE
MQRbTrade DEBUG: inifn = MQbTrade.ini msgfn = MQbTrade.dat
MQRbTrade DEBUG: Input queue open gmo = 2
```

```
MQRbTrade DUMP: Input request received
MQRequest Dump:
  request = 1
  language = EN-US
  customer = NCS
  genschema = btradev
  genusername = mailbox
  genpassword = btradev
  gendsn = btradev
  environment = DEVELOPMENT
  messagepath = /eaadmin/data
  connectorname = IBMTST
  jdbcdriver = jdbc:oracle:oci8:@
MQStatus Dump:
  rc = 0
  status =
MQData Dump:
  charformat =
  dataformat =
  data =
MQRbTrade DEBUG: Reply queue open pmo = 66
MQRbTrade DEBUG: MQbTrade.ini written
MQRbTrade DEBUG: bTrade connection established.
MQRbTrade DEBUG: GETALL started
MQRbTrade DEBUG: 1 bTrade.com messages available
MQRbTrade DEBUG: Top of GET loop
MQRbTrade DEBUG: acceptMessage invoked
MQRbTrade DEBUG: MQ Message put
MQRbTrade DEBUG: GETALL finished
MQRbTrade DEBUG: End of requests
MQRbTrade DEBUG: All requests done
```

Tr1xml.out – This file contains the XML document created as a result of a GetMail request for the IBMTST mailbox.

```
<mqbtrade><mqrequest><request>1</request><language>EN-
US</language><customer>NCS</customer><genschema>btradev</genschema><gen
username>mailbox</genusername><genpassword>btradev</genpassword><gendsn
>btradev</gendsn><environment>DEVELOPMENT</environment><messagepath>/ea
admin/data</messagepath><connectorname>IBMTST</connectorname><jdbcdrive
r>jdbc:oracle:oci8:@</jdbcdriver></mqrequest><mqstatus><rc>0</rc><statu
s></status></mqstatus><mqdata><charformat> </charformat><dataformat>
</dataformat><data>example test data file
illustrating a piece of mail
that is supposed to be
in a btrade mailbox</data></mqdata></mqbtrade>
```

- This bTrade test scenario was executed by the Release 1 EAI Core team and the expected results were received and validated.

4 EAI COMPONENT MIGRATION

The migration of the Release 1 and 2 EAI Core components as designed and developed are dependent upon the following specific legacy system requirements, required licensing, and legacy system owner approval for migrating to each legacy system production environment. For each legacy system the system installation pre-requisites, the networking dependencies, and the required configuration are defined.

It is recommended that a meeting be conducted with each legacy system owner, Modernization Partner, and the SFA EAI Manager be conducted in advance of each system migration to review the requirements, steps and access required to migrate the Release 1 and 2 EAI Core architecture components into the production environments.

4.1 EAI Component Migration for Central Processing System (CPS) and National Student Loan Data System (NSLDS)

4.1.1 System Installation

Virtual Data Center (VDC) personnel performed all OS/390 CPS Installations and Configurations. The installation of the MQSeries Messaging software and required adapters, MQSeries CICS DPL Bridge, MQSeries Batch Adapter, Trigger Monitor, will be performed by CSC. Specifically, the installation will require MQSeries administrators for the MQSeries product and the CICS systems support for the CICS DPL Bridge. These two components are provided with the MQSeries software product delivered to SFA. The instructions to install these components are provided in the MQSeries manuals, which have been provided in hard copy to CSC.

Upon completion of the MQSeries adapters the adapters will need to be configured for each application to support the OS/390 definitions required to execute the adapters. The definitions of the specific adapter configurations are dependent upon the application requirements for enabling the application on the OS/390 platform per the application EAI enablement requirements.

4.1.2 Networking

In order for MQSeries to communicate and exchange messages within the defined infrastructure network connectivity tasks must be completed. First, determine what port the system will use for the listener process. The default port is 1414. A determination must be made during installation to verify that this port is available and not previously defined. Once the port has been decided upon, contact the necessary networking personnel to request all firewalls, gateways and routers between the VDC and the CPS system be configured to include the IP protocol with telnet, ping, and ftp capability.

