DISCUSSION DRAFT
10-27-04

WASHINGTON STATE
ADMINISTRATIVE OFFICE OF THE COURTS

OXCI Software Quality Assurance Report

October 27, 2004

MTG Management Consultants, L.L.C.
1111 Third Avenue, Suite 2700
Seattle, Washington 98101-3201
206.442.5010 206.442.5011 *fax*
www.mtgmc.com

TABLE OF CONTENTS

Page

I. OVERVIEW 2
A. OBJECTIVES 2
B. SCOPE 3
C. APPROACH 3

II. PROCESS AND METHODS 5
A. SQA PROCESS 5
B. SIMULATING THE PRODUCTION ENVIRONMENT 5
C. ARCHITECTURAL COMPLIANCE TESTING 6
D. FUNCTIONAL COMPLIANCE TESTING 6
E. DEFECT REPORTING 7

III. FINDINGS 9
A. ARCHITECTURAL REQUIREMENTS 9
B. FUNCTIONAL REQUIREMENTS 17
C. AUXILIARY REQUIREMENTS 25

APPENDIX A – SQA ASSURANCE TEST SCRIPTS
APPENDIX B – ISSUE LOG
APPENDIX C – REVISION HISTORY

I. OVERVIEW

I. OVERVIEW

The state courts of Washington and Georgia are leading the Open XML Court Interface (OXCI) consortium of state courts in the development of an open source court electronic filing application. This middleware application is to be an open source implementation of an Electronic Filing Manager (EFM), conforming to the Organization for the Advancement of Structured Information Standards (OASIS) LegalXML Court Filing standards. The application is being developed in three code and testing iterations. This document is a quality assurance assessment of the code and documentation resulting from the first and second development iterations.

A. OBJECTIVES

The objectives of each software quality assurance (SQA) assessment are to verify that the code and documentation are in compliance in each of the following areas:

- *Architecture* – The application must adhere to the architectural requirements described in the following application design documents:
  - » OXCI EFM Architecture.
  - » OXCI EFM Software Requirements.
  - » OXCI EFM Software Design.
- *Functionality* – The application must support the functional requirements described in the following requirements documents:
  - » OXCI EFM Architecture.
  - » OXCI XML Interface Specifications.
  - » OXCI EFM Software Requirements.
- *Auxiliary* – The application must support some auxiliary requirements described in the following document:
  - » OXCI EFM Software Development Plan.

B. SCOPE

The scope of this assessment includes all the deliverables produced at the end of each development iteration by counterclaim.com, inc. All deliverables are available on the project Web site at *http://oxci.sourceforge.net*. The deliverables include:

- *OXCI EFM Source Code* – A ZIP archive that includes the source code and scripts for building, running, and testing the application. The source code is also available via the Concurrent Versions System (CVS) on the project Web site.
- *OXCI EFM Binary Distributions* – A ZIP archive that includes the shell script, Java ARchive (JAR), JavaServer Page (JSP), Java Servlet, and Java 2 Platform, Enterprise Edition (J2EE) Enterprise ARchive (EAR) files necessary for installing and running the application. The binary distribution can be completely built from the source code.
- *OXCI EFM Documentation* – A ZIP archive that includes the technical documentation for the EFM, including HTML-formatted Application Programming Interface (API) documentation, generated automatically from the source code by the Javadoc tool, and the following guides:
  - » OXCI EFM Installation Guide.
  - » OXCI EFM Developer's Guide.
  - » OXCI EFM ebXML Guide.
- *OXCI EFM Iteration Test Reports* – Documents that summarize the functional status of each component of the application based on the results of unit and integration testing. This includes HTML-formatted reports generated by JUnit, the unit testing tool.

C. APPROACH

SQA is traditionally an independent analysis of the code and documentation upon completion of the code. Based on the iterative approach to development employed in this project, our quality assurance program includes a review of the code and documentation produced during each of the development iterations. The SQA process helps to identify any inconsistencies between the code and the functional requirements of the application as described in the design documents.

* * * * * *

The rest of the document will describe the SQA process, methods, and findings.

