

SFA Modernization Partner

United States Department of Education

Student Financial Assistance



EAI Core Architecture
EAI Technical Specifications
(Release 1)

Task Order #54

Deliverable # 54.1.3

July 13, 2001

Table of Contents

1	INTRODUCTION.....	5
1.1	OBJECTIVES.....	5
1.2	SCOPE.....	5
1.3	BUSINESS DRIVERS	6
1.4	ASSUMPTIONS	6
1.5	APPROACH.....	6
1.6	ORGANIZATION OF THIS DOCUMENT	6
2	EAI CORE ARCHITECTURE SYSTEMS AND COMPONENTS OVERVIEW	8
3	INTEGRATION ARCHITECTURE	10
3.1	INTERFACE DEFINITIONS.....	11
3.2	MAINFRAME LEGACY SYSTEMS.....	11
3.2.1	CPS EAI System Overview	12
3.2.1.1	CPS Messaging Components	12
3.2.1.2	MQSeries Provided Adapters.....	12
3.2.1.3	CPS Custom Built Adapters	13
3.2.1.4	CPS Data Flow and Message Flow Diagrams	13
3.2.1.5	CPS MQSeries Programming Sample Source Code	15
3.2.2	NSLDS EAI System Overview.....	15
3.2.2.1	NSLDS Messaging Components.....	15
3.2.2.2	MQSeries Provided Adapters.....	16
3.2.2.3	NSLDS Custom Built Adapters.....	16
3.2.2.4	NSLDS Data Flow and Message Flow Diagrams.....	17
3.2.2.5	NSLDS MQSeries Programming Sample Source code	19
3.2.2.6	NSLDS MQSeries OS/390 Trigger Monitor and Custom Cool:Gen Adapters	19
3.3	MID-TIER LEGACY SYSTEMS	21
3.3.1	DLSS EAI System Overview	21
3.3.1.1	DLSS Messaging Components	21
3.3.1.2	MQSeries Provided Adapters.....	21
3.3.1.3	DLSS Custom Built Adapters	21
3.3.1.4	DLSS Data Flow and Message Flow Diagrams	22
3.3.1.5	DLSS MQSeries Programming Sample Source Code	25
3.3.2	PEPS EAI System Overview	25
3.3.2.1	PEPS Messaging Components	25
3.3.2.2	MQSeries Provided Adapters.....	25
3.3.2.3	PEPS Custom Built Adapters	26
3.3.2.4	PEPS Data Flow and Message Flow Diagrams	26
3.3.2.5	PEPS MQSeries Programming Sample Source code.....	28
3.3.3	bTrade EAI System Overview	28
3.3.3.1	bTrade Messaging Components.....	28
3.3.3.2	MQSeries Provided Adapters.....	28
3.3.3.3	bTrade Custom Built Adapters.....	28
3.3.3.4	BTrade Data Flow and Message Flow Diagrams	29
3.3.3.5	bTrade MQSeries Programming Sample Source code	30
3.4	WEBSHERE APPLICATION SERVER	31
3.4.1	EAI WAS Request / Reply Data Flow.....	31
3.4.2	MQ-WebSphere Adapter	32

3.5	EAI CORE ARCHITECTURE RELEASE 1 SOFTWARE PRODUCTS	32
4	EXECUTION ARCHITECTURE COMPONENTS	34
4.1	MQSERIES MESSAGING CAPABILITIES	34
4.1.1	Application Programs and Messaging	34
4.1.2	Queue Managers	35
4.1.3	Connecting an Application to a Queue Manager	35
4.1.4	Opening a Queue	35
4.1.5	Putting and Getting Messages	35
4.1.6	Transactional Integrity	35
4.1.7	Security	36
4.1.8	Triggering	36
4.2	MQSERIES INTEGRATOR CAPABILITIES	36
4.3	EXPORTING MESSAGE FLOWS BETWEEN DEVELOPMENT WORKSTATIONS AND MQSI BUILD-TIME SERVER	38
4.3.1	Exporting Message Flows	38
4.3.2	Importing Message Flows	38
4.4	DEPLOYING MQSI CONFIGURATION DATA FROM THE BUILD-TIME SERVER TO THE RUN-TIME SERVER	39
4.4.1	Three types of deployment	39
4.4.1.1	<i>Complete deployment</i>	39
4.4.1.2	<i>Delta deployment</i>	39
4.4.1.3	<i>Forced deployment</i>	40
4.4.2	Stages of Deployment	40
4.4.2.1	<i>Stage One of Deployment</i>	40
4.4.2.2	<i>Stage Two of Deployment</i>	40
4.4.3	Deploying and Checking Data In and Out	40
4.4.4	Verifying Successful Deployment	41
4.4.5	SFA EAI Release 1 Core Specific Deployment Details	41
4.4.5.1	<i>Deployment Cookbook</i>	41
4.5	SFA EAI ARCHITECTURE CUSTOMIZATION	42
4.5.1	MQSeries Implementation and Configurations for the Non-Legacy Integration Release 1	42
4.5.1.1	<i>MQSeries for WebSphere on Solaris</i>	43
4.5.1.2	<i>EAI Bus Client / Server Configuration and Design</i>	43
4.5.2	MQSeries Implementation and Configurations for the Legacy Integration Release 1	46
4.5.2.1	<i>MQSeries for CPS and NSLDS on OS/390</i>	46
4.5.2.2	<i>MQSeries for DLSS on Open VMS</i>	49
4.5.2.3	<i>MQSeries for PEPS on HP-UX</i>	49
4.5.2.4	<i>MQSeries for bTrade on HP-UX</i>	50
4.6	ERROR HANDLING	50
4.7	SCALABILITY	51
4.8	REDUNDANCY	52
4.9	LOAD BALANCING	52
5	DEVELOPMENT ARCHITECTURE	53
5.1	OVERVIEW	53
5.2	DESCRIPTION	53
5.3	OPERATING SYSTEMS	54
5.4	DEVELOPMENT PROCESS	54
5.5	MQSI DEVELOPMENT ENVIRONMENT	54
5.6	MQSI CONFIGURATION CONSIDERATIONS	55

5.7 DEVELOPMENT TOOLS 56
5.8 CONFIGURATION MANAGEMENT 56
APPENDIX A: MQSERIES SCRIPTS AND PROGRAMS 57
APPENDIX B: REFERENCE MATERIAL 61

1 INTRODUCTION

This document provides a comprehensive overview of the Student Financial Assistance (SFA) Technical Specifications required for Release 1 of the Enterprise Application Integration (EAI) Core Architecture. Release 1 provides legacy system connectivity deemed most critical for the initial implementation of the EAI Core infrastructure and a technical architecture that meets current and future implementation requirements for using the SFA EAI Integration architecture.

1.1 OBJECTIVES

The document is intended to provide the SFA specific EAI architectural detail information. The EAI Technical Specification Release 1 deliverable defines the architecture design and services provided by the EAI Core Architecture to support the Modernization effort of SFA's Information Technology (IT) Enterprise for enabling legacy systems to utilize the EAI Bus. The objective of this document is to provide the information necessary to understand the components comprising the EAI Bus, and the architectural design for enabling the Release 1 legacy systems to utilize the services provided by the enterprise EAI architecture at SFA.

1.2 SCOPE

The EAI Core Architecture effort consists of the installation, design, build, test, and implementation of a MQSeries integration architecture for the Department of Education, Office of Student Financial Assistance (SFA) Release 1 legacy systems. The Release 1 core MQSeries integration architecture consists of the following EAI software components:

- MQSeries Messaging
- MQSeries Integrator
- MQ Adapters

Collectively, the implementation of the EAI components and architecture is referred to as the EAI Bus. The EAI components provide the core architecture to enable SFA applications to utilize a common, reusable infrastructure for connecting disparate, heterogeneous systems. This document defines the technical specifications that describe the design and implementation details. The EAI core architecture provides the EAI components and services to connect the SFA Internet Domain to legacy systems through an EAI Bus. For Release 1 the EAI core architecture provides the infrastructure to connect to the following SFA legacy systems:

- BTrade (TIVWAN)
- CPS
- DLSS
- NSLDS
- PEPS

Connections to these Release 1 legacy systems is provided by MQSeries Messaging as the transport layer, and MQSeries Integrator for the transformation and routing of message data between systems. On each legacy system MQ Adapters have been installed and configured to validate the core messaging infrastructure for connecting to the EAI Bus and sample message flows have been designed and implemented for each Release 1 legacy system to validate message transformation of data as required by each system. In addition, MQ Adapters have been designed and implemented to validate sample

functionality representative of each Release 1 legacy system to verify the functionality of each system to interface with the MQSeries messaging components to retrieve and put messages from/onto the EAI Bus.

1.3 BUSINESS DRIVERS

The EAI Core Architecture for Release 1 does not include any application related functionality or business logic, other than that provided to validate the design, installation and configuration of the MQ Adapters for each Release 1 legacy system. Upon completion of the EAI Core Architecture for the Release 1 legacy systems application developers will be able to design, develop and deploy applications using the services provided by the EAI Core architecture.

1.4 ASSUMPTIONS

The EAI Core Technical Specifications deliverable is based on the following assumptions.

- No application requirements will be included within this documentation.
- The EAI Technical Specifications document will be for SFA specific customized EAI Core Architecture components.
- There are no specifications to security defined within this documentation, as these are application specific.
- There are no detailed specifications to performance tuning, as these are application specific based on message volume and transaction requirements.

1.5 APPROACH

The following approach was used to develop the EAI Technical Specifications deliverable:

- Review each Release 1 legacy system from a service and interface perspective to validate the design of the required EAI components to connect each system to the EAI Bus.
- Review the EAI functional services required for each Release 1 Legacy System
- Design the EAI Core architecture based on the development of test scenarios that will validate the MQSeries messaging and transformation architecture to connect each of the Release 1 Legacy Systems to the EAI Bus.

1.6 ORGANIZATION OF THIS DOCUMENT

This document is divided into the following sections:

- Section 1 – Introduction
- Section 2 – EAI Core Architecture Systems and Component Overview
This section identifies the systems that are a part of the EAI Core Architecture for Release 1 and identifies the components used.
- Section 3 – Integration Architecture
This section provides an overview of the integration architecture, including the Release 1 legacy systems, interfaces to the legacy systems, and interfaces to SFA's Integrated Technical Architecture (ITA) Internet architecture. Data flow and message flow diagrams are also documented.
- Section 4 – Execution Architecture Components
This section describes the standard MQSeries Messaging architecture components utilized for Release 1 of the EAI Core Architecture.

- Section 5 – Development Architecture

This section describes the MQSeries Integrator (MQSI) components utilized for the Release 1 EAI Core Architecture.

2 EAI CORE ARCHITECTURE SYSTEMS AND COMPONENTS OVERVIEW

SFA will provide all necessary development, testing, and production environment hardware including all required development and production software licenses. The following table summarizes the Release 1 legacy systems at SFA and their environments and the EAI software components installed on each system.

System Name	Hardware Platform	Operating System Platform	Database / Software	EAI Components
CPS	<ul style="list-style-type: none"> IBM 9672 R35 	<ul style="list-style-type: none"> OS/390 V2.8 (Put Level 9907) 	<ul style="list-style-type: none"> DB2 V5.1 COBOL390 2.1.0 (HCKVB00) 	<ul style="list-style-type: none"> MQSeries Messaging V5.2 Server MQ CICS Bridge Adapter DPL Bridge Trigger Monitor MQ Batch Adapter
NSLDS	<ul style="list-style-type: none"> IBM 9672 R85 	<ul style="list-style-type: none"> OS/390 V2.8 (Put Level 9907) 	<ul style="list-style-type: none"> DB2 V5.1 COBOL390 2.1.0 (HCKVB00) 	<ul style="list-style-type: none"> MQSeries Messaging V5.2 Server MQ CICS Bridge Adapter DPL Bridge Trigger Monitor MQ Batch Adapter
NSLDS Web Server	<ul style="list-style-type: none"> Compaq 	<ul style="list-style-type: none"> NT 4.0 	<ul style="list-style-type: none"> Cool:Gen IDE 	<ul style="list-style-type: none"> MQSeries Messaging V5.2 Server
PEPS	<ul style="list-style-type: none"> HP 	<ul style="list-style-type: none"> HP-UX 10.x 	<ul style="list-style-type: none"> Oracle RDBMS Oracle Forms COBOL C++ 	<ul style="list-style-type: none"> MQSeries Messaging V5.2 Server
PEPS – Development (CBMI facility)	<ul style="list-style-type: none"> HP 	<ul style="list-style-type: none"> HP-UX 10.x 	<ul style="list-style-type: none"> Oracle RDBMS Oracle Forms COBOL C++ 	<ul style="list-style-type: none"> MQSeries Messaging V5.2 Server
DLSS	<ul style="list-style-type: none"> Dec Alpha 	<ul style="list-style-type: none"> Open VMS 	<ul style="list-style-type: none"> Oracle RDB COBOL C++ 	<ul style="list-style-type: none"> MQSeries Messaging V2.2.1.1 Server
Btrade	<ul style="list-style-type: none"> HP 	<ul style="list-style-type: none"> HP-UX 11.x 	<ul style="list-style-type: none"> Oracle RDBMS Java 	<ul style="list-style-type: none"> MQSeries Messaging V5.2 Server
SU35E5	<ul style="list-style-type: none"> Sun 	<ul style="list-style-type: none"> Solaris 2.6 	<ul style="list-style-type: none"> WebSphere V3.5 	<ul style="list-style-type: none"> MQSeries Messaging V5.2 Server
SU35E14	<ul style="list-style-type: none"> Sun 	<ul style="list-style-type: none"> Solaris 2.7 	<ul style="list-style-type: none"> DB2 (Included with MQSI) 	<ul style="list-style-type: none"> MQSeries Integrator V2.0.1 MQSeries Messaging V5.2 Server (Included with MQSI)
SU35E16	<ul style="list-style-type: none"> Sun 	<ul style="list-style-type: none"> Solaris 2.7 	<ul style="list-style-type: none"> DB2 (Included with MQSI) 	<ul style="list-style-type: none"> MQSeries Integrator V2.0.1 MQSeries Messaging V5.2 Server (Included with MQSI)
SU35E17	<ul style="list-style-type: none"> Sun 	<ul style="list-style-type: none"> Solaris 2.7 	<ul style="list-style-type: none"> DB2 	<ul style="list-style-type: none"> MQSeries Integrator V2.0.1

System Name	Hardware Platform	Operating System Platform	Database / Software	EAI Components
			(Included with MQSI)	<ul style="list-style-type: none"> MQSeries Messaging V5.2 Server (Included with MQSI)
SFANT006	<ul style="list-style-type: none"> Compaq 	<ul style="list-style-type: none"> NT 4.0 	<ul style="list-style-type: none"> DB2 (Included with MQSI) 	<ul style="list-style-type: none"> MQSeries Integrator V2.0.1 MQSeries Messaging V5.1 Server (Included with MQSI)
Rational Server	<ul style="list-style-type: none"> Compaq 	<ul style="list-style-type: none"> NT 4.0 	<ul style="list-style-type: none"> DB2 (Included with MQSI) 	<ul style="list-style-type: none"> MQSeries Integrator V2.0.1 MQSeries Messaging V5.1 Server (Included with MQSI)
Development Workstations (3) at Accenture	<ul style="list-style-type: none"> Compaq 	<ul style="list-style-type: none"> NT 4.0 	<ul style="list-style-type: none"> DB2 (Included with MQSI) 	<ul style="list-style-type: none"> MQSeries Integrator V2.0.1 MQSeries Messaging V5.1 Server (Included with MQSI)

The following products have been utilized in the design, build and test of the Release 1 EAI Core Architecture:

- DB2 database v6.1 has been installed on the EAI Bus Sun Servers and the MQSI NT Servers. This product is included with the full installation of MQSI and does not require an additional license as long as it is only used for MQSI.
- WebSphere Application Server v3.5 utilized on the WAS Sun boxes in development and test.
- MQSeries v5.2 has been installed on all systems except for DLSS, which has MQSeries v2.2.1.1 installed.
- MQSeries Integrator v2.0.1 has been installed on the EAI Bus Sun Servers, the MQSI NT Servers and the MQSI development workstations.

The versions of these products were chosen based on the analysis of the five Legacy Systems used for the EAI Core Architecture Release 1. It had been determined that each Legacy System had the required prerequisites necessary to install the software versions listed above. These software versions were the most currently supported software products from IBM at the time of the analysis.