4.1.3 Configuration on the CPS System

The following tasks should be performed when performing the EAI configuration for CPS,

- 1) Install MQSeries Messaging V5.2 and the out of the box MQSeries CICS DPL bridge on the CPS system.

- 2) Once the MQSeries installation is complete, the object definitions for CPS in the scripts provided in Deliverable 54.1.3 should be used to configure the MQSeries environment. These object definitions in the text script will need to be given to the MQSeries Administrators to define.
- 3) Modify the script files on the Sun Solaris servers with new channel and queue definitions for the CPS production system.
- 4) Setting up the CICS DPL Bridge to start automatically can be done out of the CICS sequential terminal processing, the PLT startup or set on the queue object for the bridge to be TRIGGER, TRIGTYPE(FIRST), APPLICID(CKBR) and specifying the AUTH and WAIT parameters in the USERDATA field in the related process definition.

4.1.4 Configuration on the NSLDS System

The following tasks should be performed when performing the EAI configuration for NSLDS,

- 1) Install MQSeries Messaging V5.2 and the out of the box MQSeries Batch Adapter on the NSLDS system.
- 2) Once the MQSeries installation is complete, the object definitions for NSLDS provided in Deliverable 54.1.3 can be used to configure the MQSeries environment. These object definitions in the text script will need to be given to the MQSeries Administrators to define.
- 3) Modify the script files on the Sun Solaris servers with new channel and queue definitions for the NSLDS production system.
- 4) To set up the Batch Adapter and the Trigger monitor refer to Deliverable 54.1.3 – EAI Technical Specifications Document Release 1.

4.1.5 MQ Object Definitions

The MQ Objects for the CPS and NSLDS systems are defined in Deliverable 54.1.3 – EAI Technical Specifications Document Release 1. These object definitions used for the development and test phase will need to be reviewed and may require modification based on the production system configurations.

4.1.6 Other CPS and NSLDS Migration Considerations

The current installation of the MQSeries Messaging software for the CPS and NSLDS systems was based on a temporary license. The migration to production for each of these systems will require production MQSeries licenses that are based on usage and capacity. In addition, DASD must be allocated for MQSeries operation. The applications requiring MQSeries on the CPS and NSLDS systems must define their application security requirements prior to migrating into the production environment so access definitions can be updated in the RACF database.

4.2 EAI Component Migration for Direct Loan Servicing System (DLSS)

4.2.1 System Installation

The EAI configuration for the DLSS system requires the installation and configuration of MQSeries for Compaq OpenVMS v2.2.1.1 (Note: Version 5.1 of MQSeries for OpenVMS was released at the end of May, 2001).

The EAI Core Adapters developed for Release 1 were developed in 'C' and therefore the Compaq OpenVMS C Compiler for Alpha Systems is necessary for the programs to be recompiled and linked.

4.2.2 Networking

In order for MQSeries to communicate and exchange messages within the defined infrastructure network connectivity tasks must be completed. First, determine what port the system will use for the listener process. The default port is 1414. A determination must be made during installation to verify that this port is available and not previously defined. Once the port has been decided upon, contact the necessary networking personnel to request all firewalls, gateways and routers between Rockville, MD and the DLSS system be configured to include the IP protocol with telnet, ping, and ftp capability.

4.2.3 Configuration

The following tasks should be performed when configuring the DLSS production system EAI components,

- 1) Configure port 1414 (or any non-used port) for the listener process by creating the MQSeries service with the UCX utility.
- 2) Modify the system startup and shutdown procedures to mimic what was done for MQSeries on the OpenVMS test system. Specifically, start MQSeries and execute the command procedure [mqm]start_mq_batch_jobs.com
- 3) Modify the script file EAI.TST with the necessary object names. Specifically, the channel names and queue names for the production system.
- 4) Modify the script files (EAI.TST) on the Sun Solaris servers with new channel and queue definitions for the production system.