II. PROCESS AND METHODS

II. PROCESS AND METHODS

This section describes the SQA process and methods applied in the OXCI EFM development project. It includes an overview of the process and each of the following methods:

- Simulating the Production Environment
- Architectural Compliance Testing
- Functional Compliance Testing
- Defect Reporting

A. SQA PROCESS

In this project, we used the following SQA process, which we have developed based on our experience and on industry best practices:

- Extract the functional requirements from the design documents and organize them according to the compliance areas (architecture, functionality, auxiliary).
- Implement a quality assurance test bed that simulates a production environment.
- Develop tests for compliance for each of the functional requirements.
- Perform compliance tests and document any defects in detail.
- Report any defects to the developers using a bug-tracking tool.
- Verify resolution of defects from previous development iterations.
- Summarize all defects from the completed development iterations in a quality assurance report.

B. SIMULATING THE PRODUCTION ENVIRONMENT

The SQA environment is intended to simulate a variety of production environments by including support for multiple application and database servers. The SQA environment consists of the following:

- SQA server.
  - » Intel Pentium PC with 10 GB hard drive and 256 MB RAM.
  - » Fedora Core Release 2 Linux.
  - » Sun Microsystems' Java 2 SDK 1.4.2.
  - » JBoss 3.2.5 and 4.0 application servers.
  - » MySQL 3.23 database server.
  - » Symantec Corporation's AntiVirus Scan Engine 4.3.
  - » Trend Micro, Inc.'s InterScan Web Security Suite 1.0.
  - » Hermes freebXML Message Service Handler (MSH) 0.93.
- SQL Server.
  - » Intel Pentium PC.
  - » Microsoft Windows Server 2003.
  - » Microsoft SQL Server 2000 database server.
- Oracle server.
  - » Intel Pentium PC.
  - » Microsoft Windows Server 2003.
  - » Oracle 9i database server.

C. ARCHITECTURAL COMPLIANCE TESTING

Architectural compliance testing is performed by subjective evaluation of the complete source code and documentation, including the included API documentation, against the architectural requirements described in the design documents.

D. FUNCTIONAL COMPLIANCE TESTING

Functional compliance testing is performed by building the code, executing the unit tests, deploying the application to each application server and each database server in the SQA environment, and executing scripts to test each use case described in the design documents. The test scripts are included in APPENDIX A.

E. DEFECT REPORTING

Defects discovered through the SQA process are reported to the developers through the issue log located at *http://sourceforge.net/tracker/?atid=107304&group_id=7304*. Each bug is summarized in one line that shows the bug request ID, a summary, the date and time the bug was entered, a priority from 1 (low) to 9 (high), and the status (e.g., open, closed) as follows:

Request ID	Summary	Open Date	Priority	Status
1002400	Servlet vs. ReqResp vs. WebContainee vs. OpenEFMComponen	2004-08-02 20:42	5	Closed

By clicking on the Summary line, users may see much more information on the bug, including a detailed description and any responses from the developers or other users.

A snapshot of the issue log is included in APPENDIX B.

III. FINDINGS

III. FINDINGS

This section presents a quality assurance assessment of the OXCI EFM code and documentation with regard to the following groups of requirements described in the OXCI design documents:

- Architectural Requirements
- Functional Requirements
- Auxiliary Requirements

The following subsections describe the specific requirements and findings determined in each of these areas.

A. ARCHITECTURAL REQUIREMENTS

The OXCI EFM application is the core of a larger system architecture that is defined in the OXCI design documents. The EFM application must comply with the requirements of the OXCI architecture to ensure that the EFM is compatible with the other components of the total system. Specifically, the EFM must:

- Support all the system components defined in the architecture.
- Include all the software components defined in the architecture.
- Include all the classes and interfaces defined in the architecture.
- Adhere to the design decisions in the architecture.
- Adhere to the policies defined in the architecture.
- Support the business models defined in the architecture.
- Support the graphical user interface (GUI) elements defined in the software design.

The remainder of this subsection details each of these requirements, as well as the related defects and other findings.

1. Support all the system components defined in the architecture.

The system components are the high-level modules of the complete electronic filing solution. Each system component has a specific role and may be physically separate or managed separately from other components. The OXCI architecture defines the system components as follows:

System Component	Definition	Supported
EFM	The EFM server represents the middleware that connects the EFSP, CMS, DMS, and Web browser components. It includes the Web server, application server, and database server components. The OXCI EFM server may be hosted on a Linux, UNIX, or Windows platform.	✓
EFSP	The EFSP server is the electronic filing service provider's system that interfaces with the EFM server. It may, in fact, be another implementation of the OXCI EFM server using the architecture described above.	✓
CMS	The CMS is the case management system that stores case data, including information about documents filed with the court. The CMS is specific to each court or organization.	✓
DMS	The DMS is the document management system that stores electronic or imaged versions of court documents. The DMS is specific to each court or organization.	✓
Web Browser	The Web browser represents the client system and application used to support clerk review and EFM administration. The OXCI EFM supports the Internet Explorer Web browser.	✓

Notes

a. The application provides the EFM component, a clerk review and administration Web interface, an interface to the EFSP system, and an interface to the CMS and DMS adapters.

b. The CMS and DMS adapters are outside the scope of the EFM itself.