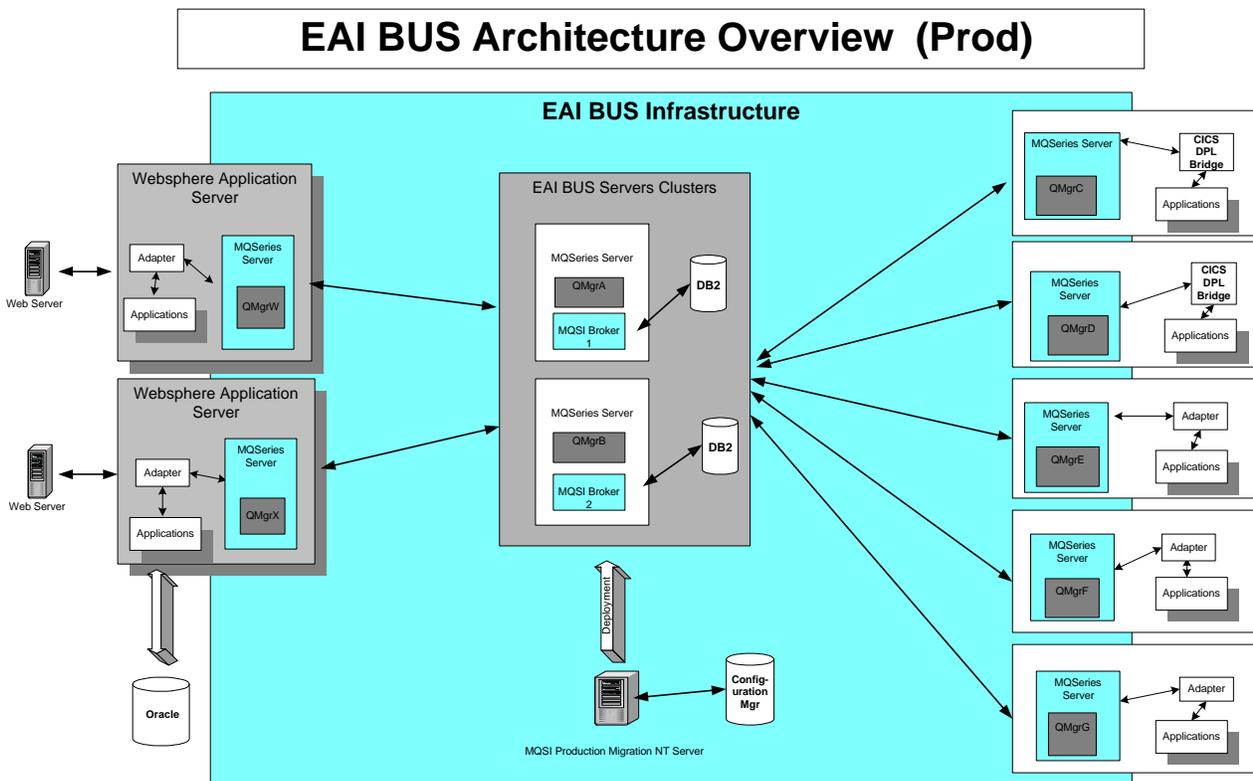
3 INTEGRATION ARCHITECTURE

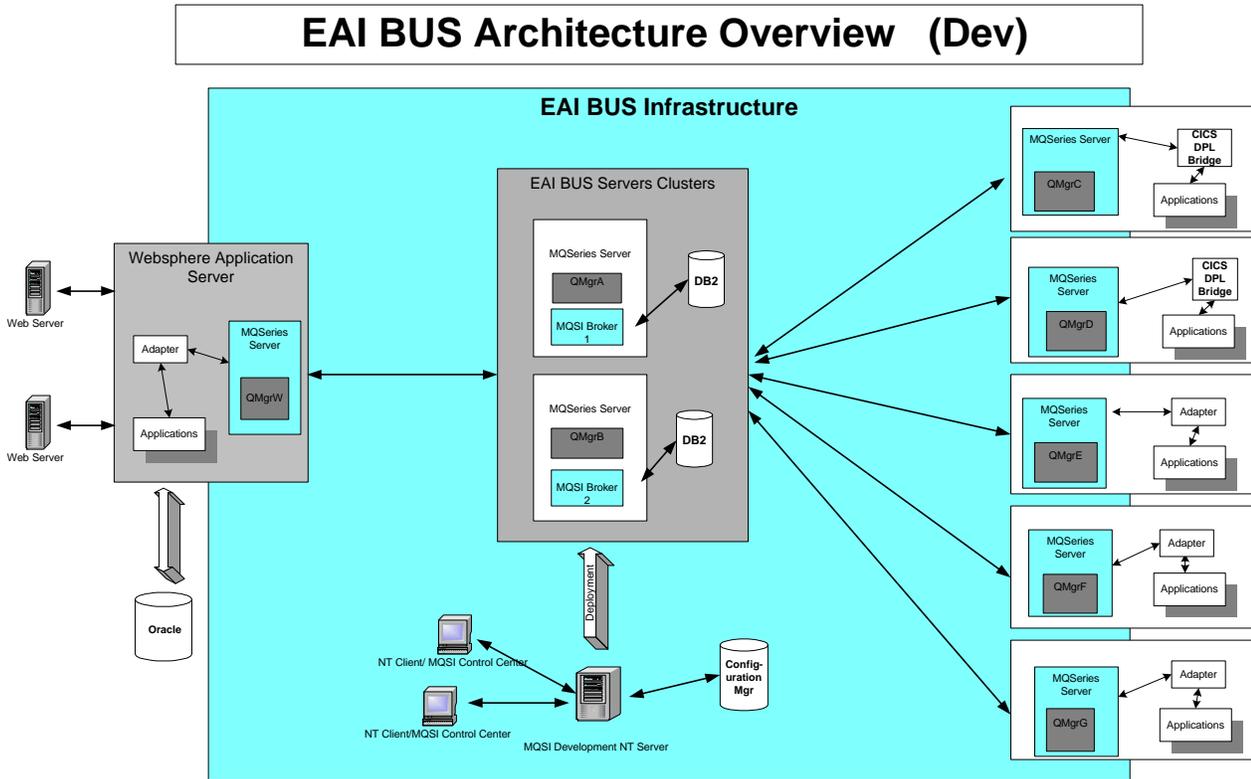
This section provides an overview of the integration architecture, including the Release 1 legacy systems, interfaces to the legacy systems, and interfaces to SFA's Integrated Technical Architecture (ITA) Internet architecture. Additionally, this section provides descriptions of the supporting applications used to satisfy the design requirements for site development and operation.

The SFA Enterprise Application Integration Core Architecture consists of the following four primary areas:

- Legacy Systems – Mainframe and Mid-Tier
- Internet – WebSphere Application Server
- EAI Bus Servers
- EAI Development Workstations

The following diagrams provide a high level view of the EAI Architecture for both the Development and Production environments at SFA.





Each of the main areas of the architecture is described in the following sections.

3.1 INTERFACE DEFINITIONS

Existing and future SFA applications and business capabilities will rely on the EAI Bus to retrieve and send data to and from legacy systems. This section provides the interface definitions and process for using the EAI Bus to connect to each of the Release 1 legacy systems. For each Release 1 legacy system a diagram depicting the architecture components, message flow and EAI components (wrappers, bridges, connectors and adapters) will be documented.

3.2 MAINFRAME LEGACY SYSTEMS

The Release 1 Mainframe systems for the EAI Core include CPS and NSLDS. On each of these systems the following EAI components have been installed,

- MQSeries Messaging
- MQ CICS Bridge Adapter
- MQ DPL Adapter
- MQ Batch Adapter
- Trigger Monitor
- Custom Developed MQ Adapters

Each of these components provides the core infrastructure for enabling SFA applications to connect to the EAI bus. The choice and use of which adapter is dependent upon the application functionality

requirements. The list of adapters developed for Release 1 for the mainframe systems are defined in Appendix A.

During the due diligence phase of EAI Core Release 1, it was determined that the software prerequisites to use the MQSeries CICS 3270 Bridge were not met for NSLDS nor CPS. Based on the findings, the MQSeries CICS 3270 Bridge has been excluded from the SFA EAI Core. The two prerequisites necessary for the installation of the MQSeries CICS 3270 Bridge are listed below:

- MQSeries for MVS/ESA Version 1.2 with APAR PQ13387
- CICS Transaction Server for OS/390 Release 2 with APARs PQ13011 and PQ13012

3.2.1 CPS EAI System Overview

The CPS system executes in a CICS (Customer Information Control System) environment. CPS provides the functionality to execute CICS developed transactions to query/request data on the system. A sample CPS CICS transaction has been identified for validation of the EAI Bus connectivity to the CPS legacy system. A request will be sent from the source server via MQSeries, transformed using MQSI, and sent to CPS for execution. The results of the transaction will be sent back to the calling source and displayed/written to a file. This test will validate the installation and configuration of EAI connectivity for CPS. The required EAI components, MQ Adapters, MQSeries CICS Bridge adapter and trigger monitor will be installed and configured on the CPS system to validate the functionality.

The pilot application supports the Loan Application Status functionality. A Request/Reply message type will be used to test the integration of the pilot application. A message will be sent from the source server and transported through the EAI Bus via MQSeries. The message data will be transformed in the EAI Bus using the MQSI software into the expected message data format of the target CICS transaction. The transformed message data will be routed to the CPS mainframe and retrieved for execution. The results of the request will be sent back to the source server for display.

3.2.1.1 CPS Messaging Components

The following messaging components are used to support the CPS sample transaction for the EAI Core Architecture Release 1. Also, the additional adapters installed and configured are described too.

- The MQSeries with the DPL (Distributed Program Link) adapter supports the integration to CICS programs (applications) which do not use any terminal related CICS commands. These CICS programs support communication through the CICS COMMAREA.
- The MQSeries CICS Bridge adapter supports the connectivity of MQSeries Queue Managers to CICS regions.
- The MQSeries OS/390 Batch Trigger Monitor and two custom-built batch adapters have been installed and configured as part of the core architecture.

The following section defines the EAI components and technical specifications for the design and development of the CPS sample transaction.

3.2.1.2 MQSeries Provided Adapters

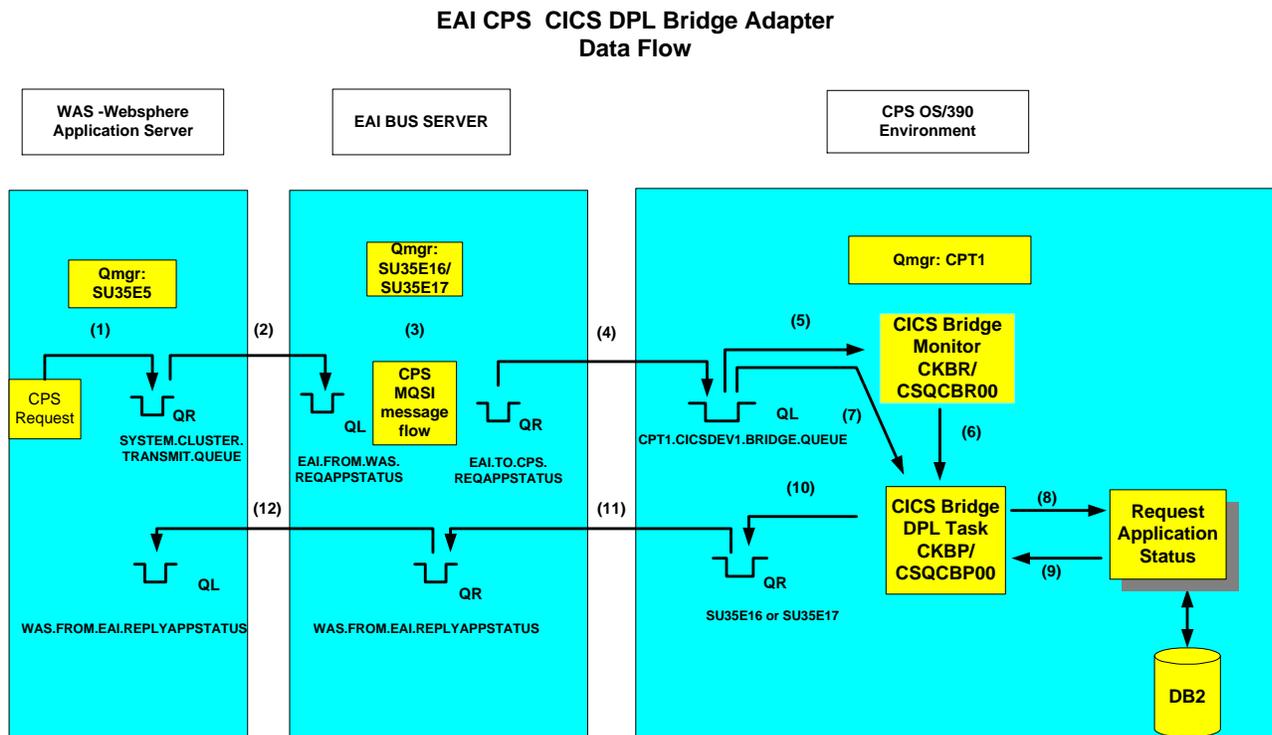
The MQSeries CICS Bridge – DPL adapter was an out-of-the box adapter used by the CPS system.

3.2.1.3 CPS Custom Built Adapters

No custom built adapters are required for CPS as part of the Enterprise Application Integration Core Architecture Release 1.

3.2.1.4 CPS Data Flow and Message Flow Diagrams

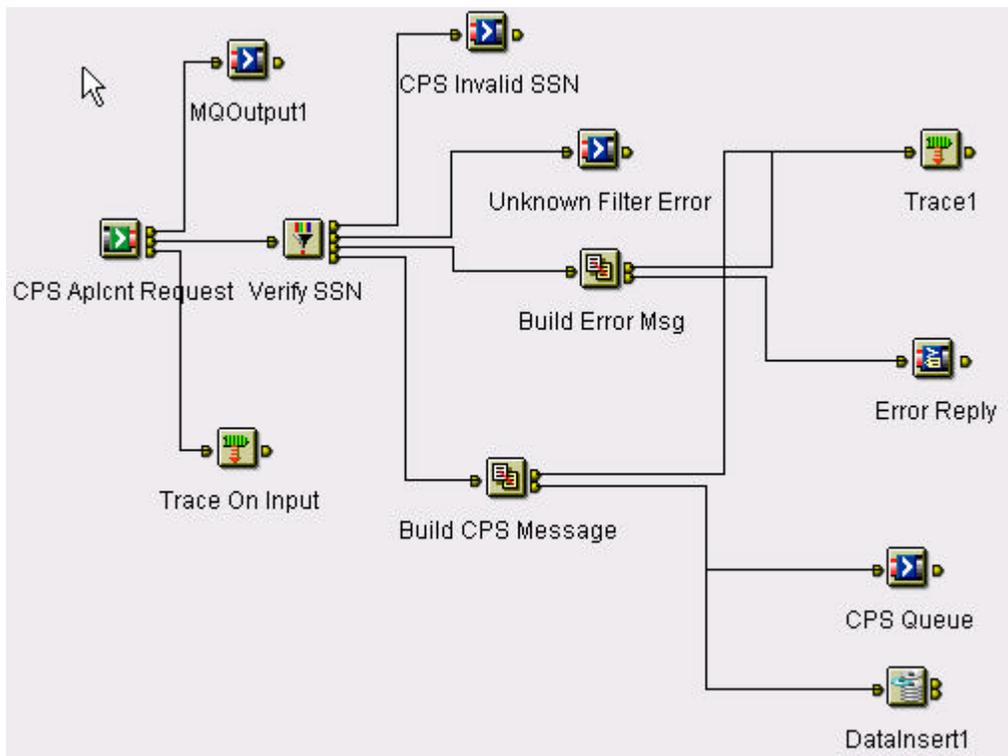
The figure below describes the message flow through the CICS DPL Bridge Adapter.



The flow of a MQSeries Request type message through the EAI CPS DPL Bridge Adapter Request Design is as follows:

- 1) A CPS MQSeries Request type message is put to the Cluster Queue EAI.FROM.WAS.REQAPPSTATUS from the WAS box.
- 2) The MQSeries Queue Manager (SU35E5) on the WAS moves the message to the Local Queue EAI.FROM.WAS.REQAPPSTATUS.
- 3) The message is pulled from the EAI.FROM.WAS.REQAPPSTATUS and processed through the CPS MQSI Message Flow and data transformation.
- 4) The output message from the CPS MQSI Message Flow is put to the Remote Queue EAI.TO.CPS.REQAPPSTATUS.
- 5) The MQSeries Queue Manager (SU35E16/SU35E17) on the EAI Bus server moves the message to the Local Queue CPT1.CICSDEV1.BRIDGE.QUEUE.

- 6) The MQSeries CICS Bridge Monitor (CKBR/CSQCBR00) monitors the CPT1.CICSDEV1.BRIDGE.QUEUE to see if there are any messages to be processed. If a message has arrived, the CICS Bridge Monitor starts the CICS Bridge DPL Task (CKBP/CSQCBP00).
- 7) The CICS Bridge DPL Task will get a message from the CPT1.CICSDEV1.BRIDGE.QUEUE.
- 8) The CICS Bridge DPL Task will do a Distributed Program Link(DPL) to the CPS pilot application, passing the message in the CICS COMMAREA.
- 9) The CPS pilot application will return the response data in the CICS COMMAREA when finished.
- 10) The CICS Bridge DPL Task will put the reply message to the transmission Queue for SU35E16 or SU35E17.
- 11) The MQSeries Queue Manager (CPT1) on the CPST LPAR moves the message to the Remote Queue WAS.FROM.EAI.REPLYAPPSTATUS.
- 12) The MQSeries Queue Manager (SU35E16/SU35E17) on the EAI Bus server moves the message to the Local Queue WAS.FROM.EAI.REPLYAPPSTATUS.



CPS Message Flow

Node	Type	Description/Function
CPS Aplcnt Request	MQInput	Gets message from input queue EAI.FROM.WAS.REQAPPSTATUS
MQOutput1	MQOutput	Puts message to queue CORE.CPS.BIGERROR
Trace on Input	Trace	Used to trace flow
Verify SSN	Filter	Checks if SSN <> 'XXXXXXXXXX' If True goto Build CPS Msg

Node	Type	Description/Function
		If False goto Build Error Msg If Failure goto CPS Invalid SSN If Unknown goto Unknown Filter Error
Build CPS Msg	Compute	Append Program name, SSN and NameID to Message
Build Error Msg	Compute	Create error message
CPS Invalid SSN	MQOutput	Puts message to CORE.CPS.FAILURE queue
Unknown Filter Error	MQOutput	Puts message to CORE.CPS.UNKNOWN queue
Trace1	Trace	Traces flow
Error Reply	MQReply	Sends response to the originator of the message
CPS Queue	MQOutput	Puts message to EAI.TO.CPS.REQAPPSTATUS queue
DataInsert1	DataInsert	Database node that allows insertion to a ODBC data source

3.2.1.5 CPS MQSeries Programming Sample Source Code

There are sample MQSeries programs that can be used as templates. The samples are provided with the product and have been installed within the MQM.V5R2M0.* libraries in the CPS TSO environment and can be used as reference for future EAI application enablement on the CPS system.

3.2.2 NSLDS EAI System Overview

The NSLDS Release 1 EAI Core Architecture provides two distinct EAI architecture components, a batch process and a CICS Cool:Gen Transaction process. The NSLDS system executes in a batch TSO (time sharing option) environment. In addition, the NSLDS system also has CICS applications built using the Cool:Gen development environment tool. For the NSLDS system, two sample functions will be validated, the execution of a batch program and the execution of a transaction. A request will be sent to the NSLDS system to execute a sample batch program which will validate the installation and configuration of the required EAI components to execute batch programs on the mainframe system. These components include the MQ Batch Adapter and the trigger monitor. For the NSLDS Cool:Gen transaction, a request will be sent via MQSeries from the NSLDS web server to the NSLDS mainframe for execution. The results of the request will be sent back to the calling source and displayed on the browser. This test will require the design and development of Cool:Gen MQ Adapters.

3.2.2.1 NSLDS Messaging Components

The following messaging components are used to support the sample application for the Enterprise Application Integration Core Architecture Release 1. Additional custom adapters that were installed and configured are also described.

- The MQSeries CICS Bridge with the DPL (Distributed Program Link) adapter cannot be used for any NSLDS CICS applications since it was developed with the Cool:Gen product, but it was installed as part of the core architecture.
- The Cool:Gen product used on NSLDS provides a tool to generate the adapter components required to provide integration to the CICS programs (in Cool:Gen terms; servers).
- The MQSeries CICS adapter supports the connectivity of MQSeries Queue Managers to CICS regions.
- The MQSeries OS/390 Batch Trigger Monitor and two custom-built batch adapters support an NSLDS batch file transfer from/to other systems.