- 5) Compile and link the adapter programs.

```
$CC / INCLUDE_DIRECTORY=MQS_INCLUDE MQPUT
$CC / INCLUDE_DIRECTORY=MQS_INCLUDE MQGET
$LINK MQPUT.OBJ, SYS$INPUT /OPTIONS <enter>
SYS$SHARE:MQM/SHARE
$LINK MQGET.OBJ, SYS$INPUT /OPTIONS <enter>
SYS$SHARE:MQM/SHARE
```

4.3 EAI Component Migration for Electronic Campus Based System (eCBS)

4.3.1 System Installation

The EAI Core components for the eCBS system require the following pre-requisites be installed in advance of migrating the Release 2 EAI components onto the eCBS production system:

- Sun Solaris Version 2.6, WebSphere Application Server and Oracle RDBMS as required by the eCBS system
- eCBS developed stored procedures to interface with the custom adapter for real-time requests
- eCBS developed shell script to interface with the custom adapter for batch requests
- MQSeries Messaging Version 5.2 with MQ base Java support
- ITA Reusable Service - Persistence Framework used to connect to the eCBS database
- Reusable EAI Function – Common Logging Component used to log MQSeries errors.
- MQECBS – EAI Core Architecture eCBS MQ Adapter
- HP hardware capacity, i.e. CPU processor, RAM, etc, as required by the eCBS production system and DASD requirements.

4.3.2 Networking

In order for MQSeries to communicate and exchange messages within the defined infrastructure network connectivity must be completed. First, determine what port the system will use for the listener process. The default port is 1414. A determination must be made during installation to verify that this port is available and not previously defined. Once the port has been decided upon, contact the necessary networking personnel to request all firewalls, gateways and routers between the VDC and the eCBS system be configured to include the IP protocol with telnet, ping, and ftp capability.

4.3.3 Configuration

The following tasks should be performed when configuring the eCBS Production system EAI components:

- 1) Configure port 1414 for the listener process by creating the MQSeries service through the UNIX inetd daemon.
- 2) Create and install MQSeries start and stop scripts to provide for an orderly shutdown of MQSeries by the system administrator, as well as automatic startup of the MQSeries messaging component upon system startup.
- 3) Create a MQSeries script file, for the eCBS system, that contain the necessary MQSeries object definitions needed to connect the eCBS system to the EAI Bus. This includes channel definitions and queue definitions.
- 4) Modify the MQSeries script file on the EAI Sun Solaris servers to contain the necessary MQSeries object definitions needed to connect the eCBS system to the EAI Bus. This includes channel definitions and queue definitions.
- 5) Compile the adapter programs using:

```
javac MQECBS.java
```

There will be three queues associated with the eCBS MQSeries Adapter. There are two inbound queues, one for processing batch requests and the other for processing real-time requests. When messages land on either queue, the eCBS MQSeries Adapter will be triggered to perform the specific request. The third queue is a transmission queue, which the eCBS MQSeries Adapter uses to send results obtained from both batch and real-time processing back to the source application.

MQSeries objects used on the development and test systems are defined in Deliverable 54.1.6 – EAI Release 2.0 Technical Specification. These objects will have to be reviewed and may require modification based on the application requirements for enabling an application onto the EAI Bus using the eCBS system.

4.4 EAI Component Migration for Financial Management System (FMS)

4.4.1 System Installation

The EAI Core components for the FMS system require the following pre-requisites be installed in advance of migrating the Release 2 EAI components onto the FMS production system:

- HP-UX Version 11.0, and Oracle RDBMS as required by the FMS system
- FMS developed database staging tables to interface with the custom adapter for real-time requests
- FMS developed shell script to interface with the custom adapter for batch requests
- MQSeries Messaging Version 5.2 and Application Messaging Interface Version 1.2
- Reusable EAI Function – Common Logging Component used to log MQSeries errors.
- MQFMS – EAI Core Architecture FMS MQ Adapter
- HP hardware capacity, i.e. CPU processor, RAM, etc, as required by the FMS production system and DASD requirements.