Defects

No defects have been identified in this area.

2. Include all the software components defined in the architecture.

The software components are logical modules of the EFM application. Software components may contain subcomponents. The OXCI architecture defines the software components within the EFM and a set of products to support each component, including both commercial and open source solutions. The compatibility with each of the products will be tested in functional compliance testing. The OXCI software components are supported by the EFM code as follows:

Component	Sub-component	Summary	Products	Supported
Web Server	EFM GUI	The EFM GUI represents the server-side code used to support clerk review and EFM administration.	Tomcat	✓
Web Server	MSH	This represents the MSH that connects the EFSP with the EFM application.	freebXML MSH, XMLGlobal GoXML Messaging Server	See Note a.
Application Server	EFM Application	The EFM application subcomponent provides the core business logic for the application.	JBoss, IBM WebSphere	✓
Application Server	XML Parser	The XML parser provides a service for traversing and extracting data stored in XML.	Xerces	✓
Database Server	Database	The database server provides persistence for the MSH and the EFM application.	MySQL, IBM DB2, Oracle	✓

Notes

a. In order to eliminate the need for an ebXML MSH, counterclaim chose to incorporate ebXML Messaging Service (ebMS) support in the EFM and support the ebMS header elements with the SOAP w/Attachments API. However, the EFM does not fully implement the ebMS specification as evidenced by the fact that it does not populate the mandatory elements in the ebXML Messaging Header (From, To, CPAID, ConversationID, Service, Action, MessageData, and Timestamp). This has no impact on testing with the provided EFSP simulator but will prevent interoperability with other ebXML implementations.

Defects

No defects have been identified in this area.

3. Include all the classes and interfaces defined in the architecture.

The OXCI architecture identifies data and system classes, as well as classes that represent the realizations of interfaces on the EFSP, CMS, and DMS. The names in the OXCI architecture are only suggestions, and the developers are free to assign different names to these classes. The designed classes with descriptions and their corresponding classes in the actual EFM code are as follows:

Classes	Description	Associated Classes in Code	Supported
FilingSubmissions	Describes a filing and is submitted from the EFSP to the EFM.	LxmlFiling	✓
FilingConfirmations	Describes a confirmation to a filing and is returned from the EFM to the EFSP.	LxmlResponse	✓
FilingQueries	Used by the EFSP to request information from the EFM.	Query	✓
FilingResponses	Used by the EFM to return the results of queries to the EFSP.	QueryResponse	✓
CourtPolicy	Describes the filing policies of a particular court.	CourtPolicy	✓
AbstractManager	Represents an abstract transaction manager. This class encapsulates the functionality common to the FilingManager, QueryManager, and PolicyManager classes.	AbstractOpenEFM Component	✓
FilingManager	Manages the filing and notice processes and provides the functionality for clerk review. This class implements the FilingService and ReviewGUI interfaces.	LxmlFilingManager	✓

Classes	Description	Associated Classes in Code	Supported
QueryManager	Manages the query and response processes. This class extends the AbstractManager class and implements the QueryService interface.		See Note a.
PolicyManager	Manages court policy requests. This class extends the AbstractManager class and implements the PolicySer-vice interface.	CourtPolicyImpl	✓
SecurityManager	Manages authentication and logging. This class implements the AdminGUI interface.	SecurityManager	✓
StorageManager	Manages data storage.	EJBObjectManager	✓
AbstractValidator	Represents an abstract schema validator. This class encapsulates the functionality common to the FilingValidator and QueryValidator classes. It includes methods for validating the envelope and the attached documents using external virus checking software.	LxmlValidator	✓
FilingValidator	Validates court filings. This class extends the AbstractValidator class.	LxmlValidator	✓
QueryValidator	Validates court queries and responses. This class extends the AbstractValidator class.		See Note a.
CMSConnector	Connects the FilingManager and QueryManager objects and the CMSService and DMSService interfaces.	CMSConnector	✓
EFSPStub	Implements the EFSPService interface.	EFSPImpl	✓
CMSAdapterStub	Implements the CMSService interface.	CMS-DMS Stub	✓
DMSAdapterStub	Implements the DMSService interface.	CMS-DMS Stub	✓

Notes

a. The current version of the EFM does not implement the query/response interface.