A Request/Reply message type will be used to test the integration of the sample functionality. The adapters support Request/Reply message types for the sample application to pull data from NSLDS and provide the response back to the EAI Bus.

The following two sections define the NSLDS architecture for the NSLDS batch program architecture and message flow/processing and the architecture and message flow/processing for the CICS Cool:gen transaction.

3.2.2.2 MQSeries Provided Adapters

The following sections define the EAI components for the NSLDS Batch Process.

NSLDS MQSeries CICS Adapter

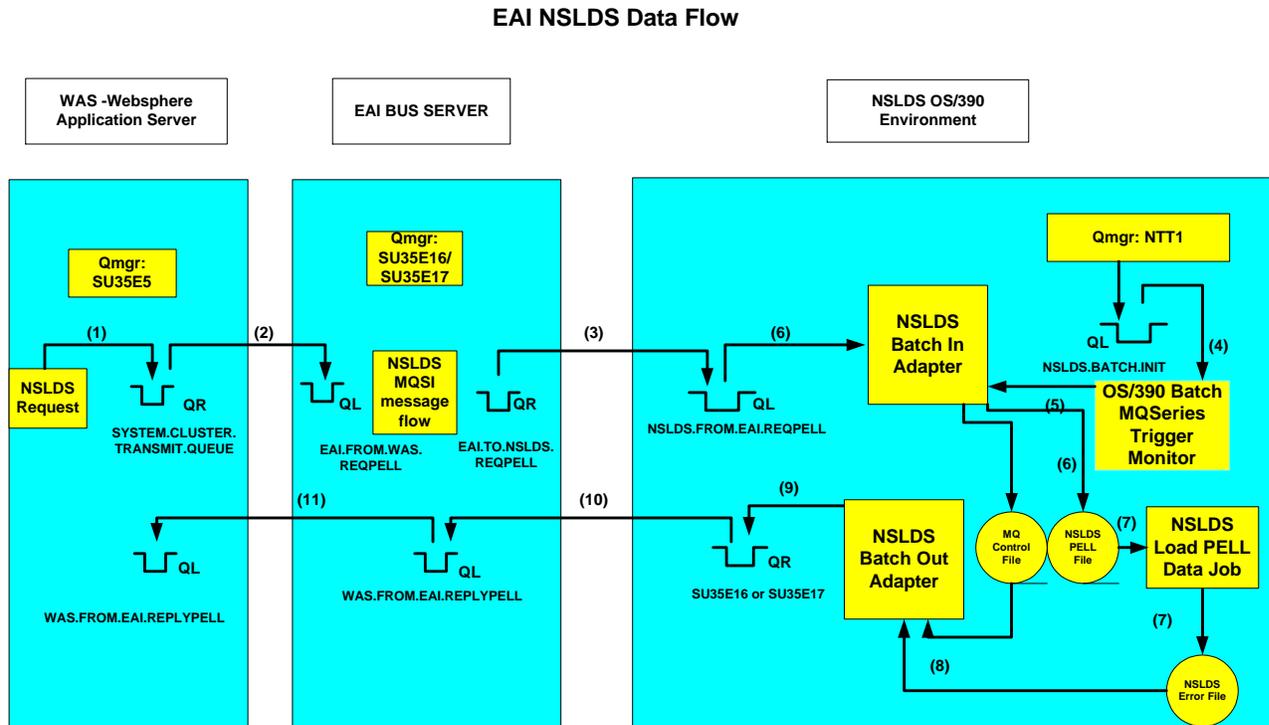
The MQSeries CICS adapter is installed and configured within the parameters of CICS regions. The MQSeries NSLDS CICS Adapter is not functional for Release 1, per the reasons defined in Section 3.2.

The MQSeries OS/390 Trigger Monitor is an out-of-the-box adapter. It is described below and is used as part of the custom adapter test flow. The figure below describes the message flow through the MQSeries OS/390 Trigger Monitor.

3.2.2.3 NSLDS Custom Built Adapters

Custom batch adapters were built to validate the NSLDS batch EAI Core functionality. These adapters were utilized to process receipt of a file on the NSLDS system, execute a batch process, and read the output of the batch file, sending the results back to the calling source application. The batch adapters are defined in Appendix A. The source code for the NSLDS MQ adapters is stored in the ClearCase repository.

3.2.2.4 NSLDS Data Flow and Message Flow Diagrams

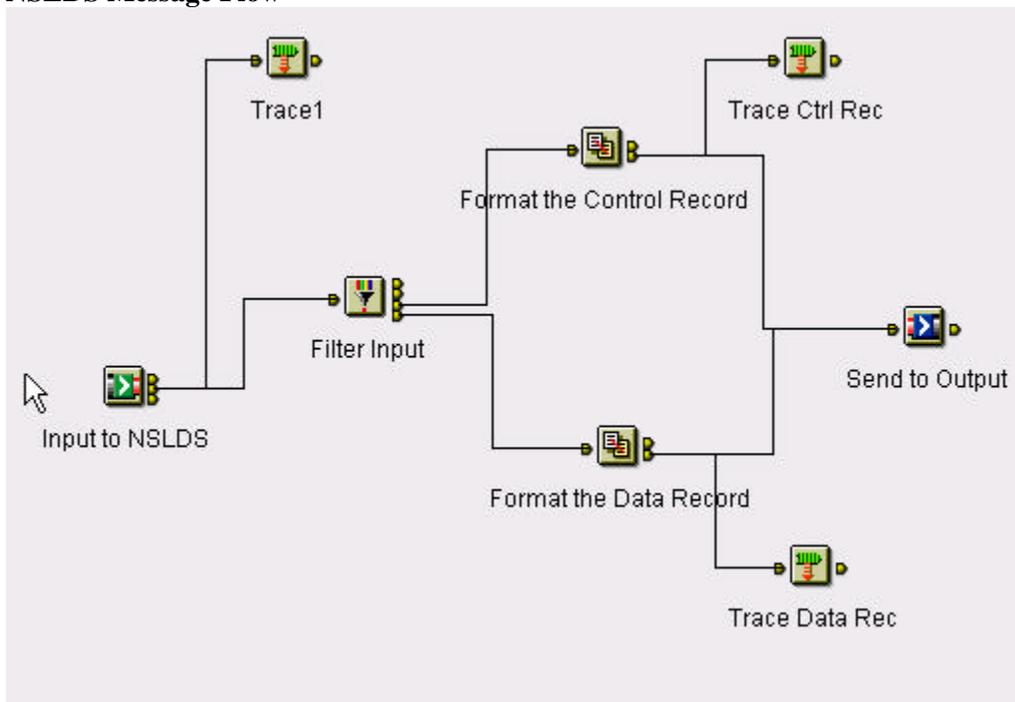


The flow of a MQSeries Request type message through the EAI System is as follows:

- 1) A MQSeries Request type message is put to the Cluster Queue EAI.FROM.WAS.REQPELL from the WAS box.
- 2) The MQSeries Queue Manager (SU35E5) on the WAS moves the message to the Local Queue EAI.FROM.WAS.REQPELL. The message is pulled from the EAI.FROM.WAS.REQPELL and processed through the NSLDS – Pell MQSI Message Flow.
- 3) The output message from the MQSI Message Flow is put to the Remote Queue EAI.TO.NSLDS.REQPELL. The MQSeries Queue Manager (SU35E16/SU35E17) on the EAI Bus server moves the message to the Local Queue NSLDS.FROM.EAI.REQPELL and based on the attributes set up in the queue, the MQSeries Queue Manager (NTT1) puts a trigger message on the NSLDS.BATCH.INIT
- 4) The OS/390 Batch MQSeries Trigger Monitor application pulls the trigger message on the NSLDS.BATCH.INIT queue.
- 5) The MQSeries Trigger Monitor application starts the NSLDS Batch In Adapter program.
- 6) The NSLDS Batch In Adapter program pulls the messages from the queue. It then creates the NSLDS PELL File and the MQ Control File.

- 7) Then the NSLDS Load Pell Data job will be started via CA7 to process the NSLDS PELL Fil. The file output from this job will be the NSLDS Error file.
- 8) The last step in the NSLDS Load Pell Data job will execute the NSLDS Batch Out Adapter.
- 9) The NSLDS Batch Out Adapter will read the MQ Control file (to know where to send the reply to) and push the NSLDS Error file into the transmission queue for SU35E16 or SU35E17.
- 10) The message is moved to the queue WAS.FROM.EAI.REPLYPELL on the EAI bus.
- 11) The MQSeries Queue Manager on the EAI bus moves the message to WAS.FROM.EAI.REPLYPELL.

NSLDS Message Flow



Node	Type	Description/Function
Input to NSLDS	MQInput	Gets message from the input queue – EAI.FROM.WAS
Trace1	Trace	Trace flow
Filter Input	Filter	Checks if Input is a Control Record or a Data Record. If Data Record – go to Format the Data Record If Control Record – go to Format the Control Record
Format the Control Record	Compute	Formats the record
Format the Data Record	Compute	Formats the record
Send to Output	MQOutput	Puts message to EAI.TO.NSLDS queue
Trace Ctrl Rec	Trace	Traces output flow of Trace Record
Trace Data Rec	Trace	Traces output flow of Trace Data Record

3.2.2.5 NSLDS MQSeries Programming Sample Source code

There are sample MQSeries programs that can be used as templates. The samples are provided with the product and have been installed within the MQM.V5R2M0.* library in the CPS TSO environment and can be used as reference for future EAI application enablement on the NSLDS system.

3.2.2.6 NSLDS MQSeries OS/390 Trigger Monitor and Custom Cool:Gen Adapters

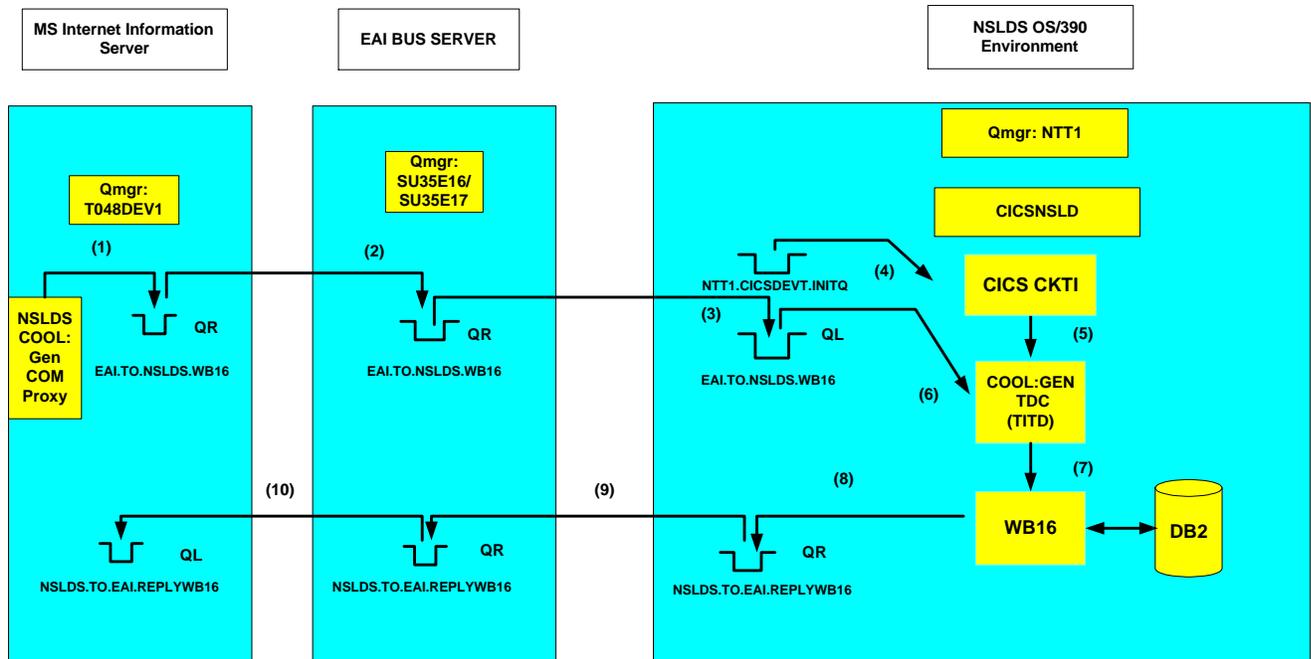
The communications features available with COOL:Gen for connectivity to the IBM Mainframe are as follows:

- With the use of Comm Bridge (TCP/IP or SNA LU6.2)
- Encina to CICS via DPL.
- IBM's MQSeries.
- DCE

The NSLDS Transaction functionality requires the licensing and use of the Cool:Gen development tool suite and a run-time license to execute the Cool:Gen application, on both the source server and the NSLDS mainframe. The Cool:Gen development tools allow the user to develop the Cool:Gen transaction and the required interface component to MQSeries, on both the source and target server. This developed Cool:Gen MQ component is required to provide the interface to the Cool:Gen application, on the source server and the Cool:Gen Transaction program on the target system. Any future SFA EAI application development which utilizes Cool:Gen will require the Cool:Gen development tool suite as well as the Cool:Gen run-time license to execute the Cool:Gen application.

The pre-requisites for developing EAI applications to interface with Cool:Gen must first be identified and installed/configured for each server which requires this functionality. The version of Cool:Gen, development and run-time are dependent upon the hardware and software platforms defined as part of the development and production environments.

**EAI NSLDS COOL:GEN
 Data Flow**



- 1) A request message is put to the Remote Queue EAI.TO.NSLDS.WB16.
- 2) The message is routed through the EAI BUS Server – QMgr (SU35E16 or SU35E17) using the EAI.TO.NSLDS.WB16 queue.
- 3) The COOL:Gen request message arrived in the EAI.TO.NSLDS.WB16 queue on NSLDS OS/390 System QMgr (NTT1). The EAI.TO.NSLDS.WB16 queue is defined and set for triggering.
- 4) On every message arrival, Qmgr NTT1 creates a trigger message based on the information defined on the PROCESS definition and puts it on the NTT1.CICSDEVT.INITQ object.
- 5) The CICS trigger monitor in the CICS region CICSNSLD gets the trigger message, examines its contents and start transaction 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. Communication between MQSeries and CICS program WB1612DS is done through 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 Server – QMgr (SU35E16 or SU35E17) using the NSLDS.TO.EAI.REPLYWB16 queue.
- 10) The reply message arrives at the QMgr – T048DEV1 on the NSLDS web server. COOL:Gen COM Proxy gets the reply message from the queue and displays the result.

3.3 MID-TIER LEGACY SYSTEMS

The Mid-Tier Legacy Systems for Release 1 of the EAI Core architecture that have been identified to build interfaces into the EAI Bus are DLSS, PEPS and bTrade.

3.3.1 DLSS EAI System Overview

The DLSS system executes in batch type environment. A custom MQSeries adapter will be built to allow the DLSS application to process with no application modifications required.

3.3.1.1 DLSS Messaging Components

A custom built adapter is required to support the DLSS Loan application used for the EAI Core Architecture Release 1. A Request/Reply message type will be used to test the integration of the pilot application. The adapter supports a Request/Reply message type for the pilot application to pull data from DLSS and provide the response back to the EAI Bus.

3.3.1.2 MQSeries Provided Adapters

No MQSeries provided Adapters are available to validate the sample representative functionality for the DLSS system as part of the Enterprise Application Integration Core Architecture Release 1.

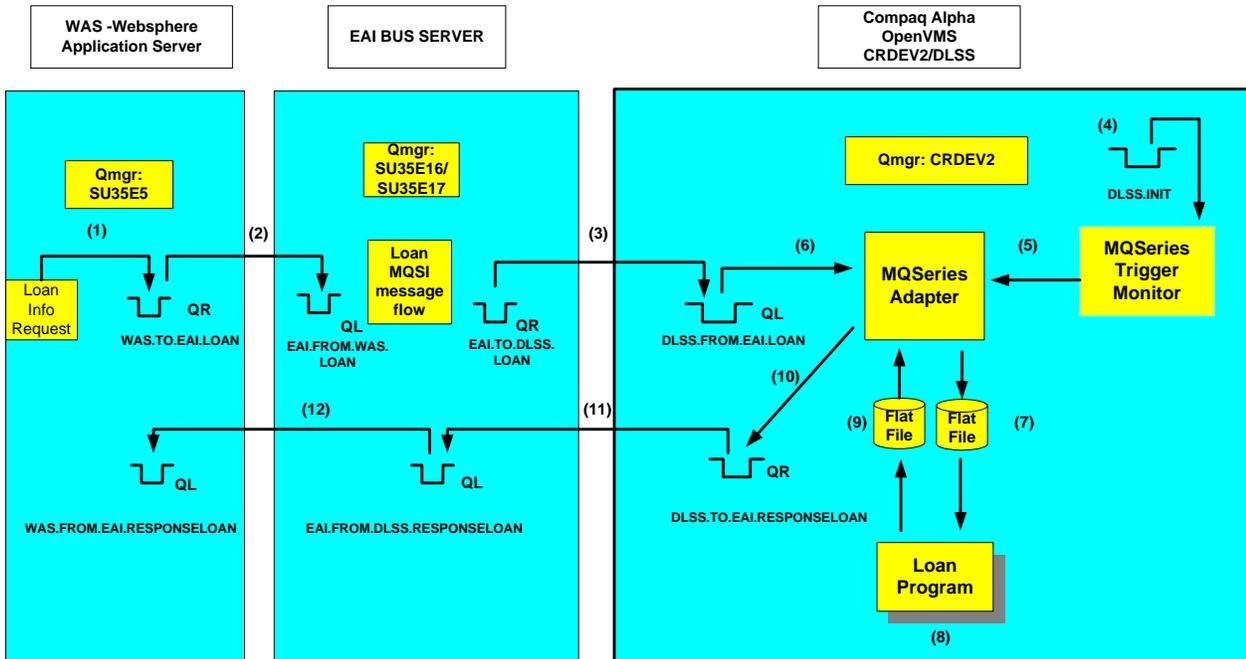
3.3.1.3 DLSS Custom Built Adapters

Two adapters were created for the DLSS system. The first adapter gets messages from a queue and writes it to a file and the second adapter reads data from a file and puts the message to a queue.