4.4.2 Networking

In order for MQSeries to communicate and exchange messages within the defined infrastructure network connectivity must be completed. First, determine what port the system will use for the listener process. The default port is 1414. A determination must be made during installation to verify that this port is available and not previously defined. Once the port has been decided upon, contact the necessary networking personnel to request all firewalls, gateways and routers between the VDC and the FMS system be configured to include the IP protocol with telnet, ping, and ftp capability.

4.4.3 Configuration

The following tasks should be performed when configuring the FMS Production system EAI components:

- 1) Configure port 1414 for the listener process by creating the MQSeries service through the UNIX inetd daemon.
- 2) Create and install MQSeries start and stop scripts to provide for an orderly shutdown of MQSeries by the system administrator, as well as automatic startup of the MQSeries messaging component upon system startup.
- 3) Create a MQSeries script file, for the eCBS system, that contain the necessary MQSeries object definitions needed to connect the eCBS system to the EAI Bus. This includes channel definitions and queue definitions.
- 4) Modify the MQSeries script file on the EAI Sun Solaris servers to contain the necessary MQSeries object definitions needed to connect the eCBS system to the EAI Bus. This includes channel definitions and queue definitions.
- 5) Compile the adapter programs using:

```
javac MQFMS.java
```

There will be three queues associated with the FMS MQSeries Adapter. There are two inbound queues, one for processing batch requests and the other for processing real-time requests. When messages land on either queue, the FMS MQSeries Adapter will be triggered to perform the specific request. The third queue is a transmission queue, which the FMS MQSeries Adapter uses to send results obtained from both batch and real-time processing back to the source application.

MQSeries objects used on the development and test systems are defined in Deliverable 54.1.6 – EAI Release 2.0 Technical Specification. These objects will have to be reviewed and may require modification based on the application requirements for enabling an application onto the EAI Bus using the FMS system.

4.5 EAI Component Migration for LO System – Electronic Master Promissory Note (eMPN)

4.5.1 System Installation

The EAI Core components for LO System-eMPN require the following pre-requisites be installed in advance of migrating the Release 2 EAI components onto the LO System production system:

- HP-UX Version 11.x
- EDS developed java classes to retrieve eMPN reference information from a database.
- MQSeries Messaging Version 5.2 and Application Messaging Interface v1.2
- EDS developed API socket program must be running to retrieve eMPN reference information from a database.
- HP hardware capacity, i.e. CPU processor, RAM, etc, as required by the LO System production system and DASD requirements.

4.5.2 Networking

In order for MQSeries to communicate and exchange messages within the defined infrastructure network connectivity tasks must be completed. First, determine what port the system will use for the listener process. The default port is 1414. A determination must be made during installation to verify that this port is available and not previously defined. Once the port has been decided upon, contact the necessary networking personnel to request all firewalls, gateways and routers between the VDC and the LO system be configured to include the IP protocol with telnet, ping, and ftp capability.

4.5.3 Configuration

The following tasks should be performed when configuring the LO System-eMPN Production EAI components:

- 1) Install the eMPN adapter, create an MQSeries process that points to the adapter, and set up the queue object to be TRIGGER, TRIGTYPE(EVERY).
- 2) Configure port 1414 (or any non-used port) for the listener process by updating the /etc/services and /etc/inetd.conf files.
- 3) Modify the system startup and shutdown procedures to mimic what was done for MQSeries on the LO Web-T Server.
- 4) Modify the script file LOWEB.TST with the necessary object names. Specifically, the channel names and queue names for the production system.
- 5) Modify the script files (EAI.TST) on the Sun Solaris servers with new channel and queue definitions for the production system.
- 6) Compile the adapter programs:

```
export CLASSPATH=empn1.jar:common.jar  
javac EMPN.java
```

There will be two queues associated with the eMPN MQSeries Adapter, one inbound and the other outbound. When messages land on the inbound queue, the eMPN MQSeries Adapter will be triggered to perform the specific request. The outbound queue is a transmission queue, which the eMPN MQSeries Adapter uses to send results back to the source application.