Defects

The EFM meets this requirement. The following defect has been identified and closed:

Request ID	Summary	Open Date	Priority	Status
1002400	Servlet vs. ReqResp vs. WebContainee vs. OpenEFMComponen	2004-08-02 20:42	5	Closed

4. Adhere to the design decisions in the architecture.

The OXCI architecture identified a number of design decisions regarding which protocol and schema standards the EFM should support. These decisions were made after careful evaluation of the alternatives in each area. The OXCI design decisions are supported in the EFM as follows:

Design Issue	OXCI Decision	Supported
Network Protocols	TCP/IP	✓
Communication Protocols	HTTP 1.1	✓
Messaging Protocols	ebMS 2.0	Partial. See Note a.
Authentication Protocols	EFSP: Secure Sockets Layer (SSL) as Needed; Filer: None	✓
Encryption Protocols	SSL; Future ebMS for XML Encryption as Needed	✓
Application Data Structure Definitions	XML Schema	✓
Application Envelope Schema	Court Filing Blue When Available; New Global Justice XML Data Model (GJXDM) 3.0-Based Envelope Element	✓
Application Object Schema	Court Filing Blue When Available; GJXDM 3.0-Based Court Filing, Query/Response, and Court Policy Objects in Interim	✓
Service Description Schema	Web Services Description Language (WSDL)	Partial. See Note b.
Collaboration Agreement Schema	None	See Note c.

Design Issue	OXCI Decision	Supported
Registry and Repository Schema	Universal, Description, Discovery and Integration (UDDI)	See Note c.
Database Interface	Relational	✓
Location of the Clerk Review Interface	In the EFM	✓

Notes

a. The EFM does not fully implement the ebMS 2.0 specification.

b. Although the EFM includes WSDL definitions for the EFSP, CMS, and DMS interfaces, the WSDL definitions were not used to implement these interfaces.

c. The Collaboration Agreement Schema and UDDI registry and repository were out of scope for this project.

Defects

No defects have been identified in this area.

5. Adhere to the policies defined in the architecture.

The OXCI architecture included a number of policy requirements that added further functional and implementation constraints. The OXCI policy requirements are supported by the EFM as follows:

Policy	Supported
Use freely licensed and open source technologies.	✓
Simplify the architecture to minimize the cost and complexity of implementation.	See Note a.
Scale to support both large and small courts.	✓
Support all the electronic filings for a single court.	✓
Support filing fees and payments managed through an external payment system.	See Note a.
Support antivirus scanning.	✓

Notes

a. Some pilot sites have reported issues implementing the EFM in their environments. However, J2EE applications are inherently complex and require some customization for each environment during deployment.

b. The SQA environment did not include an interface with an external payment system to test the processing of filing fees and payments.

Defects

The EFM meets this requirement. The following defect has been identified and closed:

Request ID	Summary	Open Date	Priority	Status
1002405	Virus Scanning is Currently Unimple-mented	2004-08-02 20:51	5	Closed

6. Support the business models defined in the architecture.

The OXCI architecture defines a range of business models in which the EFM may be deployed. The business models are supported by the EFM as follows:

Business Model	Description	Supported
Court Control Model	The court manages both the EFSP and the EFM.	✓
Vendor Control Model	Vendors provide both the EFSPs and the EFM.	✓
Split Control Models	Vendors provide the EFSPs that connect to the EFM managed by the court.	✓
Single Source Control Model	Courts and automated legal firms all provide their own EFSPs and EFMs.	✓

Notes

a. The open source feature of the EFM makes it available for use by the court, the vendor, or both.

Defects

No defects have been identified in this area.

7. Support the GUI elements defined in the software design.

The OXCI Software Design illustrates the GUI elements that the EFM must support. The screens are supported by the EFM as follows:

GUI Element	Supported
Login	✓
Manage Filings	✓
Filing Detail	See Note a.
Deletion of Aged Filings	✓
Manage EFSPs	✓
EFSP Detail	✓
Manage Users	✓
User Detail	✓

Notes

a. The Filing Detail screen does not list any documents attached to the filing or allow the clerk to view the attachments.