3.3.1.4 DLSS Data Flow and Message Flow Diagrams

DLSS Batch Adapter

EAI DLSS Data Flow

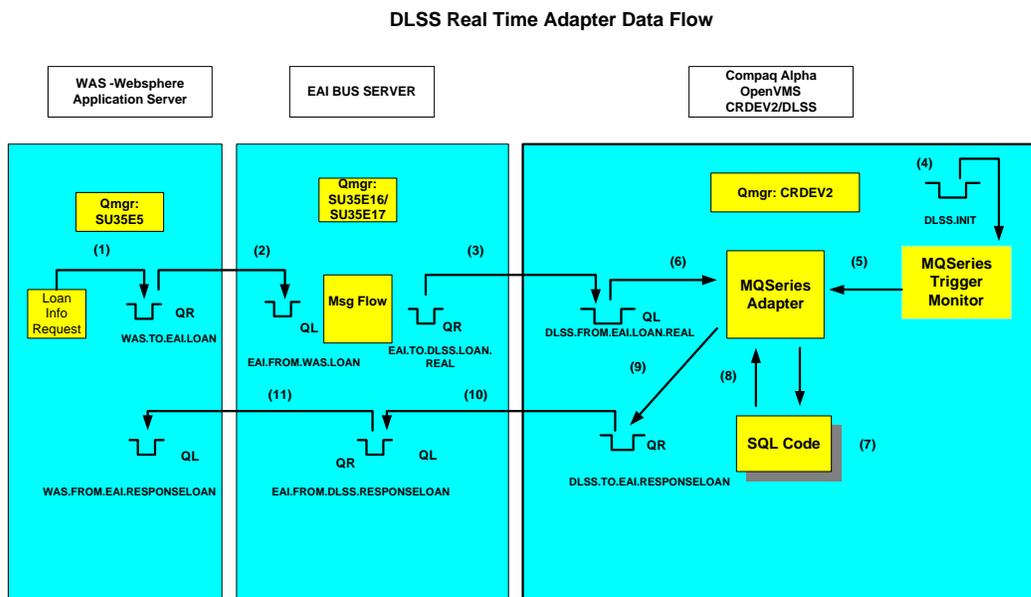


The flow of a MQSeries Request type message through the EAI System is as follows:

- 1) A MQSeries Request type message is put to the Cluster Queue EAI.FROM.WAS.LOAN from the WAS box.
- 2) The MQSeries Queue Manager (SU35E5) on the WAS moves the message to the Local Queue EAI.FROM.WAS.LOAN. The message is pulled from the EAI.FROM.WAS.LOAN and processed through the DLSS – Loan MQSI Message Flow.
- 3) The output message from the MQSI Message Flow is put to the Remote Queue EAI.TO.DLSS.LOAN
- 4) The MQSeries Queue Manager (SU35E16/SU35E17) on the EAI Bus server moves the message to the Local Queue DLSS.FROM.EAI.LOAN and based on the attributes set up in the queue, the MQSeries Queue Manager (CRDEV2) puts a trigger message on the DLSS.INIT
- 5) The MQSeries Trigger Monitor application pulls the trigger message on the DLSS.INIT queue.
- 6) The MQSeries Trigger Monitor application starts the DLSS Loan Wrapper/Adapter program. The program reads the message from the queue and

- 7) Creates a flat file.
- 8) The DLSS Loan application is scheduled to pick up the newly created file. The Loan Application uses the information in the file (SSN & Name) to retrieve detailed loan information.
- 9) The Loan application creates a flat file and calls the Adapter program.
- 10) The adapter program moves the data from the flat file to the transmission queue for SU35E16 and SU35E17.
- 11) The message is moved to the queue WAS.FROM.EAI.RESPONSELOAN on the EAI bus.
- 12) The MQSeries Queue Manager on the EAI bus moves the message to WAS.FROM.EAI.RESPONSELOAN

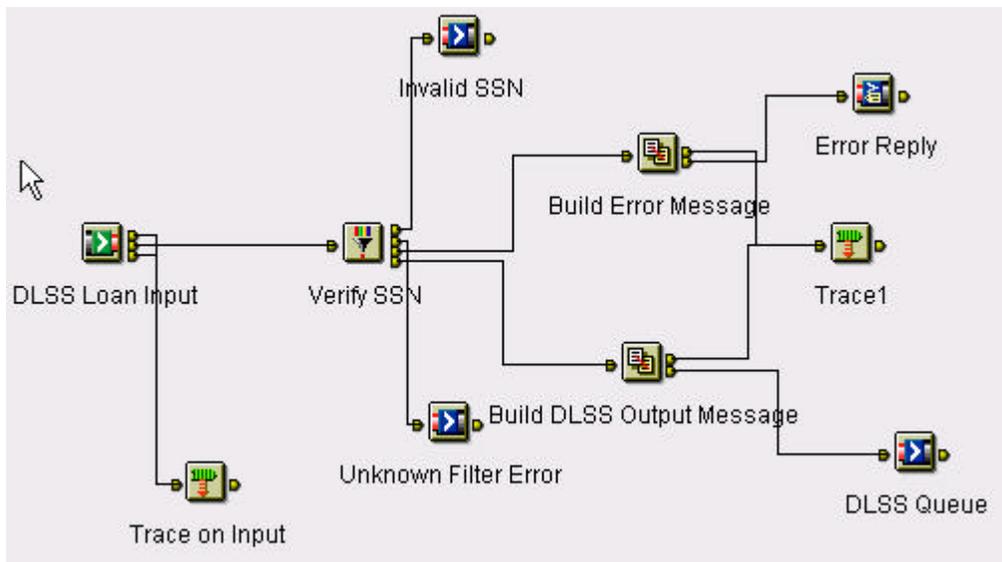
DLSS Real Time Adapter



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

- 1) A MQSeries Request type message is put to the Cluster Queue EAI.FROM.WAS.LOAN from the WAS box.
- 2) The MQSeries Queue Manager (SU35E5) on the WAS moves the message to the Local Queue EAI.FROM.WAS.LOAN. The message is pulled from the EAI.FROM.WAS.LOAN and processed through the DLSS –Message Flow.
- 3) The output message from the MQSI Message Flow is put to the Remote Queue EAI.TO.DLSS.LOAN.REAL

- 4) The MQSeries Queue Manager (SU35E16/SU35E17) on the EAI Bus server moves the message to the Local Queue DLSS.FROM.EAI.LOAN.REAL and based on the attributes set up in the queue, the MQSeries Queue Manager (CRDEV2) puts a trigger message on the DLSS.INIT
- 5) The MQSeries Trigger Monitor application pulls the trigger message on the DLSS.INIT queue.
- 6) The MQSeries Trigger Monitor application starts the DLSS Real time/Adapter program. The program reads the message from the queue and
- 7) SQL code is executed
- 8) The DLSS real time adapter retrieves the resulting data from the sql call
- 9) The adapter program moves puts the data to the queue.
- 10) The message is moved to the queue WAS.FROM.EAI.RESPONSELOAN on the EAI bus.
- 11) The MQSeries Queue Manager on the EAI bus moves the message to WAS.FROM.EAI.RESPONSELOAN



DLSS Message Flow

Node	Type	Description/Function
DLSS Loan Input	MQInput	Gets message from EAI.FROM.WAS.LOAN queue
Trace on Input	Trace	Traces Flow
Verify SSN	Filter	Checks value of SSN <> 'XXXXXXXX' If True goto Build DLSS Output Message If False goto Build Error Message If Unknown goto Unknown Filter Error If Failure goto Invalid SSN
Build DLSS Output Message	Compute	Builds DLSS Loan message
Build Error Message	Compute	Builds Error Message
Unknown Filter Error	MQOutput	If Unknown Filter Error - Puts message to queue FLOW2.UNKNOWN
Invalid SSN	MQOutput	If failure on validation, message is put to queue FLOW2.FAILURE

Node	Type	Description/Function
Error Reply	MQReply	Sends message to the originator of the message
Trace1	Trace	Traces flow
DLSS Queue	MQOutput	Puts message to queue EAI.TO.DLSS.LOAN

3.3.1.5 DLSS MQSeries Programming Sample Source Code

There are sample MQSeries programs that can be used as templates. The samples are provided with the product and have been installed on the system. The sample programs are located in the directory pointed to by the system logical mqs_examples on the DLSS system. The samples can be used as reference for future EAI application development on the DLSS system.

3.3.2 PEPS EAI System Overview

The PEPS system uses Oracle Forms to access data from a database. There currently does not exist an Oracle Forms adapter for MQSeries. The decision to validate the EAI connectivity for the PEPS system was to create stored procedures on the PEPS server to replicate the business application logic built into the Oracle Forms. A stored procedure will be used as the sample application that will be built for the test of the PEPS interface to the EAI Bus. A custom MQSeries adapter will be required to access the sample application using the stored procedure.

3.3.2.1 PEPS Messaging Components

A custom built adapter is required to support the sample application used for the Release 1 EAI Core Architecture. The sample application supports the OPE Id School Eligibility Request functionality. A Request/Reply message type will be used to test the integration of the sample application. The adapter supports a Request/Reply message type for the sample application to pull data from PEPS and provide the response back to the EAI Bus.

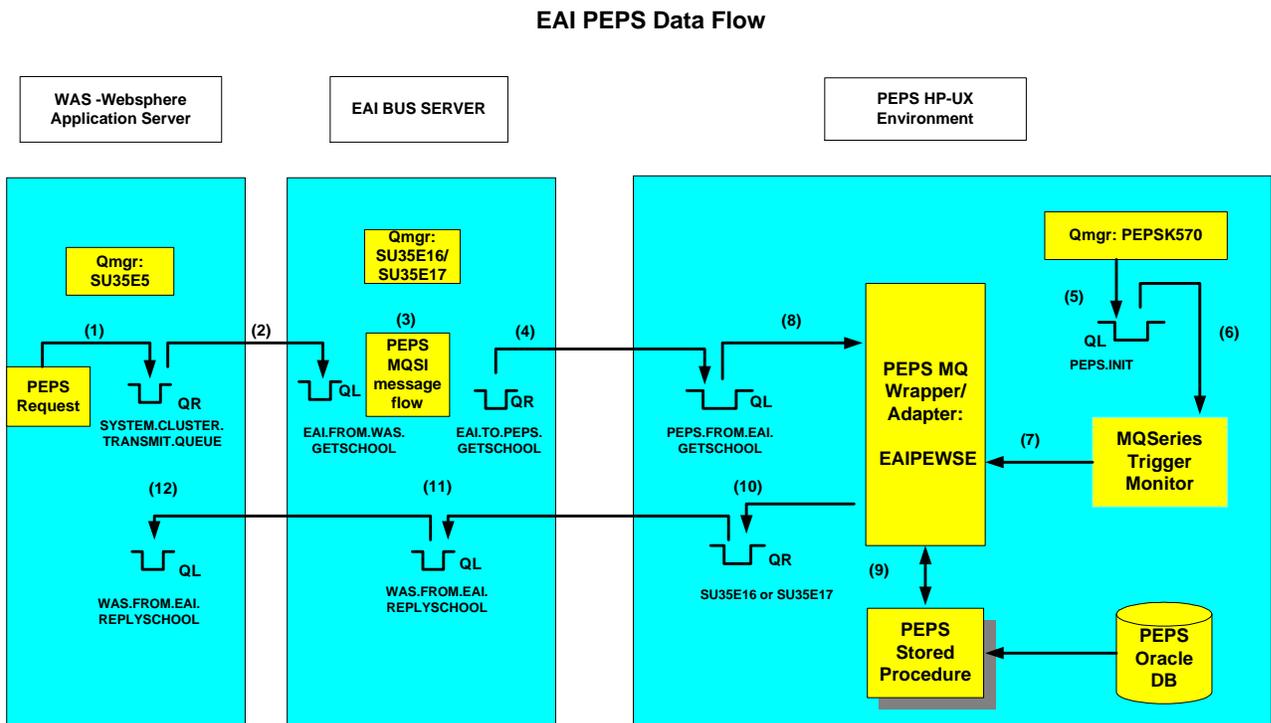
3.3.2.2 MQSeries Provided Adapters

No MQSeries provided adapters have been identified as part of the Enterprise Application Integration Core Architecture Release 1. A custom developed MQ Adapter was required for the validation of the PEPS EAI Core sample functionality. The description and function of this adapter is defined in Section 3.3.2.3.

3.3.2.3 PEPS Custom Built Adapters

A MQ custom developed adapter was designed and built to validate the EAI Core functionality for the PEPS system. This adapter is written in Java and provides the functionality to retrieve a message from an input queue on the PEPS system, execute a stored procedure to retrieve data from the PEPS data base, and put the results into an output queue to be sent to the source server. The adapter is defined in Appendix A. The source code for the PEPS custom MQ adapter is stored in the ClearCase repository.

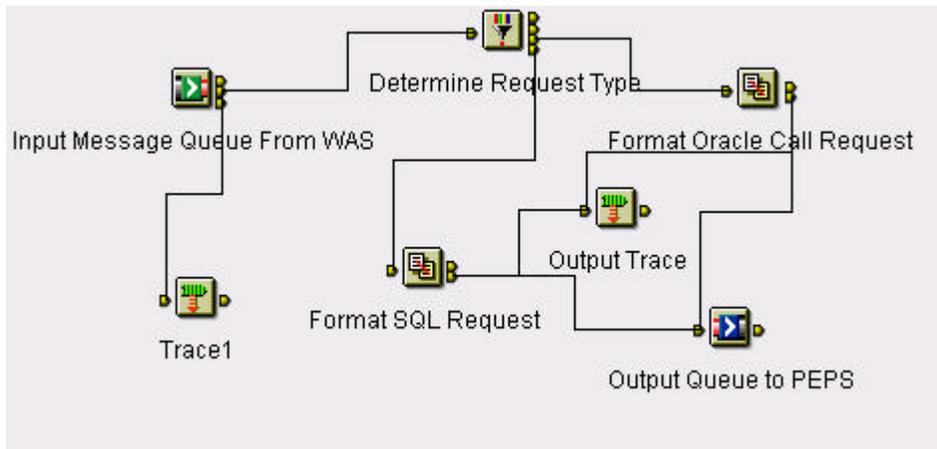
3.3.2.4 PEPS Data Flow and Message Flow Diagrams



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 (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 (SU35E16/SU35E17) on the EAI Bus server moves the message to the Local Queue PEPS.FROM.EAI.GETSCHOOL and based on the attributes set up in the queue, 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 SU35E16 or SU35E17.
- 11) The MQSeries Queue Manager (PEPSK570) on PEPS moves the reply message to the Remote 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.



PEPS Message Flow

Node	Type	Description/Function
Input Message Queue From WAS	MQInput	Gets message from queue EAI.FROM.WAS.GETSCHOOL
Trace1	Trace	Traces flow
Determine Request Type	Filter	Checks if requesttype = 1 If True goto Format SQL Request If False goto Format Oracle Call Request
Format SQL Request	Compute	Builds message
Output Trace	Trace	Traces flow
Format Oracle Call Request	Compute	Builds message
Output Queue to PEPS	MQOutput	Puts message to queue EAI.TO.PEPS.GETSCHOOL

3.3.2.5 PEPS MQSeries Programming Sample Source code

There are sample MQSeries programs that can be used as templates. The samples are provided with the product and have been installed on the system. The sample programs can be found in /home/mqm/peps/.

3.3.3 bTrade EAI System Overview

The bTrade system uses proprietary software that has an API (Application Programming Interface) that will be used in conjunction with a custom developed MQSeries adapter as a sample application for the test of the interface to the EAI Bus. The EAI Core team will develop a function to validate the ability to extract mailbox data from a specified test mailbox on the bTrade server.

3.3.3.1 bTrade Messaging Components

The following messaging components are used to support the bTrade sample application for the Release 1 Enterprise Application Integration Core Architecture. A custom adapter was created for the bTrade system. The sample application used to interface to MQSeries supports the Request for Mailbox Data functionality. A Request/Reply message type will be used to test the integration of the sample application. The adapter supports a Request/Reply message type for the sample application to pull data from a bTrade mailbox and provide the response back to the EAI Bus.

3.3.3.2 MQSeries Provided Adapters

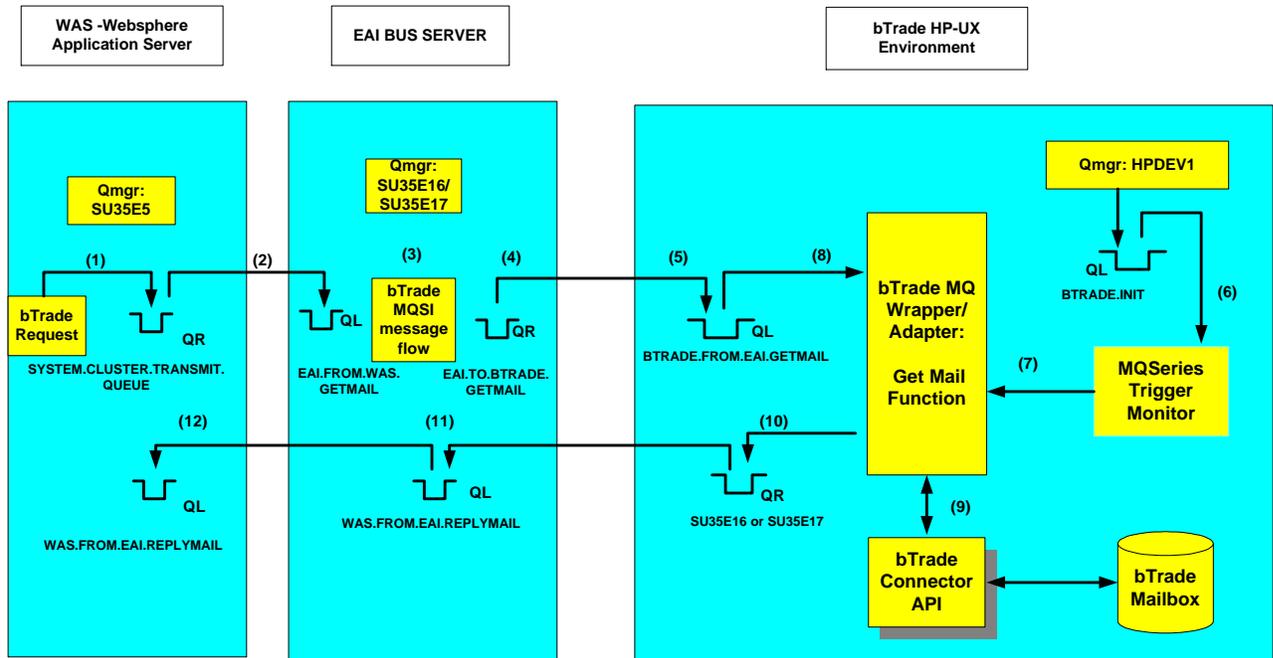
No MQSeries provided adapters have been identified as part of the Release 1 Enterprise Application Integration Core Architecture. A custom developed MQ Adapter was required for the validation of the bTrade EAI Core sample functionality. The description and function of this adapter is defined in Section 3.3.3.3.