Develop and install MQSeries start and stop scripts to provide for an orderly shutdown of the system by the system administrator, and to provide the automatic startup of the MQSeries messaging component upon system startup.

MQSeries objects used on the development and test systems are defined in Deliverable 54.1.6 – EAI Release 2.0 Technical Specification. These objects will have to be reviewed and may require modification based on the application requirements for enabling an application onto the EAI Bus using the eMPN system.

4.6 EAI Component Migration for LO System – Promissory Note Imaging (P-Note Imaging)

4.6.1 System Installation

The EAI Core components for LO System-P-Note Imaging require the following pre-requisites be installed in advance of migrating the Release 2 EAI components onto the LO System production system:

- HP-UX Version 11.x
- MQSeries Messaging Version 5.2 and Application Messaging Interface v1.2
- HP hardware capacity, i.e. CPU processor, RAM, etc, as required by the LO System production system and DASD requirements.

4.6.2 Networking

In order for MQSeries to communicate and exchange messages within the defined infrastructure network connectivity tasks must be completed. First, determine what port the system will use for the listener process. The default port is 1414. A determination must be made during installation to verify that this port is available and not previously defined. Once the port has been decided upon, contact the necessary networking personnel to request all firewalls, gateways and routers between the VDC and the LO system be configured to include the IP protocol with telnet, ping, and ftp capability.

4.6.3 Configuration

There will be two queues associated with the P-Note Imaging MQSeries Adapter and test program, one inbound and the other outbound. When messages land on the inbound queue, the P-Note Imaging test program will be triggered to perform the specific request. The outbound queue is a transmission queue, which the P-Note Imaging MQSeries Adapter uses to send results back to the source application.

Develop and install MQSeries start and stop scripts to provide for an orderly shutdown of the system by the system administrator, and to provide the automatic startup of the MQSeries messaging component upon system startup.

MQSeries objects used on the development and test systems are defined in Deliverable 54.1.6 – EAI Release 2.0 Technical Specification. These objects will have to be reviewed and may require modification based on the application requirements for enabling an application onto the EAI Bus using the P-Note Imaging system.

The following tasks should be performed when configuring the LO System-eMPN Production EAI components:

- 1) Install the eMPN adapter, create an MQSeries process that points to the adapter, and set up the queue object to be TRIGGER, TRIGTYPE(EVERY).
- 2) Configure port 1414 (or any non-used port) for the listener process by updating the /etc/services and /etc/inetd.conf files.
- 3) Modify the system startup and shutdown procedures to mimic what was done for MQSeries on the LO Web-T Server.

- 4) Modify the script file LOWEB.TST with the necessary object names. Specifically, the channel names and queue names for the production system.
- 5) Modify the script files (EAI.TST) on the Sun Solaris servers with new channel and queue definitions for the production system.
- 6) Compile the adapter programs:
javac EAILog.java
javac IEAI.java
javac MSG.java
javac EAIException.java
javac IeMPN.java
javac EAI.java
javac PNOTE.java

4.7 EAI Component Migration for National Student Loan Data System (NSLDS)-Cool:Gen

4.7.1 System Installation

The current NSLDS development and run-time environments utilize the Cool:Gen product from Computer associates. Cool:Gen is an Integrated Development Environment (IDE) tool for developing software applications. Integration of MQSeries and Cool:Gen on the NSLDS system requires the following products,

- MQSeries V5.2 for OS/390 – NSLDS mainframe
- MQSeries V5.1 for NT Server plus CSD 5 or higher – NSLDS Web Server
- Cool:Gen Version V5.1 Plus PTF's applied – NSLDS mainframe
- CICS V4.1 – NSLDS mainframe
- OS/390 V2R8 or V2R10 – NSLDS mainframe
- Microsoft Windows NT Web Server – NSLDS Web Server
- Cool:Gen MQSeries Transaction Dispatcher for CICS (TDC) - NSLDS mainframe

The base components of TDC should be installed as part of the standard MVS install. The base components must be defined to CICS in order for the components to be accessible. Additionally, Virtual Storage Access Mechanism (VSAM) definitions will be necessary if the application intends to use this option for temporary storage.

The MQSeries CICS Trigger Monitor (CKTI) must be installed and enabled.

CICS installation

The modules TIRMQTDC and, if used, TIRMQTDX must be in the DFHRPL concatenation. If the COOL:Gen LOADLIB is not allocated to DFHRPL, copy TIRMQTDC from the Cool:Gen LOADLIB to the DFHRPL library of your choice.

Run DFHCSDUP using the following deck (the language for TIRMQTDX may be changed, if necessary),

```
DEFINE TRANSACTION(TITD)
DESCRIPTION(COOL:Gen Transaction dispatcher)
PROGRAM(TIRMQTDC)
TASKDATALOC(ANY)
GROUP(TDCGROUP)
DEFINE PROGRAM(TIRMQTDC)
DESCRIPTION(COOL:Gen Transaction dispatcher)
LANGUAGE(ASSEMBLER)
DATALOCATION(ANY)
GROUP(TDCGROUP)
DEFINE PROGRAM(TIRMQTDX)
DESCRIPTION(COOL:Gen Transaction dispatcher control exit)
LANGUAGE(LE370)
DATALOCATION(ANY)
GROUP(TDCGROUP)
```

4.7.2 Networking

In order for MQSeries to communicate and exchange messages within the defined infrastructure network connectivity tasks must be completed. First, determine what port the system will use for the listener process. The default port is 1414. A determination must be made during installation to verify that this port is available and not previously defined. Once the port has been decided upon, contact the necessary networking personnel to request all firewalls, gateways and routers between the VDC and NSLDS be configured to include the IP protocol with telnet, ping, and ftp capability.

4.7.3 Configuration

The following are configuration steps required to migrate the NSLDS Cool:Gen system to the production environment,

- 1) Modify the system startup and shutdown procedures to replicate what was done for MQSeries on the test system.
- 2) Move the script file T048DEV1.TST to the production system. Modify it with the necessary object names for the production system.
- 3) Modify the script file T048DEV1.TST with the MQSeries definitions for the production system.

4.8 EAI Component Migration for Post-Secondary Education Participants System (PEPS)

4.8.1 System Installation

The EAI Core components for the PEPS system require the following pre-requisites be installed in advance of migrating the Release 1 EAI components onto the PEPS production system:

- HP/UX Version HP-UX V10.x and Oracle RDBMS as required by the PEPS system
- PEPS developed stored procedures to interface with the MQ Adapter
- MQSeries Messaging Version 5.2 with MQ base Java support and the Product Extension MA88
- IBM XML4J XML Parser 3.1.1
- MQPEPS 1.1.0.8 – EAI Core Architecture PEPS MQ Adapter
- HP hardware capacity, i.e. CPU processor, RAM, etc, as required by the PEPS production system and DASD requirements.

4.8.2 Networking

In order for MQSeries to communicate and exchange messages within the defined infrastructure network connectivity must be completed. First, determine what port the system will use for the listener process. The default port is 1414. A determination must be made during installation to verify that this port is available and not previously defined. Once the port has been decided upon, contact the necessary networking personnel to request all firewalls, gateways and routers between the VDC and the PEPS system be configured to include the IP protocol with telnet, ping, and ftp capability.

4.8.3 Configuration

There will be two queues associated with the PEPS MQSeries Adapter. One, an inbound queue which, clients can put requests to the PEPS for processing and from which the PEPS MQSeries Adapter will be triggered to perform the requests. The other, an outbound queue, which the PEPS MQSeries Adapter will put results obtained from processing of a PEPS stored procedure to be returned to the calling application.

Develop and install MQSeries start and stop scripts to provide for an orderly shutdown of the system by the system administrator, and to provide the automatic startup of the MQSeries messaging component upon system startup.