3.3.3.3 bTrade Custom Built Adapters

The figure below describes a Request/Reply type message flow through the bTrade custom adapter. This adapter is written in Java. The adapter provides the functionality to retrieve a message from an input queue on the bTrade system, call the bTrade Connector API to retrieve data from a specified mailbox on the bTrade server, and put the results into an output queue to be sent to the source server. The adapter is defined in Appendix A. The source code for the bTrade custom MQ adapter is stored in the ClearCase repository.

3.3.3.4 BTrade Data Flow and Message Flow Diagrams

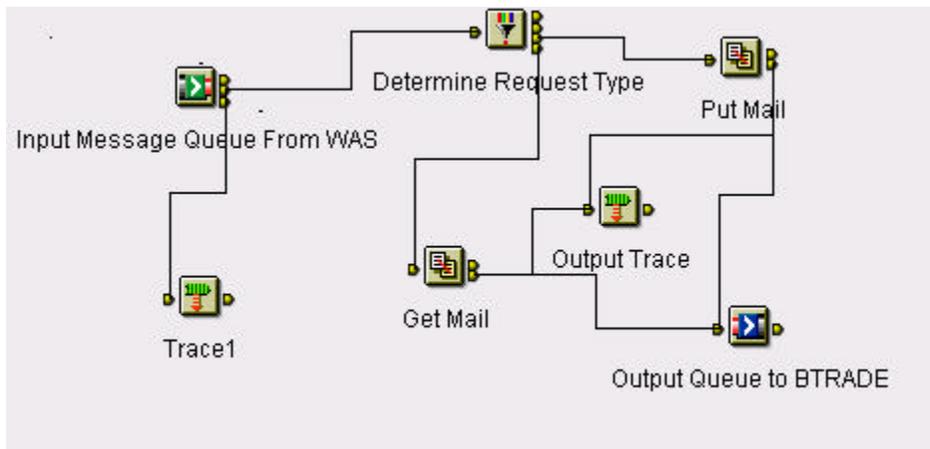
EAI bTrade Data Flow



The flow of a MQSeries Request type message through the EAI 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 (SU35E5) on the WAS moves the message to the Local 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 Remote Queue EAI.TO.BTRADE.GETMAIL.
- 5) The MQSeries Queue Manager (SU35E16/SU35E17) on the EAI Bus server moves the message to the Local 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 SU35E16 or SU35E17.
- 11) The MQSeries Queue Manager (HPDEV1) on bTrade moves the reply message to the Remote Queue WAS.FROM.EAI.REPLYMAIL.
- 12) The MQSeries Queue Manager (SU35E16/SU35E17) on the EAI Bus server moves the reply message to the Local Queue WAS.FROM.EAI.REPLYMAIL.



bTrade Message Flow

Node	Type	Description/Function
Input Message Queue From Was	MQInput	Gets message from the input queue – EAI.FROM.WAS.GETMAIL
Trace1	Trace	Trace flow
Determine Request Type	Filter	Checks if RequestType = '1' , if true goes to Get Mail If false go to Put Mail
Put Mail	Compute	Sets request value = 2, connectorname, and data.
Output Queue to BTRADE	MQOutput	Puts message to EAI.TO.BTRADE.GETMAIL queue
Output Trace	Trace	Traces output flow
Get Mail	Compute	Sets request value =1 and connectorname

3.3.3.5 bTrade MQSeries Programming Sample Source code

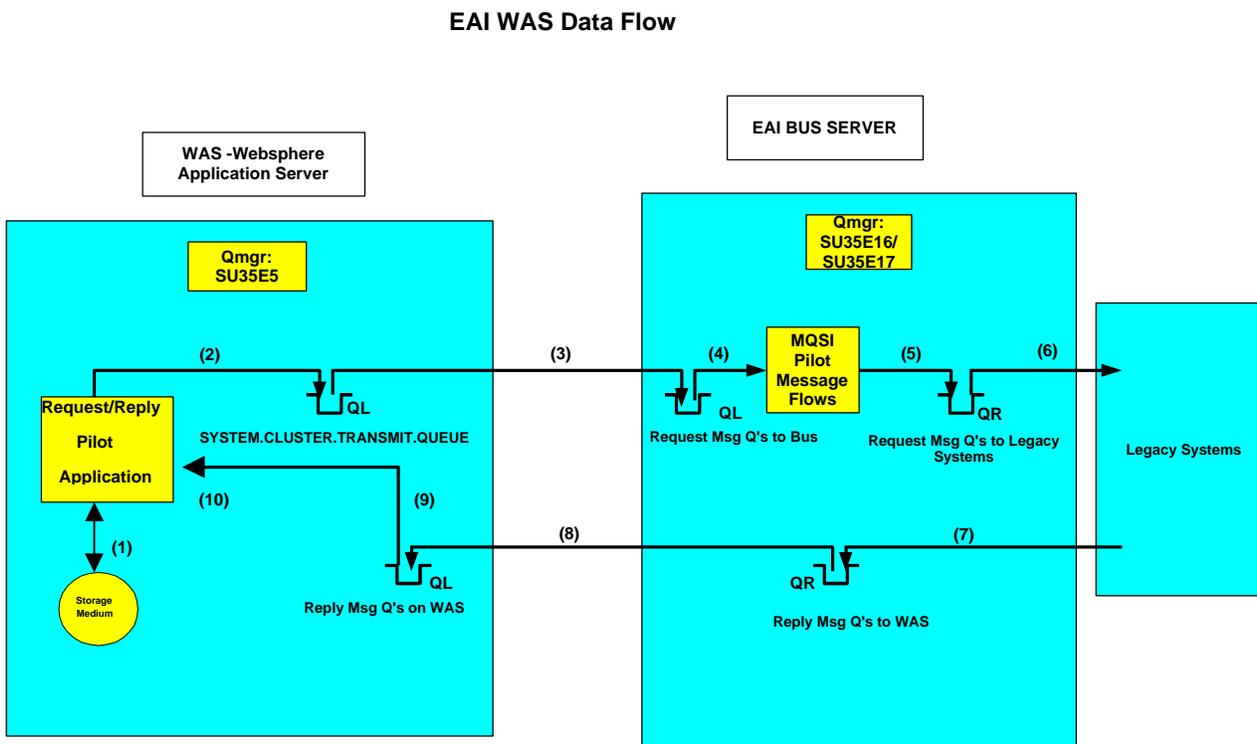
There are sample MQSeries programs that can be used as templates. The samples are provided with MQ Series Messaging and have been installed on the bTrade server. The sample programs can be found in /home/mqm/btrade/.

3.4 WEBSHERE APPLICATION SERVER

The Integrated Technical Architecture (ITA) team on the Modernization project supports installation and configuration of the WebSphere application Server (WAS). MQSeries messaging has been installed on a development WebSphere Application Server to support the validation of the Release 1 EAI Core architecture to send messages from the ITA Internet domain, through the EAI Bus to each targeted Release 1 legacy system. The results of each request are sent back to the source server for display/write to a file. For each of the Release 1 legacy systems a simple inquiry function will be implemented to send a message to each target legacy system, build the required message transformation, and route to the target legacy system. The EAI Core Architecture team worked along side the ITA team to provide an adequate test and development environment to design, develop and implement an EAI Release 1 test application to validate each of the sample EAI functionality for each Release 1 legacy system.

3.4.1 EAI WAS Request / Reply Data Flow

The figure below describes the message flow through the WebSphere application server.



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

- 1) A message is read from some storage medium from a WAS Request/Reply test application.
- 2) The test application creates a Request type message and puts it to a Cluster Queue from the WAS box.

- 3) The MQSeries Queue Manager (SU35E5) on the WAS moves the Request message to the Local Queue on an EAI Bus server.
- 4) The Request message is pulled from the queue and processed through a MQSI Message Flow.
- 5) The output Request message from the MQSI Message Flow is put to a Remote Queue.
- 6) The MQSeries Queue Manager (SU35E16/SU35E17) on the EAI Bus server moves the message to a Local Queue on a Legacy System.
- 7) After the Legacy System MQSeries Adapter(s) process the Request message, a Reply message is put to a Remote Queue on the EAI Bus Server.
- 8) The MQSeries Queue Manager (SU35E16/SU35E17) on the EAI Bus server moves the Reply message(s) to a Local Queue on the WAS.
- 9) The test application on the WAS, pulls the Reply message from the Local Queue.
- 10) The test application will write the response out to some storage medium.

3.4.2 MQ-WebSphere Adapter

In order to interact with MQSeries from a WebSphere Application Server application it is necessary to create a mechanism by which the two may interact. WAS has no implicit knowledge of MQSeries so applications must invoke manually whatever function is needed. For the CORE release a single class was created which has two major functions. First, it accepts parameters from application programs and formats XML requests specific to the target system. For example, a CPS request to inquire on application status requires that the user input an SSN and a Name ID. These items are formatted into an XML document that is ready to then be placed into MQSeries. Secondly, the class establishes a connection with MQSeries and places that request into the proper queue. It then waits a fixed period of time for a reply, sending that reply in string format back to the calling application that then formats it for display. This layer of isolation removes the need for the application programs to have any knowledge of MQSeries.

For each sample application validated for Release 1 the input/output data for/from each legacy system is stored on a directory on the WAS server. Each legacy system contains its own high-level directory structure with 2 subdirectories, input and output. The input directory contains the input data for sending a message to the target system, while the output directory contains the resulting response data from each legacy system.

See the Appendix A for the name of the WAS MQ Adapter. The source code for the adapter is stored in the ClearCase repository.

3.5 EAI CORE ARCHITECTURE RELEASE 1 SOFTWARE PRODUCTS

The following products have been utilized in the design, build and test of the Release 1 EAI Core Architecture:

- DB2 database v6.1 has been installed on the EAI Bus Sun Servers and the MQSI NT Servers. This product is included with the full installation of MQSI and does not require an additional license as long as it is only used for MQSI.
- WebSphere Application Server v3.5 utilized on the WAS Sun boxes in development and test.

- MQSeries v5.2 has been installed on all systems except for DLSS, which has MQSeries v2.2.1.1 installed.
- MQSeries Integrator v2.0.1 has been installed on the EAI Bus Sun Servers, the MQSI NT Servers and the MQSI development workstations.

The versions of these products were chosen based on the analysis of the five Legacy Systems used for the EAI Core Architecture Release 1. It had been determined that each Legacy System had the required prerequisites necessary to install the software versions listed above. These software versions were the most currently supported software products from IBM at the time of the analysis.

4 EXECUTION ARCHITECTURE COMPONENTS

The purpose of this section is to define the execution components for the EAI Core architecture. The EAI Core architecture was designed and built to validate the EAI core infrastructure design, development and validation of sample functionality for each Release 1 legacy system there are no specific SFA EAI application execution components defined. As SFA develops EAI applications the application development teams will utilize the services and components developed to define and implement the EAI Core architecture services and interfaces. The developed MQ Adapters for each system will serve as reusable components that can be utilized as is or modified to support each application's EAI requirements.

The following sections defines in detail the execution components developed and provided by the EAI Core architecture to serve as a foundation for designing and developing applications to use the SFA EAI infrastructure.

4.1 MQSERIES MESSAGING CAPABILITIES

This section describes the standard MQSeries Messaging architecture components utilized for Release 1 of the Enterprise Application Integration Core Architecture. These components are being used as services provided by MQSeries Messaging.

4.1.1 Application Programs and Messaging

The MQSeries products enable programs to communicate with each other across a network of unlike components, such as processors, subsystems, operating systems, languages and communication protocols.

MQSeries programs use a consistent application program interface (API) across all platforms, enabling application programs to communicate with each other using messages and queues. This form of communication is referred to as asynchronous messaging. As implemented by MQSeries, it provides assured, once-only delivery of messages. Using MQSeries means that application developers can decouple application programs, so that the program sending a message can continue processing without having to wait for a reply from the receiver. If the receiver, or the communication channel to it, is temporarily unavailable, the message can be forwarded at a later time. MQSeries also provides mechanisms for generating acknowledgments of messages received.

The programs that utilize MQSeries software can be running on different computers, on different operating systems, and at different locations. The applications are written using a common programming interface known as the Message Queue Interface (MQI), so that applications developed on one platform can be transferred to another.

MQSeries messaging products make it straightforward for applications to exchange information between 35 platforms. The MQSeries products take care of network interfaces, assure delivery of messages, deal with communications protocols, and handle recovery after system problems. Message queues in a cluster can automatically configure and define their resources with one another, balance work amongst themselves, and provide 'hot standby' in case of failure.

Programs communicate using the MQSeries API, an easy-to-use, high-level program interface that shields programmers from the complexities of different operating systems and underlying networks. Developers can focus on the business logic, while MQSeries manages the connections to the computer systems.

4.1.2 Queue Managers

In MQSeries, queue objects are managed by a component called a queue manager. The queue manager provides messaging services for the applications and processes. The queue manager ensures that messages are put on the correct queue or that the messages are routed to another queue manager.

Before applications can send any messages, a queue manager must be created along with queue objects. MQSeries for Windows NT provides the MQSeries Explorer GUI utility to help create queue managers and define other MQSeries objects needed for applications.

4.1.3 Connecting an Application to a Queue Manager

Any MQSeries application must make a successful connection to a queue manager before it can make any other MQI calls. When an application successfully makes the connection, the queue manager returns a connection handle. This is an identifier that the application must specify each time it issues an MQI call. An application can connect to only one queue manager at a time (known as its local queue manager), so only one connection handle is valid (for that particular application) at a time. When the application has connected to a queue manager, all the MQI calls it issues are processed by that queue manager until it issues another MQI call to disconnect from that queue manager.

4.1.4 Opening a Queue

Before an application can use a queue for messaging, it must open the queue. If putting a message on a queue, the application must open the queue for output. Similarly, if getting a message from a queue, the application must open the queue for input or browse. An application can specify that a queue be opened for both getting and putting, if required. The queue manager returns an object handle if the open request is successful. The application specifies this handle, together with the connection handle, when it issues a put or a get call. This ensures that the request is carried out on the correct queue.

4.1.5 Putting and Getting Messages

When the open request is confirmed, an application can put a message on the queue. To do this, it uses another MQI call on which a number of parameters and data structures are specified. These define all the information about the message being put, including but not limited to the parameters of a message type, format, persistence, priority, correlation id, message id, reply to queue and reply to queue manager. The message data (that is, the application-specific contents of the message the application is sending) is defined in a buffer, which is specified in the MQI call. When the queue manager processes the call, it adds a message descriptor, which contains information that is needed to ensure the message can be delivered properly. The message descriptor is in a format defined by MQSeries; the message data is defined by the application (this is what is put into the message data buffer in the application code).

The program that gets the messages from the queue must first open the queue for getting messages. It must then issue another MQI call to get the message from the queue. On this call, there is an option to specify which message to get or to retrieve in FIFO or an order of priority.

4.1.6 Transactional Integrity

MQSeries supports transactional messaging, which means that operations on messages can be grouped into 'Units Of Work' (UOW). A unit of work is either committed in its entirety, or backed-out, so that it's as if none of the operations took place. This means that data is always in a consistent state. Transactional messaging is done in each adapter by using the MQSeries function call MQCMIT and MQBACK.

The MQCMIT call indicates to the queue manager that the application has reached a syncpoint, and all of the message gets and puts that have occurred since the last syncpoint are to be made permanent. Messages put as part of a unit of work are made available to other applications; messages retrieved as part of a unit of work are deleted.

The MQBACK call indicates to the queue manager all of the message gets and puts that have occurred since the last syncpoint are to be backed out. Messages put as part of a unit of work are deleted; messages retrieved as part of a unit of work are reinstated on the queue.

4.1.7 Security

Security is an important aspect for a distributed system and MQSeries provides a flexible security framework that allows the appropriate security architecture to be implemented to meet SFA's enterprise requirements. There are several aspects to the MQSeries security model. A detailed security model and its implementation are outside the scope of the Release 1 EAI Core architecture. Security should be addressed on an enterprise level. At a minimum, each SFA application that wants to connect to the EAI Bus should perform a detailed security analysis of their message data and interfaces during the design stage to ensure an adequate implementation and prevention of unauthorized access.

4.1.8 Triggering

The asynchronous nature of message queuing may mean that applications are idle for periods of time when there are no messages to process. To avoid having idle processes consume system resources while there is no work to do, MQSeries provides a mechanism to 'trigger' applications to start when certain conditions are met. Triggering works by defining one or more specific conditions for an application's queue, which, when met, will cause the queue manager to send a trigger message to a MQSeries queue called an initiation queue. The trigger message is processed by a special application called a trigger monitor, which reads the trigger message from the initiation queue and uses the information in the message to help decide which application to start to process the messages on the application's queue. By using a trigger monitor there can be a single process that initiates many application processes to handle messages arriving on many different queues, as required.

4.2 MQSERIES INTEGRATOR CAPABILITIES

This section describes the standard MQSeries Integrator architecture components utilized for the Release 1 Enterprise Application Integration Core Architecture. These components are being used as services provided by MQSeries Integrator. The MQSI product includes multiple IBM Primitive node types that can be used to build a message flow. MQSI provides the capabilities to implement business logic, data transformation and message routing in a centralized broker repository. The SFA EAI Bus includes redundant MQSI servers that process all MQ messages and perform transformation and routing as defined in each message flow. Message flows can be specific to an application or can be shared among applications based on the content of the message data.

MQSeries Integrator is a powerful message broker that reacts to business events, using MQSeries to deliver messages. It simplifies the complexity of connections between applications by establishing a bus through which messages pass and where operations on messages can be performed, and enterprise-defined rules can be intelligently applied. Operations on messages include, but are not limited to filtering, transformation, routing and data extraction from databases. Reformatting invoked dynamically by the run-time engine, supports parsing and reformatting messages. Message formats can be self-defining using XML, defined using an IBM-provided format repository, or, if desired, by using third-party message dictionaries.