MQSeries Objects used on the development and test systems are defined in Deliverable 54.1.3 – EAI Technical Specifications Document Release 1. These objects will have to be reviewed and may require modification based on the application requirements for enabling an application onto the EAI Bus using the PEPS system.

4.9 EAI Component Migration for Student Aid Internet Gateway (SAIG/bTrade)

4.9.1 System Installation

The EAI Core components for the bTrade system require the following pre-requisites be installed in advance of migrating the Release 1 EAI components onto the bTrade production system:

- HP/UX Version HP-UX hpdev1 B.11.00 U 9000/800 (tb)
- Java JDK HP-UX Java C.01.18.01 12/10/1999 12:48:46 dyo640
- The installation of the bTrade connectorAPI, provided by bTrade/NCS, as per the bTrade connectorAPI program design specification document.
- The installation and configuration of Oracle 8.1 with the required JDBC support – this is a bTrade application requirement.
- Validation of the bTrade connectorAPI operation and configuration for all required API parameters and object states
- MQSeries Messaging Version 5.2 with MQ base Java support and the Product Extension MA88
- IBM XML4J XML Parser 3.1.1
- MQRbTrade 1.1.0.5 – EAI Core Architecture bTrade MQ Adapter
- HP hardware capacity, i.e. CPU processor, RAM, etc, as required by the bTrade production system and DASD requirements.

4.9.2 Networking

In order for MQSeries to communicate and exchange messages within the SFA EAI infrastructure network connectivity tasks must be completed. First, determine what port the system will use for the listener process. The default port is 1414. A determination must be made during installation to verify that this port is available and not previously defined. Once the port has been decided upon, contact the necessary networking personnel to request all firewalls, gateways and routers between the VDC and the bTrade system be configured to include the IP protocol with telnet, ping, and ftp capability.

4.9.3 Configuration

There are two queues associated with the bTrade MQSeries Adapter. One, an inbound queue, which clients can put requests for bTrade system processing and from which the bTrade MQSeries Adapter will be triggered to perform the requests. The other, an outbound queue, which the bTrade MQSeries Adapter will put data and status obtained from bTrade processing and from which clients can be triggered on and or pull from.

Develop and install MQSeries start and stop scripts to provide for an orderly shutdown of the system by the system administrator, and to provide the automatic startup of the MQSeries messaging component upon system startup.

MQSeries Objects used on the development and test system are defined in Deliverable 54.1.3 – EAI Technical Specifications Document Release 1. These objects will have to be reviewed and may require modification based on the application requirements for enabling an application onto the EAI Bus using the bTrade system.

5 SIEBEL ADAPTER ANALYSIS

5.1 MQSeries/Siebel Integration Options

At this point in their product lifecycles both IBM MQSeries and Siebel 2000 are mature products with substantial install bases. Accordingly the question of how to integrate the two has been asked many times. Solutions range from COTS adapters to custom code and combinations thereof.

5.1.1 Siebel EAI

With the release of Siebel 2000 a service, Siebel EAI, is provided which makes integration with MQSeries simple. Siebel EAI is a framework that allows Siebel application developers to communicate using a simplified API, reducing the coding effort that would otherwise be involved with inserting the MQSeries API code into a large number of interface points.

The Siebel EAI framework using the EAI MQSeries Transport Connector accepts MQSeries messages containing XML documents implementing certain schema and responds with other XML documents. Applications that wish to interact with Siebel 2000 using Siebel EAI must do so in one of two ways. Messages may either be sent directly to Siebel in the format required or they can be transformed using an integration broker such as IBM MQSeries Integrator and then forwarded.

Within Siebel itself it is necessary to write Siebel Script or VB script programs using the Siebel EAI API to get and put messages to queues. This API is highly abstract and requires almost no knowledge of MQSeries to use effectively. The Siebel EAI MQSeries Transport Connector is written to use the MQSeries Application Messaging Interface(AMI), which has been adopted as a standard by SFA.