MQSeries Integrator (MQSI) is designed on the premise that, instead of direct connection between systems, a new connection point is established in the middle of the “map” -- a bus -- and each system has a single connection to the bus. The maze of connections dramatically simplifies to a star diagram, and the bus at the center of the star becomes the focus for enterprise intelligence. This “hub” at SFA is considered to be the EAI Bus servers, also often referred to as a “message broker”. All messages pass through the EAI Bus servers, which are architected to handle large volumes of message traffic. This is possible because the EAI Bus servers are not a single OS process but a collection of processes on multiple machines, operating together under the concept of a *broker domain*, each broker of which consists of one or more multithreaded OS processes acting as integration engines. This multi-broker, multi-process, multi-threaded architecture enables the EAI Bus servers to distribute work, tolerate a failing broker, restart failing components automatically, facilitate planned outages, and scale to the needs of the enterprise, either vertically or horizontally.

The EAI Bus servers hold the core of the enterprise's intelligence, in that the servers serve as the repository of information that is used to orchestrate the integration of pillar applications on different platforms. The knowledge the servers can maintain are of two types: knowledge of the business, including information and rules by which the business is run, and knowledge of the applications in the enterprise systems. This knowledge allows the EAI Bus servers to take action, based on the type of messages that come in.

Message broker(s) that comprise the EAI Bus servers are associated with MQSeries queue managers, which are responsible for overseeing message traffic going into and out of the message broker itself. The queue managers are responsible for message integrity and transactional control, including message logging, rollback and/or redirection in the event of a failure. Because the EAI Bus servers have been configured for MQSeries Clustering the scalability and load-balancing capabilities inherent to clustering can be taken advantage of.

Knowledge of the business and applications enables the core functions of transformation and routing. Transformation between interfaces is nothing new; most enterprises have applications that have developed over the years, on different systems, using different programming languages and different methods of communication, that have interfaces to one another. Standard message queuing technology can bridge differences like these, but message queues (or the applications that use them) have to be explicitly told about the location and characteristics of message destinations. MQSeries Integrator changes all that. Using MQSI this knowledge about each application, and its relationship to other applications, is stored just once in the bus, which uses intelligent routing to decide where each message is to go, and in what format it is to go there. For instance, personal names are held in many forms in different applications. Surname first or last, with or without middle initials, upper or lower case: these are just some of the permutations. Another example is data augmentation, or enrichment; perhaps an input message contains an account number, but not a name. Using this, business logic in the bus can issue an SQL query against a customer database using the account number, obtain the customer's name, and then forward that to other applications in the appropriate format.

These examples illustrate how the EAI Bus servers, supplied with the information definition of each application, can supply data in the right format to any receiving application, without the sending application needing to be modified in any way. Knowledge of business rules and information requirements enables intelligent routing of information to where it's needed. Intelligent routing is another capability of the EAI Bus Broker servers. Intelligent routing encapsulates business knowledge of how information should be distributed between message sending and receiving applications throughout the

enterprise. This knowledge is stored in the EAI Bus Servers as a set of rules that are applied to each message that passes through the EAI Bus servers. For example, all loan applications over a certain amount may be routed to a person who must authorize them. Messages can be sent on or distributed, according to criteria applied to the content of fields within the message data or attributes of the message contained in the one or more headers that may be added to the front of the messages.

The advantage of this capability is that a far more flexible approach can be taken to the distribution of information. It is here that the organization can really see the application of enterprise intelligence in the information systems -- a business manager with a concept for an enhancement to an automated process has only to articulate it in terms of a few business rules. Then the rules can simply be stored in the EAI Bus Servers, rather than having to laboriously modify the appropriate applications, or maintain that information at many points in the EAI infrastructure.

4.3 EXPORTING MESSAGE FLOWS BETWEEN DEVELOPMENT WORKSTATIONS AND MQSI BUILD-TIME SERVER

4.3.1 Exporting Message Flows

Exporting message flows and message sets is a two-part operation.

1) From the MQSI control center, choose File->Export and chose a file name and path to export your workspace. This will contain message flows and topology

2) From the command line, enter:

```
mqsimrmimpexp -e MQSIMRDB mqsiuid mqsipw <Message Set Name> <msgsetname>.mrp
```

- MQSIMRDB is the MRM ODBC database alias;
- *mqsiuid* is the userid for that DB;
- *mqsipwd* is the password for that DB;
- *<Message Set Name>* is the message set name;
- *<msgsetname>.mrp* is the name of the export file to be created. This file will contain the named message set.

4.3.2 Importing Message Flows

Importing message flows and message sets is a four-part operation.

1) Stop the configuration manager by issuing the following command from a command prompt:
mqsisstop configmgr

2) From the command prompt, enter:

```
mqsimrmimpexp -i MQSIMRDB mqsiuid mqsipw <msgsetname>.mrp
```

- MQSIMRDB is the MRM ODBC database alias;
- *mqsiuid* is the userid for that DB;
- *mqsipwd* is the password for that DB;
- *<msgsetname>.mrp* is the name of the message set file to be imported

- 3) Restart the configuration manager by issuing the following from a command prompt: `mqsistart configmgr`
- 4) From the MQSI control center, chose File->Import and select the workspace file which you exported previously. When prompted, select only message flows to be imported.

Restart the control center and the flows and message sets will be present.

4.4 DEPLOYING MQSI CONFIGURATION DATA FROM THE BUILD-TIME SERVER TO THE RUN-TIME SERVER

Migration of message flows and message sets to the brokers is called deployment. When requesting a deployment of any type of configuration data, the Configuration Manager copies the relevant configuration data from the shared configuration and transmits it to the relevant brokers. When the deployment is successful, the brokers are able to act in accordance with the newly deployed data.

The following types of configuration data need to be deployed before being used in the broker domain:

- Assignments data: Execution groups to brokers; message flows to execution groups; and message sets to brokers.
- Topology data: Broker and collective data for the broker domain. Collectives are related to publish and subscribe and are not applicable for SFA at this time.

4.4.1 Three types of deployment

Each type of configuration data may be deployed separately or all data may be deployed at once. For each of these types of configuration data, the following can be requested:

- 1) Complete deployment
- 2) Delta deployment
- 3) Forced deployment (This type of deployment is only valid when all configuration data of all types is being deployed)

4.4.1.1 Complete deployment

A complete deployment:

- 1) Deletes all configuration data of that type that is currently deployed on the target brokers
- 2) Creates new configuration data from the shared configuration. For example, if requesting a complete deployment of topics data, the Configuration Manager deploys instructions to all brokers to delete *all* currently deployed topics data and create a new set of topics data from those in the shared configuration.

This type of deployment can take extended periods of time depending on the topology and number of resources in the broker domain and delta deployment may be used most of the time.

4.4.1.2 Delta deployment

When requesting a delta deployment, the Configuration Manager compares the configuration data of that type that is currently deployed on the target brokers with the shared configuration, and deploys only the differences between the two versions. Therefore, the delta deployment is better for performance, especially when there is a large amount of configuration data in the shared configuration.

4.4.1.3 Forced deployment

The forced deployment, which overrides any outstanding deployment request, is used typically to correct error situations. Therefore, to maintain consistency of the configuration data throughout the broker domain, a forced deployment is allowed only when deploying all types of configuration data. A forced deployment is always a complete deployment.

4.4.2 Stages of Deployment

Deployment of configuration data occurs in two stages.

4.4.2.1 Stage One of Deployment

During stage one of deployment, which is synchronous, the Configuration Manager sends a configuration data stream to the `SYSTEM.BROKER.ADMIN.QUEUE` of each target broker. When the configuration data has been sent to all relevant brokers, control is returned to the user. If the first stage is successful, message `BIP1520I` is displayed identifying the brokers to whom the data was deployed.

However, if an error is detected during the first stage of deployment, the deployment is abandoned: no configuration data is sent to any broker, and an appropriate error message is displayed in a Control Center dialog box.

4.4.2.2 Stage Two of Deployment

During stage two of the deployment process, which is asynchronous, the target brokers process the received configuration data and return a response on the Configuration Manager's `SYSTEM.BROKER.ADMIN.REPLY` queue. The Configuration Manager then updates its record of the deployed configuration. Deployment of data to a target broker might be only partially successful. This is because the unit of deployment on a broker is the execution group: the deployment of one execution group to a broker might succeed, but the deployment of another to the same broker might fail. A unit of deployment is transactional, however, so either all changes are made to a given execution group or no change is made.

For deployment purposes, topology data is considered to belong to a separate unit of deployment, so either all changes are made to topology, or no change is made.

4.4.3 Deploying and Checking Data In and Out

When a deployment of any type of configuration data takes place, the data of that type that has been checked into the shared configuration by all Control Center users in the broker domain is that which is deployed to the configuration repository. Data that has not been checked in is not deployed. Note also that descriptive text that is supplied when defining Control Center resources is not deployed.

If the fact that some data has not been checked in leaves the shared configuration in an inconsistent state, the deployment is likely to fail. If the Configuration Manager detects an inconsistency, a message is received indicating that some Control Center resources are not checked in.

To avoid this situation occurring, request a list of all resources in the workspace that have not been checked in (using the `File` → `Check In List` action) before deploying. The user can also check in all checked-out configuration data in your workspace using the `File` → `Save to Shared` action. Of course, if multiple users are creating shared configuration data, that activity must cease while a deployment takes place, and all users must check in any checked-out resources before the deployment is requested.

4.4.4 Verifying Successful Deployment

To determine whether stage two of a deployment has succeeded refresh the Log view: click the green refresh button on the taskbar, or select View → Refresh. It might take a while for the response to arrive. The refreshed Log view displays a group of messages for each broker to which configuration data has been deployed. Typical messages are:

<u>Message</u>	<u>Meaning</u>
BIP2056	Indicates that a deployment was completely successful for the broker.
BIP2086	Indicates that a deployment was partially successful for the broker.
BIP2087	Indicates that a deployment was completely unsuccessful for the broker.

If a deployment fails completely or partially succeeds, and message BIP4046 also appears in the Log view the broker in question is out of step with the rest of the broker domain. The user *must* correct the problem that caused the failure and deploy again to restore consistency of data throughout the broker domain.

For deployment purposes, data is considered to belong to a separate unit of deployment, so either all changes are made to the topology, or no change is made.

4.4.5 SFA EAI Release 1 Core Specific Deployment Details

The EAI bus currently includes the MQSI brokers in the NT SFANT006 server and the Sun SU35E16 and SU35E17 servers. MQSeries channels are defined between the configuration manager and the brokers to enable resource deployment.

4.4.5.1 Deployment Cookbook

Step 1: Define MQSeries resources

Create all required queues on each broker queue manager

Step 2: Define any database resources required of the flow

Create any required databases, tables or data required by the flow. Ensure these databases are ODBC enabled.

Step 3: Assign Resources

In order to deploy new message flows, all that need be done is to assign them to execution groups and assign applicable message sets to their respective brokers. This is done using the MQSI Control Center. Selecting the assignments tab will display a three-paned interface where these may be assigned. In order to assign message flows, the execution group must be checked out. In order to assign message sets, the broker must be checked out. To assign a message flow, simply click and drag it to the target execution group. To assign a message set, simply click and drag it to the target broker. When complete be sure to check in all brokers and execution groups. Ensure that corresponding message sets are assigned at the same time as message flows which require them. MQSI will neither reject an assignment nor a deployment of a message flow for which a required message set has not been assigned, instead a runtime error will be generated.

Step 4: Initiate Deployment

Once resources have been assigned, begin deployment by selecting **File->Deploy->(type)** from the MQSI Control Center, where type is the type of deployment required. To deploy assignments data, the user must be a member of group mqbrops.

Step 5: Verify Deployment

Once deployment is initiated via the MQSI Control Center, select the Log tab. Continue to refresh the log display by clicking the green circular arrow icon at the top of the panel. Normally the log will display entries indicating the success or failure of the deployment, however it is sometimes necessary to view other sources of information. On the configuration manager and NT broker the Windows NT event log contains entries which can also provide an indication as to success or failure. On the Sun brokers, the /var/adm/messages file is where MQSI directs stderr.

It is possible for the deployment of an execution group to time out while the target broker is processing it. This effectively leaves the status of the execution group in doubt. This status is shown in the Operations view by the appearance of a yellow question mark over the traffic light status icon. A message in the Log view confirms the problem. The in-doubt status of the execution group can be resolved only by a subsequent deployment of all assignments data. (Note that a subsequent delta deployment is automatically converted to a complete deployment if any execution group is in the “in-doubt” state).

A simple way to verify the successful deployment of a message flow is to view the properties of the MQSeries queue that is named on the MQInput node. The IPPROCS parameter will indicate the number of processes, which have the queue open for input. If this is greater than one the flow is most likely active.

Step 6: Test the newly deployed flow

Even if a deployment appears to be successful, errors may occur at runtime. Test, and if necessary, correct the message flow and re-deploy following the same steps outlined above. The actual testing of a deployed message flow occurs during the system integration and acceptance testing in the pre-production or staging environment. The staging environment replicates the behavior and functionality of the production system to validate the integration of the EAI components, i.e. message flows, are interoperable and compatible with the existing production applications.

4.5 SFA EAI ARCHITECTURE CUSTOMIZATION

This section describes the SFA Enterprise Application Integration (EAI) Bus Architecture customization required. The installation and configuration customization and other components that are customized for SFA are described.

4.5.1 MQSeries Implementation and Configurations for the Non-Legacy Integration Release 1

This section describes installation and configuration customization required for the Non-Legacy systems at SFA. Each system described will have unique characteristics to it. The sections have been organized for each system in such a way that makes this section easier to follow. Systems with common characteristics may be combined together to reduce complexity and redundancy within the documentation.

The queue managers that have been created on the EAI Bus servers have been defined as part of a MQSeries cluster. These two queue managers are both full repository queue managers. This means that each queue manager will hold all information required when adding queues to the cluster and joining new queue managers to the cluster. This way there is one queue manager as a back up cluster repository. The cluster name that is defined for this cluster is “EAI”. In order to prepare the EAI Bus servers to define

the cluster, MQSeries for Sun Solaris version 5.2 has been installed and configured. The table below describes the MQSeries objects and names of the objects created for the EAI Bus servers. The MQSeries objects that have been created were the minimum required objects to support the EAI Core Architecture Release 1 sample application testing.

Each of the queue managers has been altered to use their specific Dead Letter Queue. Each of the queue managers has channel definitions defined for them to the Legacy Systems defined as part of the Release 1 EAI Core Architecture. A listener process has been started in background for each receiving channel for a queue manager using the supplied listener from MQSeries. These have been started in background to help prevent someone from killing the process unnecessarily. In order for the MQSeries listener to survive a system restart and terminate properly upon system shutdown, the MQSeries control command to start the listener must be placed in the system startup and shutdown scripts.

The systems described are as follows:

- 1) WebSphere Application Server
- 2) EAI Bus Servers
- 3) MQSI Configuration Manager NT Server

4.5.1.1 MQSeries for WebSphere on Solaris

The table below contains the MQSeries objects defined on the WebSphere Application Server

MQSeries Queue Manager(s) and Objects

MQ ObjectName	Description	Object Type
SU35E5	Queue Manager for Test Environment	Queue Manager
SU35E5.DEAD.LETTER.QUEUE	Dead Letter Queue	Local Queue
TO.SU35E16	Sender Channel	Cluster Sender Channel
TO.SU35E5	Receiver Channel	Cluster Receiver Channel
WAS.FROM.EAI.REQPELL	Reply Queue for NSLDS pilot	Local Queue
WAS.FROM.EAI.REQAPPSTATUS	Reply Queue for CPS pilot	Local Queue
WAS.FROM.EAI.RESPONSELOAN	Reply Queue for DLSS pilot	Local Queue
WAS.FROM.EAI.REPLY SCHOOL	Reply Queue for PEPS pilot	Local Queue
WAS.FROM.EAI.REPLYMAIL	Reply Queue for bTrade pilot	Local Queue

4.5.1.2 EAI Bus Client / Server Configuration and Design

The table below contains the MQSeries objects defined on the EAI Bus.

MQSeries Queue Manager(s) and objects

MQ ObjectName	Description	Object Type
EAI	Cluster Name	Cluster
SU35E16	Queue Manager for Test Environment	Queue Manager
SU35E16.DEAD.LETTER.QUEUE	Dead Letter Queue	Local Queue
TO.SU35E5	Sender Channel	Cluster Sender Channel
TO.SU35E16	Receiver Channel	Cluster Receiver Channel
TO.SU35E17	Sender Channel	Cluster Sender Channel
SU35E5	Transmission Queue for Test Environment	Transmission Queue
NTT1	Transmission Queue for Test Environment	Transmission Queue
NPT1	Transmission Queue for Test Environment	Transmission Queue
CPT1	Transmission Queue for Test Environment	Transmission Queue
CRDEV2	Transmission Queue for Test Environment	Transmission Queue
HPDEV1	Transmission Queue for Test Environment	Transmission Queue
PEPSK570	Transmission Queue for Test Environment	Transmission Queue
EAI.FROM.WAS.REQPELL	Input Queue for NSLDS pilot	Local Queue
EAI.TO.NSLDS.REQPELL	Output Queue for NSLDS pilot	Remote Queue
WAS.FROM.EAI.REPLYPELL	Reply Queue for NSLDS pilot	Remote Queue
EAI.FROM.WAS.REQAPPSTATUS	Input Queue for CPS pilot	Local Queue
EAI.TO.CPS.REQAPPSTATUS	Output Queue for CPS pilot	Remote Queue
WAS.FROM.EAI.REPLYAPPSTATUS	Reply Queue for CPS pilot	Remote Queue
EAI.FROM.WAS.LOAN	Input Queue for DLSS pilot	Local Queue

MQ ObjectName	Description	Object Type
EAI.TO.DLSS.LOAN	Output Queue for DLSS pilot	Remote Queue
WAS.FROM.EAI.RESPONSELOAN	Reply Queue for DLSS pilot	Remote Queue
EAI.FROM.WAS.GETSCHOOL	Input Queue for PEPS pilot	Local Queue
EAI.TO.PEPS.GETSCHOOL	Output Queue for PEPS pilot	Remote Queue
WAS.FROM.EAI.REPLY SCHOOL	Reply Queue for PEPS pilot	Remote Queue
EAI.FROM.WAS.GETMAIL	Input Queue for bTrade pilot	Local Queue
EAI.TO.BTRADE.GETMAIL	Output Queue for bTrade pilot	Remote Queue
WAS.FROM.EAI.REPLYMAIL	Reply Queue for bTrade pilot	Remote Queue
SU35E17	Queue Manager for Test Environment	Queue Manager
SU35E17.DEAD.LETTER.QUEUE	Dead Letter Queue	Local Queue
TO.SU35E5	Sender Channel	Cluster Sender Channel
TO.SU35E17	Receiver Channel	Cluster Receiver Channel
TO.SU35E16	Sender Channel	Cluster Sender Channel
SU35E5	Transmission Queue for Test Environment	Transmission Queue
NTT1	Transmission Queue for Test Environment	Transmission Queue
NPT1	Transmission Queue for Test Environment	Transmission Queue
CPT1	Transmission Queue for Test Environment	Transmission Queue
CRDEV2	Transmission Queue for Test Environment	Transmission Queue
HPDEV1	Transmission Queue for Test Environment	Transmission Queue
PEPSK570	Transmission Queue for Test Environment	Transmission Queue
EAI.FROM.WAS.REQPELL	Input Queue for NSLDS pilot	Local Queue
EAI.TO.NSLDS.REQPELL	Output Queue for NSLDS pilot	Remote Queue
WAS.FROM.EAI.REPLYPELL	Reply Queue for NSLDS pilot	Remote Queue
EAI.FROM.WAS.REQAPPSTATUS	Input Queue for CPS pilot	Local Queue

MQ ObjectName	Description	Object Type
EAI.TO.CPS.REQAPPSTATUS	Output Queue for CPS pilot	Remote Queue
WAS.FROM.EAI.REPLYAPPSTATUS	Reply Queue for CPS pilot	Remote Queue
EAI.FROM.WAS.LOAN	Input Queue for DLSS pilot	Local Queue
EAI.TO.DLSS.LOAN	Output Queue for DLSS pilot	Remote Queue
WAS.FROM.EAI.RESPONSELOAN	Reply Queue for DLSS pilot	Remote Queue
EAI.FROM.WAS.GETSCHOOL	Input Queue for PEPS pilot	Local Queue
EAI.TO.PEPS.GETSCHOOL	Output Queue for PEPS pilot	Remote Queue
WAS.FROM.EAI.REPLYSCHOOL	Reply Queue for PEPS pilot	Remote Queue
EAI.FROM.WAS.GETMAIL	Input Queue for bTrade pilot	Local Queue
EAI.TO.BTRADE.GETMAIL	Output Queue for bTrade pilot	Remote Queue
WAS.FROM.EAI.REPLYMAIL	Reply Queue for bTrade pilot	Remote Queue

4.5.2 MQSeries Implementation and Configurations for the Legacy Integration Release 1

This section describes installation and configuration customization required for the Legacy systems at SFA. Each system described will have unique characteristics to it. This document has tried to organize the sections in such a way that systems with common characteristics are combined together in order to reduce complexity and redundancy within the documentation. The systems described in this section are:

1. CPS and NSLDS
2. DLSS
3. PEPS
4. bTrade

4.5.2.1 MQSeries for CPS and NSLDS on OS/390

The MQSeries objects defined on the CPS and NSLDS systems are contained in the table below.

MQSeries Queue Manager(s) and objects:

MQ ObjectName	Description	Object Type
CST1	Queue Manager for System LPAR (test region)	Queue Manager
CST1.DEAD.QUEUE	Dead Letter Queue	Local Queue
N/A	Sender Channel	Sender Channel
N/A	Receiver Channel	Receiver Channel

MQ ObjectName	Description	Object Type
N/A	Transmission Queue for Test region	Transmission Queue
N/A	Sender Channel	Sender Channel
N/A	Receiver Channel	Receiver Channel
N/A	Transmission Queue for Test region	Transmission Queue
N/A	Reply Queue for pilot	Remote Queue
N/A	Input Queue for DPL Bridge programs	Local Queue
CPT1	Queue Manager for Production LPAR (test region)	Queue Manager
CPT1.DEAD.QUEUE	Dead Letter Queue	Local Queue
CPT1.SU35E16	Sender Channel	Sender Channel
SU35E16.CPT1	Receiver Channel	Receiver Channel
SU35E16	Transmission Queue for Test region	Transmission Queue
CPT1.SU35E17	Sender Channel	Sender Channel
SU35E17.CPT1	Receiver Channel	Receiver Channel
SU35E17	Transmission Queue for Test region	Transmission Queue
CPT1.CICSDEV2.BRIDGE.QUEUE	Input Queue for DPL Bridge programs	Local Queue
CPP1	Queue Manager for Production LPAR (production region)	Queue Manager
CPP1.DEAD.LETTER.QUEUE	Dead Letter Queue	Local Queue
CSP1.SU35E16	Sender Channel	Sender Channel
SU35E16.CSP1	Receiver Channel	Receiver Channel
SU35E16	Transmission Queue for Test region	Transmission Queue
CSP1.SU35E17	Sender Channel	Sender Channel
SU35E17.CSP1	Receiver Channel	Receiver Channel
SU35E17	Transmission Queue for Test region	Transmission Queue
CPP1.CICS.BRIDGE.QUEUE	Input Queue for DPL Bridge programs	Local Queue
NST1	Queue Manager for System LPAR (test region)	Queue Manager
NST1.DEAD.QUEUE	Dead Letter Queue	Local Queue
N/A	Sender Channel	Sender Channel
N/A	Receiver Channel	Receiver Channel
N/A	Transmission Queue for Test region	Transmission Queue
N/A	Sender Channel	Sender Channel

MQ ObjectName	Description	Object Type
N/A	Receiver Channel	Receiver Channel
N/A	Transmission Queue for Test region	Transmission Queue
N/A	Reply Queue for pilot	Remote Queue
N/A	Input Queue for DPL Bridge programs	Local Queue
N/A	Input Queue for Other tests required	Local Queue
NTT1	Queue Manager for Test LPAR (test region)	Queue Manager
NTT1.DEAD.QUEUE	Dead Letter Queue	Local Queue
NTT1.SU35E16	Sender Channel	Sender Channel
SU35E16.NTT1	Receiver Channel	Receiver Channel
SU35E16	Transmission Queue for Test region	Transmission Queue
NTT1.SU35E17	Sender Channel	Sender Channel
SU35E17.NTT1	Receiver Channel	Receiver Channel
SU35E17	Transmission Queue for Test region	Transmission Queue
NSLDS.FROM.EAI.PELL	Input Queue for pilot	Local Queue
NSLDS.PELL.PROCESS	Process for pilot	Process definition
NSLDS.INIT	Initiation Queue for pilot	Local Queue
NPT1	Queue Manager for Production LPAR (test region)	Queue Manager
NPT1.DEAD.QUEUE	Dead Letter Queue	Local Queue
NPT1.SU35E16	Sender Channel	Sender Channel
SU35E16.NPT1	Receiver Channel	Receiver Channel
SU35E16	Transmission Queue for Test region	Transmission Queue
NPT1.SU35E17	Sender Channel	Sender Channel
SU35E17.NPT1	Receiver Channel	Receiver Channel
SU35E17	Transmission Queue for Test region	Transmission Queue
NSLDS.FROM.EAI.PELL	Input Queue for Other tests required	Local Queue
NSLDS.PELL.PROCESS	Process for pilot	Process definition
NSLDS.INIT	Initiation Queue for pilot	Local Queue
NPP1	Queue Manager for Production LPAR (production region)	Queue Manager
NPP1.DEAD.QUEUE	Dead Letter Queue	Local Queue
NPP1.SU35E16	Sender Channel	Sender Channel
SU35E16.NPP1	Receiver Channel	Receiver Channel
SU35E16	Transmission Queue for Test region	Transmission Queue

MQ ObjectName	Description	Object Type
NPP1.SU35E17	Sender Channel	Sender Channel
SU35E17.NPP1	Receiver Channel	Receiver Channel
SU35E17	Transmission Queue for Test region	Transmission Queue
NSLDS.FROM.EAI.PELL	Input Queue for Other tests required	Local Queue
NSLDS.PELL.PROCESS	Process for pilot	Process definition
CPS.BATCH.INIT	Initiation Queue for pilot	Local Queue
CPS.TEST.BATCH.QUEUE	Input Queue for pilot	Local Queue
CPS.BATCH.PROCESS	Process for Pilot	Process
NSLDS.BATCH.INIT	Initiation Queue for pilot	Local Queue
NSLDS.FROM.EAI.PELL	Input queue for Pilot	Local Queue
NSLDS.INIT	Initiation Queue for pilot	Local Queue

4.5.2.2 MQSeries for DLSS on Open VMS

The MQSeries objects defined on the DLSS system are contained in the table below.

MQSeries Queue Manager(s) and objects:

MQ ObjectName	Description	Object Type
CRDEV2	Queue Manager for Test Environment	Queue Manager
CRDEV2.DEAD.LETTER.QUEUE	Dead Letter Queue	Local Queue
CRDEV2.SU35E16	Sender Channel	Sender Channel
SU35E16.CRDEV2	Receiver Channel	Receiver Channel
SU35E16	Transmission Queue for Test Environment	Transmission Queue
CRDEV2.SU35E17	Sender Channel	Sender Channel
SU35E17.CRDEV2	Receiver Channel	Receiver Channel
SU35E17	Transmission Queue for Test Environment	Transmission Queue
DLSS.LOAN.PROCESS	Process definition for pilot	Process
DLSS.INIT	Initiation Queue for pilot	Local Queue
DLSS.FROM.EAI.LOAN	Input Queue for pilot	Local Queue

4.5.2.3 MQSeries for PEPS on HP-UX

The MQSeries objects defined on the PEPS system are contained in the table below.

MQSeries Queue Manager(s) and objects:

MQ ObjectName	Description	Object Type
----------------------	--------------------	--------------------

MQ ObjectName	Description	Object Type
PEPSK570	Queue Manager for Test Environment	Queue Manager
PEPSK570.DEAD.LETTER.QUEUE	Dead Letter Queue	Local Queue
PEPSK570.SU35E16	Sender Channel	Sender Channel
SU35E16. PEPK570	Receiver Channel	Receiver Channel
SU35E16	Transmission Queue for Test Environment	Transmission Queue
PEPSK570.SU35E17	Sender Channel	Sender Channel
SU35E17. PEPK570	Receiver Channel	Receiver Channel
SU35E17	Transmission Queue for Test Environment	Transmission Queue
PEPS.SCHOOL.PROCESS	Process definition for pilot	Process
PEPS.INIT	Initiation Queue for pilot	Local Queue
PEPS.FROM.EAI.GETSCHOOL	Input Queue for pilot	Local Queue

4.5.2.4 MQSeries for bTrade on HP-UX

The MQSeries objects defined on the bTrade system are contained in the table below.

MQSeries Queue Manager(s) and objects:

MQ ObjectName	Description	Object Type
HPDEV1	Queue Manager for Test Environment	Queue Manager
HPDEV1.DEAD.LETTER.QUEUE	Dead Letter Queue	Local Queue
HPDEV1.SU35E16	Sender Channel	Sender Channel
SU35E16. HPDEV1	Receiver Channel	Receiver Channel
SU35E16	Transmission Queue for Test Environment	Transmission Queue
HPDEV1.SU35E17	Sender Channel	Sender Channel
SU35E17.HPDEV1	Receiver Channel	Receiver Channel
SU35E17	Transmission Queue for Test Environment	Transmission Queue
BTRADE.GETMAIL.PROCESS	Process definition for pilot	Process
BTRADE.INIT	Initiation Queue for pilot	Local Queue
BTRADE.FROM.EAI.GETMAIL	Input Queue for pilot	Local Queue

4.6 ERROR HANDLING

In MQSeries, errors are handled at the application level, the system level or both. At the application level, a failure such as the inability of an MQPUT request to put a message to a queue, perhaps caused by the queue having no available space, would be indicated to the putting application by returning a specific reason code describing the error. Thus enabling the application to take an appropriate action.

At the system level, instrumentation events can be used to indicate actions whenever the queue manager detects that a predefined condition (or set of conditions) has occurred. In the previous example, an event could be raised when the queue manager detects the “Queue Full” condition on the application queue.

Monitoring software can then be configured to take predefined actions when specific events are detected. Also, the handling of many exception conditions can be automated. For example, undeliverable messages can be automatically directed to an undeliverable message queue. If a failure occurs when processing a message causing the message to be rolled back onto the input queue, MQSeries can be configured to automatically shunt the message to a Dead Letter Queue for exception processing, rather than forcing each application to write additional code to handle this.

With MQSI V2, sophisticated error and exception handling facilities are available. For example, Business Logic is represented to the message broker using a concept called a “message flow”. This concept includes constructs for grouping related actions together and executing them under the control of a “TryCatch” node, which provides a special handler for exception processing. If an exception is subsequently thrown by a downstream node, it is caught by this node, which then routes the original message to its catch terminal, along with an exception list structure describing the nature of the exception. The message flow can then branch off from that point to attempt error recovery, generate an alert, retry the process, etc.

During operation the message broker generates exceptions to handle error conditions. These exceptions are generated and processed by the broker. Errors processed by the broker are written to the standard input/output logs defined during setup and configuration. The provided error handling capabilities of the broker can be expanded through the development of application specific error handling programs, programs written in C, to further expand the existing capabilities of the broker.

4.7 SCALABILITY

Scalability with MQSeries is achieved through several avenues. With MQSeries Integrator, the SFA EAI Bus servers are not a single OS process but a collection of processes on one or more machines, operating together under the concept of a *broker domain*, each broker consists of one or more multithreaded OS processes acting as integration engines.

At the individual broker level, these integration engines support *thread pooling*, with the administrator controlling the number of threads assigned to individual message flows within the engine. Volume increases can be accommodated by increasing the number of threads assigned to the pool. There can also be multiple integration engines (called Execution Groups), each with its own thread pool. This approach can be used to separate, at the OS process level, message flows that have low latency requirements from those that do not require the same level responsiveness.

At the broker domain level, scalability is achieved by enabling more than one broker operating on a SFA EAI Bus Server. For situations where scalability is needed to accommodate increases in message throughput, MQSeries Integrator will operate fully in a MQSeries Clustering environment. The environment created for SFA was configured for MQSeries clustering. This means the enterprise can have any number of individual brokers defined as a cluster, with messages directed to MQSeries queues within the cluster and being serviced by any broker in the cluster, whether on the same machine or on separate machines. As volume load increases additional queue managers and brokers can easily be created and joined to the existing queue managers and brokers to the cluster, without the need to take the cluster offline. Additionally, this approach offers the benefit of load balancing, as MQSeries will automatically distribute messages destined for a specific cluster queue to all instances of that queue within the cluster. This mechanism also provides automatic fail over, as a failing broker will be detected by the MQSeries workload balancing mechanism, which will stop routing messages to that broker's queue's until the broker comes back online and rejoins the cluster.

At the adapter level, scalability is less of a consideration, as any one adapter processes a fraction of the overall message load. However, in the event that the message volume to a single instance of an adapter causes performance degradation within the adapter, it is possible to run additional instances of the adapter, serving the same MQSeries queue and communicating with the same application. Typically, the adapters will match or exceed the processing speed of the application.

All these capabilities together mean this multi-broker, multi-process, multi-threaded architecture will enable distribution of work, tolerate a failing broker, restart failing components automatically, facilitate planned outages, and scale to the needs of the enterprise, either vertically or horizontally.

The SFA EAI Bus provides for redundant MQSI servers, and a MQSeries Messaging cluster implementation. Additional servers can be added to the cluster to support increases in message volume as well as adding additional server(s) to the bus to accommodate increases in message flow processing. The determination of deploying additional servers is based on the transaction volumes as SFA EAI applications are developed and deployed onto the EAI Bus. The EAI Core Architecture provides the foundation to accommodate growth and increases in utilization as the application EAI requirements are defined by the application development teams.

4.8 REDUNDANCY

As part of the SFA Release 1 Core Architecture implementation MQSeries clustering between the WebSphere Application servers and the EAI Bus servers has been implemented, which allows MQSeries to support part of the redundancy needed between these two systems. To fully accomplish the ability to handle a failure of one of the EAI Bus servers, the same MQSI execution components must be installed on each server and would require the use of a shared DB2 database. The existing infrastructure at the VDC does not provide the capability to share a single DB2 database repository for the EAI Bus. Redundancy has been implemented in Release 1 of the EAI Core architecture by maintaining parallel DB2 database copies of each Broker domain on both EAI Run-time servers in the EAI Bus.

4.9 LOAD BALANCING

MQSeries Clustering can be used to achieve load balancing of messages across queue managers, and, by extension, message brokers on the EAI Bus servers. With clustering, the system can have any number of individual queue managers (or brokers) defined in a cluster, with messages directed to MQSeries queues within the cluster. These queues will be serviced by any queue manager (or broker) in the cluster, that could exist on the same EAI Bus server or on separate machines within the same cluster. As volume increases additional queue managers (and brokers) can easily be joined to the cluster, without the need to take the cluster, or any queue managers that are part of it, offline. Load balancing is achieved, by default, by MQSeries automatically distributing message traffic destined for a specific cluster queue to all instances of that queue within the cluster, in a round-robin fashion. Although available, the EAI Core Architecture Release 1 has not implemented any custom cluster workload exit to support any other types of load balancing capabilities.

5 DEVELOPMENT ARCHITECTURE

5.1 OVERVIEW

This section describes the MQSeries Integrator (MQSI) components utilized for the Release 1 Enterprise Application Integration Core Architecture.

5.2 DESCRIPTION

The MQSI Development Architecture for SFA consists of multiple hardware and software components. Within this development environment, there is an MQSI NT development server that houses the Configuration Database and a broker for validating developed message flows. A server used for the MQSI configuration management process and three workstations used for the MQSI Control Centers.

The MQSI runtime environment consists of the MQSI brokers, which are used to host and control the EAI Core pilot message flows. The brokers enable an execution environment to support the component and/or a system testing.

The MQSI configuration manager serves three main functions:

- It maintains configuration details in the configuration repository. This is a set of database tables that provide a central record of the broker domain components.
- It manages the initialization and deployment of brokers and message processing operations in response to actions initiated through the Control Center. It communicates with other components in the broker domain using MQSeries transport services.
- It checks the authority of defined user IDs to initiate those actions.

The MQSI control center has two main functions:

- The creation, manipulation, and deployment of configuration data for a broker domain
- The monitoring and management of the operational state of the same broker domain

For each Release 1 legacy systems, a development environment has been defined to develop the custom adapters. Some development systems are not located within the VDC. The following are the locations of each of the development systems.