The Siebel EAI framework and its MQSeries Transport Connector are included with the base Siebel 2000 offering and therefore are available at no additional cost.

5.1.2 Neon (Sybase) Adapter for Siebel e-Business Applications

Neon (New Era of Networks) offers the Neon Adapter for Siebel e-Business Applications. This preconfigured plug-in works with Neon Integration Servers to connect existing systems with Siebel. The adapter allows the Integration Server to be quickly installed by loading the interface definitions as they are defined for a customized application into the Integration Servers' metadata database.

The adapter itself functions by using the Microsoft COM interface provided by the Siebel application. This interface allows real-time update of Siebel data from external sources.

IBM MQSeries integrator is capable of using the Neon adapter for Siebel. In this way, messages sent via MQSeries to MQSeries Integrator can be then used as input to the adapter. The adapter would then make synchronous calls to Siebel via the COM interface.

This solution is only partially compliant with the architecture designed for SFA since it would use the Microsoft COM technology to cross platforms and access Siebel.

The Neon adapter for Siebel would cost approximately \$65k per installed server.

5.1.3 Propelis EAI (Enterprise Access Integrator)

Propelis offers an EAI suite designed to integrate disparate applications. It works much like many other integration hubs, by being aware of many protocols and interfaces and simply bridging one to another.

Propelis offers a MQSeries gateway to their EAI hub which accepts MQSeries messages as input and then reformats them and executes a request using whatever native interface is appropriate. In the case of Siebel it uses the same COM interface which the Neon adapter leverages.

This offering is not compliant with our stated SFA architecture and should not be considered.

5.1.4 SFA Custom Solution

Since both MQSeries and Siebel 2000 have API interfaces available it is possible to develop custom components that would bridge the two. A MQSeries message could be sent to the Siebel server where a custom program would retrieve it and then use the Siebel COM interface to execute updates or queries.

This solution would require a developer to have knowledge of both MQSeries and Siebel APIs. Additionally it would require maintenance to be performed in case of business requirement changes.

The SFA Custom Solution may be also be implemented using the MQSeries Adapter Offering toolkit described below.

5.1.5 MQSeries Adapter Offering (MQAO)

The MQSeries Adapter Offering provides a tool to make building on- and off- ramp adapters more productive. It can be used as a general purpose tool for building adapters and adapter templates to any application. Building a library of specific Adapter Templates for an application (rather like a subroutine library) makes the tool more productive than always building from scratch. It can also be used in conjunction with connectors to commonly used package applications that again makes the tool more productive.

The Adapter Builder is a tool that allows a developer to import an application's interface into a repository either by processing 'C' header files containing function prototypes and structure definitions or by importing Java classes (preferably JavaBeans) from .java source files. It also allows a message format definition to be imported into the repository by processing XML Data Type Documents (DTDs). The OAG provide DTDs for all of their BODs on their Web site¹. The repository used by the tool is based on the same technology used by MQSeries Integrator V2, and therefore the message format definitions can be shared between the adapters and the message broker.

The MQSeries Adapter Kernel provides standard runtime functionality that's required by any MQSeries adapter created by the Adapter Builder, including interaction with MQSeries queuing, configuration, and handling log and debug information. The Adapter Kernel is the same for all adapters and the same code is therefore built for each of the supported platforms. Adapters can be run using base MQSeries messaging, or JMS (Java Message Service). JMS is the transport of choice where adherence to open standards is paramount, and when building solutions for WebSphere.

5.2 Conclusion

The Siebel 2000 standard EAI framework provides more than sufficient functionality for the SFA EAI environment. It is tightly integrated with the Siebel product itself and complies fully with the SFA EAI architecture. Furthermore, since it is a part of the base product no additional products must be licensed and no costs incurred. Its API is simple and easily understood and developers will be quickly able to become proficient in its use.

While other offerings are all capable, they are either intended for use as the hub in an overall integration strategy, or would require considerable development in order to implement.

The use of the Siebel 2000 EAI services is recommended for integration with the SFA EAI Bus.