- NSLDS: The system has a test LPAR (NSLT) that contains the development TSO and CICS subsystems. This system is located at the VDC.
- CPS: The system has a production LPAR (CPSP) that contains the development TSO and CICS subsystems. This system is located at the VDC.
- DLSS: Their development system (CRDEV2) is located in Rockville.
- bTrade: The development system (HPDEV1) that was used for the core development and component testing is located at bTrade in Dallas. The integration testing was performed on a HP server at the VDC.
- PEPS: The development system (PEPSK570) that was used for the testing is located at the VDC. The system was commonly referred to as the pseudo-PEPS system.
- WAS: The WebSphere development server is a Sun E3500 and is located at the VDC

5.3 OPERATING SYSTEMS

Within the SFA Release 1 EAI Bus Core Development Architecture, there are multiple operating systems involved. Each of the three MQSI software components is spread across different systems to support the development environment for SFA. The MQSI Broker components are installed and configured on two Sun Solaris systems and the NT MQSI build time server. The MQSI Configuration Manager component and a broker are installed and configured on a single Windows NT system. The MQSI Control Centers are installed and configured on three Windows NT workstations.

5.4 DEVELOPMENT PROCESS

There are minimum SFA specific development processes that were used to develop the Release 1 EAI Core Architecture components. Some systems required custom adapters to be built. The systems requiring custom built adapters and the languages used are listed below.

NSLDS: COBOL and Cool:Gen MQ Components
CPS: COBOL
PEPS: Java
BTrade: Java
DLSS: C

The NSLDS online system required the use of the Cool:Gen tool to generate all components involved in accessing the existing NSLDS CICS transactions. The pre-requisites for designing, developing and deploying Cool:Gen components are based on the development and production hardware and system configurations.

5.5 MQSI DEVELOPMENT ENVIRONMENT

This section provides an overview of the development and production environments for MQSI message flow development and testing and operation. These are only recommendations and are not a substitute for a formal planning and sizing exercise in which requirements are accurately determined.

For production use, it is recommended that the components of MQSI V2 be allocated over multiple machines with the following purposes:

- One or more machines to support Control Center usage.
- One machine to support the Configuration Manager. This may also include one Control Center.
- One or more machines to support brokers. By following this organization the brokers can run in a shielded environment as messages are processed. It is important that this processing proceeds without competition for resources from other processes in order to ensure the smooth flow of messages through the enterprise.

A recommended machine specification for the Control Center is a fast uni-processor (Pentium III 500 Mhz processor) with 256MB memory.

A recommended machine specification for the Configuration Manager is a fast uni-processor (Pentium III 500Mhz processor) with 512MB or more of memory.

The specification of the broker machine is more difficult to determine since it requires knowledge of the expected message rate, the types of node that are to be used and the level of transaction control that is used. A recommended minimum specification would be a 2 way processor with 512MB memory. The specification may need to be upgraded if message rates are high or there are many execution groups. In

such cases more detailed planning would be required. To accurately determine resource requirements, prototyping and benchmarking should be considered. The results produced will then be specific and tailored to the individual configuration being built.

A message is termed persistent if it survives when MQSeries restarts. This implies that the message must be logged, or saved, and can be reinstated as part of the recovery procedure. If persistent messages are to be used then it is recommended that solid state disks or disks with a non volatile fast write cache be used for the device on which the MQSeries queue log manager is located. If the message rate is less than 50 msgs/second fast input/output will improve message response time only. If the rate is greater than 50 msgs/second then there will be an improvement in message throughput.

A separate disk is also recommended for the MQSeries queue manager queue data.

If business data is accessed from a relational database the database log and data should each be located on dedicated disks. Consider using a fast device for the database manager log.

In estimating memory requirements for MQSI V2 there are a number of components that need to be considered. These are:

- The Control Center: Most likely, multiple Control centers will be in use.
- The Configuration Manager: There is one Configuration Manager per MQSI V2 implementation.
- The Broker: There may be multiple brokers and within these multiple execution groups and so multiple operating system processes.
- MQSeries Queue Manager: There will be one queue manager per broker.
- Relational Database: A DB2 system is required to hold information on behalf of the Configuration Manager and broker. Additional relational databases may be in use which hold business data.

For the Control Center an initial recommendation is to allow 100MB memory per Control Center. This would be for development use.

The Configuration Manager and its associated DB2 database and queue manager should have a minimum of 256MB of memory available in a development environment, but the recommendation is to have more. The amount of memory required by a broker will depend on the way in which it is configured. A guideline is to allow 300MB for MQSI V2 and its dependent software (broker-related components only, no Configuration Manager or Control Center), with an additional 25 MB per running execution group. This recommendation is based on a MQSeries queue manager configuration consisting of 10 SVRCONN channels and a small number of queue definitions (less than 25). If the number of MQSeries resources (channels, queues etc.) to be configured in a system is different you must make an allowance and amend the amount of memory required accordingly. A SVRCONN channel is a type of MQSeries channel object. For example, there is a server connection channel called SYSTEM.ADMIN.SVRCONN that exists on every remote queue manager. This channel is mandatory for every remote queue manager being administered, without it, remote administration is not possible. When MQSeries is installed on NT, a server connection channel is automatically created and its function is to allow clients to connect to the queue manager.

5.6 MQSI CONFIGURATION CONSIDERATIONS

The following section defines those considerations that should be taken into account when developing MQSI message flows. Consider the following points when building an MQSI V2 configuration:

- It is recommended to use a separate database instance for each of the Configuration Manager and broker.
- It is not recommended to use the database instances for the Configuration Manager or broker to hold business data.
- It is recommended to ensure that the database instance for the Configuration Manager is local to the machine on which the Configuration Manager is installed.
- It is recommended to ensure that the database instance for the broker is local to the machine on which the broker runs.
- It is recommended to use a local database for business data. If the database is remote from the broker machine, ensure that there is a fast, preferably dedicated, communications link between the broker machine and the database manager.
- Carefully examine default settings for nodes and message flows, especially those related to recovery, to ensure that the values are those required. The transaction mode parameter for a MQInput node will default to yes, meaning that the message flow will proceed under transaction control.
- When creating and deploying large message flows increase the heap allocation of the Configuration Manager database. In DB2 this is the APP_CTL_HEAP_SIZE parameter. This value should be increased as determined through performance testing and tuning of the system. DB2 Administrator support will be required as part of database administration.

5.7 DEVELOPMENT TOOLS

The development tools that were used to develop the Release 1 EAI Core Architecture components included the MQSeries Integrator Control Center, the MQSeries commands and system specific compilers for the custom adapters. For the NSLDS system, the Cool:Gen tool was used to build the custom adapters used for the NSLDS CICS transaction environment. Deliverable 54.1.5 EAI Build and Test Report (Release 1) includes the specifics of the custom adapters built and the tools used to build them.

The MQSeries Integrator Control Center was used to build the Release 1 EAI Core Architecture message flows and message sets. The Control Center was installed and configured without any customization.

The MQSeries commands and control commands were used to build the EAI Core Architecture Release 1 MQSeries objects and to start MQSeries specific processes. The command sets are provided as part of the base MQSeries software product.

5.8 CONFIGURATION MANAGEMENT

The configuration management tool in use for the Release 1 EAI Core Architecture is the Rational Clear Case product. All Release 1 EAI software components have been migrated into the specified ClearCase directory for the Release 1 baseline.

Within the MQSeries Integrator software there is a configuration management tool for the MQSI message flows and message set. There is nothing specific to SFA about this tool. In addition, the message flows developed for Release 1 of the EAI Core architecture have been exported and saved in the ClearCase repository.

APPENDIX A: MQSERIES SCRIPTS AND PROGRAMS

The scripts used to define the MQSeries objects on each system reside in the SFA EAI ClearCase Repository. The filenames containing the MQSeries commands are as follows:

bTrade

<i>File name</i>	<i>File description</i>
.profile	Mqm userid .profile
bTrade_MQSeries_Adapter_Arch.vsd	VISIO diagram of MQRbTrade
bTrade_Test_Report.doc	MQRbTrade test report
CbusConnectorAPI.java	Unit test dummy of bTrade class
CDBGenObject.java	Unit test dummy of bTrade class
CDBMessage.java	Unit test dummy of bTrade class
com.btrade.gnx.connectorapi.jar	Version of bTrade connectorAPI
com.btrade.gnx.connectorapi-logger.jar	Version of bTrade connectorAPI
hello.class	Dummy/test hello world Java app executable
hello.java	Dummy/test hello world Java app
Inetd.conf	Copy of bTrade box's inetd.conf file
Mqbtrade	Shell script used to run MQRbTrade
MQbTrade.dat	Copy of temp interface file to bTrade connectorAPI
MQbTrade.ini	Copy of temp interface file to bTrade connectorAPI
MQData.class	MQData object executable
MQData.java	MQData object source
MQGet.class	Test Java app to get messages from a queue executable
MQGet.java	Test Java app to get messages from a queue
MQPut.class	Test Java app to put messages to a queue executable
MQPut.java	Test Java app to put messages to a queue
MQRbTrade.class	MQRbTrade main executable
MQRbTrade.java	MQRbTrade main
Mqrbtrade.mqm	Queue configurations for MQRbTrade
mqrbrade.xsd	XMLSchema for MQRbTrade
MQRbTrade_ProgOperationGuide.doc	MQRbTrade operations guide
MQRbTradeProgDesignSpec.doc	MQRbTrade program design spec (old)
MQRequest.class	MQRequest object executable
MQRequest.java	MQRequest object
MQRProcException.class	MQRProcException object executable
MQRProcException.java	MQRProcException object
MQStatus.class	MQStatus object executable
MQStatus.java	MQStatus object
Services	Copy of bTrade box's services file
temp.Thu Jun 14 08_22_47 EDT 2001.out	MQRPEPS Debug output
temp.Thu Jun 14 08_24_50 EDT 2001.out	MQRPEPS Debug output
tr1.xml	MQRPEPS IVP testcase
tr1debug.out	MQRPEPS IVP Debug output
tr1xml.out	MQRPEPS IVP XML output

<i>File name</i>	<i>File description</i>
tr2.xml	MQRPEPS IVP testcase
tr2debug.out	MQRPEPS IVP Debug output
tr2xml.out	MQRPEPS IVP XML output
XML4J-J-bin.3.1.1.tar.gz	XML4J Parser UNIX distribution

PEPS

<i>File name</i>	<i>File description</i>
Employee.class	Oracle supplied JDBC IVP executable
Employee.java	Oracle supplied JDBC IVP
hello.class	Dummy/test hello world Java app executable
hello.java	Dummy/test hello world Java app
MQData.class	MQData object executable
MQData.java	MQData object source
MQGet.class	Test Java app to get messages from a queue executable
MQGet.java	Test Java app to get messages from a queue
mqrpeps	Shell script used to run MQRPEPS
MQPut.class	Test Java app to put messages to a queue executable
MQPut.java	Test Java app to put messages to a queue
MQRequest.class	MQRequest object executable
MQRequest.java	MQRequest object
MQRPEPS.class	MQRPEPS main executable
MQRPEPS.java	MQRPEPS main
mqrpeps.mqm	Queue configurations for MQRPEPS
mqrpeps.xsd	XMLSchema for MQRPEPS
MQRPEPS_Arch.vsd	VISIO diagram of MQRPEPS
MQRPEPS_ProgOperationGuide.doc	MQRPEPS operations guide
MQRPEPSProgDesignSpec.doc	MQRPEPS program design spec (not started)
MQRProcException.class	MQRProcException object executable
MQRProcException.java	MQRProcException object
MQStatus.class	MQStatus object executable
MQStatus.java	MQStatus object
PEPS_Test_Report.doc	MQRPEPS test report
SQLCallList.class	SQLCallList object executable
SQLCallList.java	SQLCallList object
temp.Thu Jun 14 11_14_10 EDT 2001.out	MQRPEPS Debug output
temp.Thu Jun 14 12_51_13 EDT 2001.out	MQRPEPS Debug output
temp.Thu Jun 14 16_15_44 EDT 2001.out	MQRPEPS Debug output
temp.Thu Jun 14 16_16_11 EDT 2001.out	MQRPEPS Debug output
tr1.xml	MQRPEPS IVP test case
tr1debug.out	MQRPEPS IVP Debug output
tr1xml.out	MQRPEPS IVP XML output
tr2.xml	MQRPEPS IVP test case
tr2debug.out	MQRPEPS IVP Debug output
tr2xml.out	MQRPEPS IVP XML output

<i>File name</i>	<i>File description</i>
types.out	List of SQL types
XML4J-J-bin.3.1.1.tar.gz	XML4J Parser UNIX distribution

NSLDS

<i>File name</i>	<i>File description</i>
JCL	
Cmpjcl02.txt	Compile job to compile adapters NSBATCH1 and NSBATCH2
MQCKTIBA.txt	Job to start the OS/390 TSO Batch Trigger Monitor
MQCKTIEN.txt	Job to stop the OS/390 TSO Batch Trigger Monitor
Proc	
PELL.txt	The procedure that executes the adapters and the NSLDS pilot
Source - Adapters	
NSBATCH1.txt	Batch adapter to pull messages from MQ and create NSLDS PELL File.
NSBATCH2.txt	Batch adapter to push reply messages to MQ from the ERROR file from NSLDS.
Batch Trigger Monitor	
CKTIBAT2.txt	OS/390 TSO Batch Trigger Monitor
CKTIEND.txt	Batch Trigger Monitor stop program
MQ Object Definitions	
MQADMN1.NTT1.OBJECTS.DEFS.txt	Objects for NSLDS development and test environments

CPS

<i>File name</i>	<i>File description</i>
MQ Object Definitions	
MQADM2.CPT1.DEV-N-TSO.OBJECTS.DEFS.txt	Objects for CPS development online and batch environments
MQADM2.CPT1.TST2.OBJECTS.DEFS.txt	Objects for CPS test online environments. The batch is for NSLDS specific.

DLSS

<i>File name</i>	<i>File description</i>
VALID_SSNS.DAT	Valid social security numbers
366821582.OUT === CI024S1.DAT	Sample output file
DLSSMQ.H	Include file for c programs
MQLOAN.COM	Command procedure
MQGET.C	C program – gets message from queue (Batch)
MQPUT.C	C program – puts message to queue (Batch)
REALPUT.C	C program – puts message to queue (Real-time)
REALGET.C	C program – gets message from queue (Real-time)
NOFILE.DAT	File containing no output file created message
366821582.IN == CI001S1.INP	Sample input file

<i>File name</i>	<i>File description</i>
CI001S1.FDL	FDL file used to convert input file to correct format
EAI.TST	Script file containing MQ object definitions
START_MQ_BATCH_JOBS.COM	Command file to start MQ jobs – started at system startup time
TRIG_CHANNEL.COM	Command file to trigger the sender channel

WebSphere Server – SU35E5

<i>File name</i>	<i>File description</i>
CORE.JAVA	MQ Adapter for sending/getting messages to/from a message queue
E5startstop.txt	Script file to start/stop MQSeries on the SU35E5 server
E16startstop.txt	Script file to start/stop MQSeries on the SU36E16 server
E17startstop.txt	Script file to start/stop MQSeries on the SU35E17 server
Defs.dat	Script file containing the MQ Object definitions on the EAI Bus servers (SU35E16 and SU35E17)

APPENDIX B: REFERENCE MATERIAL

For more information on the software and hardware prerequisites for the OS/390, please refer to the “MQSeries for OS/390 v5.2 Program Directory” and the “MQSeries for OS/390 v5.2 Concepts and Planning Guide” books on the IBM website:

(url= <http://www-4.ibm.com/software/ts/mqseries/library/manualsa/>).

For more information on WebSphere Application Server prerequisites, please refer to the “MQSeries for Windows NT and 2000 Quick Beginnings” book on the IBM website:

(url= <http://www-4.ibm.com/software/ts/mqseries/library/manualsa/>).

For more information on EAI BUS prerequisites, please refer to the “MQSeries for Windows NT and 2000 Quick Beginnings” book on the IBM website(url= <http://www-4.ibm.com/software/ts/mqseries/library/manualsa/>).

For more information on DLSS prerequisites, please refer to the “MQSeries for Compaq (DIGITAL) OpenVMS System Management” book on the IBM website:

(url= <http://www-4.ibm.com/software/ts/mqseries/library/manualsa/>).

For more information on PEPS prerequisites, please refer to the “MQSeries for HP-UX v5.2 Quick Beginnings” book on the IBM website:

(url= <http://www-4.ibm.com/software/ts/mqseries/library/manualsa/>).

For more information on BTrade prerequisites, please refer to the “MQSeries for HP-UX v5.2 Quick Beginnings” book on the IBM website:

(url= <http://www-4.ibm.com/software/ts/mqseries/library/manualsa/>).

For more information on how to customize MQSeries objects for application specific requirements, please refer to the IBM website:

(url= <http://www-4.ibm.com/software/ts/mqseries/library/manualsa/>).

For more information on MQSeries application error handling, event monitoring and MQSI error handling, please refer to the following books:

“MQSeries Application Programming Reference”

“MQSeries Event Monitoring”

“MQSeries Integrator Introduction and Planning”

on the IBM website: (url= <http://www-4.ibm.com/software/ts/mqseries/library/manualsa/>).

For more information on managing clusters and developing a custom cluster workload exit, please refer to the “MQSeries Queue Manager Clusters” book on the IBM website:

(url= <http://www-4.ibm.com/software/ts/mqseries/library/manualsa/>).

For more information on the MQSeries Integrator Control Center and the MQSeries commands and control commands, please refer to the following books:

“MQSeries Integrator Using the Control Center”

“MQSeries MQSC Command Reference”

“MQSeries Systems Administration”

“MQSeries for Compaq (DIGITAL) OpenVMS System Management”

“MQSeries for OS/390 System Administration Guide”

on the IBM website: (url= <http://www-4.ibm.com/software/ts/mqseries/library/manualsa/>).

For more information on the MQSI configuration manger, please refer to the “MQSeries Integrator Using the Control Center” book on the IBM website:

(url= <http://www-4.ibm.com/software/ts/mqseries/library/manualsa/>).

MQSeries Application Programming Guide can be found at:

<http://www-4.ibm.com/software/ts/mqseries/library/manualsa/> - Latest family books

MQSeries Application Programming Reference can be found at:

<http://www-4.ibm.com/software/ts/mqseries/library/manualsa/> - Latest family books

MQSeries Application Messaging Interface manual can be found at:

<http://www-4.ibm.com/software/ts/mqseries/library/manualsa/> - Latest family books

MQSeries Using C++ manual can be found at:

<http://www-4.ibm.com/software/ts/mqseries/library/manualsa/> - Latest family books

MQSeries Using Java manual can be found at:

<http://www-4.ibm.com/software/ts/mqseries/library/manualsa/> - Latest family books