Defects

The EFM does not fully meet this requirement. The following defects have been identified:

Request ID	Summary	Open Date	Priority	Status
1054900	Can't Review Attached Documents	2004-10-26 15:18	5	Open
926365	Customize UI for OXCI EFM	2004-03-30 16:18	3	Closed

B. FUNCTIONAL REQUIREMENTS

In addition to compliance with the OXCI architecture, the EFM application must comply with the functional requirements described in the design documents to ensure that the EFM meets the needs of the users. Specifically, the EFM must:

- Document how to build, deploy, use, and extend the application.
- Build the application successfully.
- Execute the unit tests successfully.
- Deploy the application successfully.
- Execute the use cases defined in the architecture successfully.
- Support the OXCI XML Interface Specifications.

The remainder of this subsection details each of these requirements, as well as the related defects and other findings.

1. Document how to build, deploy, use, and extend the application.

The OpenEFM includes existing documentation but the significant changes necessary to create the OXCI EFM require updates of the development and installation guides, APIs, and even in-line comments. The OXCI documentation requirements are met by the EFM as follows:

Documentation	Description	Supported
Installation and Users' Guide	Instructions for installing, configuring, and using the EFM.	✓ See Note a.
EFSP Simulation Guide	Instructions for simulating a filing into the EFM.	✓
Developer's Guide	Developer instructions for extending the EFM and for writing CMS and DMS adapters.	✓
ebXML Guide	Developer instructions for creating an ebXML Message.	✓
Java APIs	API documentation generated by Javadoc.	✓

Notes

a. counterclaim actually provides two separate installation guides that ultimately should be consolidated into a single installation guide:

   » An OpenEFM Installation Guide for building and running the stand-alone (non-J2EE) version of the application and the Ozone database. This guide also includes instructions for configuring accounts for the EFSPs and users and using the clerk review functions.

» A J2EE Installation Guide for building and deploying the J2EE/Enterprise JavaBeans (EJB) 2.0 version of the application with JBDC-accessible databases.

Defects

The EFM meets this requirement. The following defects have been identified and closed:

Request ID	Summary	Open Date	Priority	Status
1002407	Outdated Documentation	2004-08-02 20:54	5	Closed
1002406	Remove References to Ozone	2004-08-02 20:53	5	Closed
1002402	IMPLEMENTME and FIXME	2004-08-02 20:46	4	Closed

2. Build the application successfully.

The process to build the application as directed in the OpenEFM-Test-Plan-Install-Linux.txt document is:

```
# cd <openefm-repository>/build
# rf –fR ../out
# ant ear-me-roar
```

The following warnings and errors result from building the application:

Build Warnings and Errors	Notes
ozone-opp: [java] com/counterclaim/openefm/model/ozone/OzoneObjectManager.java:8: warning: Unable to determine the method name for the found update definition in line 8. [java] com/counterclaim/openefm/model/ozone/OzoneModelObject.java:12: warning: Unable to determine the method name for the found update definition in line 12. [java] com/counterclaim/openefm/model/lxml/ozone/OzoneLxmlFiling.java:16: warning: Unable to determine the method name for the found update definition in line 16. [java] com/counterclaim/openefm/model/user/ozone/OzoneUserManager.java:16: warning: Unable to determine the method name for the found update definition in line 16.	See Note a.

Build Warnings and Errors	Notes
stub: [copy] Copying 35 files to /home/oxci/openefm/out/dist/Web/stub/WEB-INF/lib [copy] Copying 3 files to /home/oxci/openefm/out/dist/Web/stub/WEB-INF [war] Building war: /home/oxci/openefm/out/dist/Web/stub.war [war] Warning: selected war files include a WEB-INF/Web.xml which will be ignored (please use Webxml attribute to war task) ear-me-roar: [war] Building war: /home/oxci/openefm/out/dist/Web/admin.war [war] Warning: selected war files include a WEB-INF/Web.xml which will be ignored (please use Webxml attribute to war task) [ear] Building ear: /home/oxci/openefm/out/dist/Web/openefm.ear	See Note b.

Notes

b. This warning and bug #926363, below, result from the fact that the Java compiler does not understand certain comments in the Ozone files. According to counterclaim:

> "The //update comments are used by the ozone object persistence layer. Ozone generates persistent objects based on those interfaces; the //update comment tells ozone which method calls should invoke state persistence."

These warnings can be safely ignored or could be eliminated altogether if the Ozone objects are removed from the build.

c. This warning results from the fact that certain application servers require a separate configuration file, Web.xml, outside the main archive file. JBoss, however, requires that the Web.xml be inside the WEB-INF directory of the archive file. Therefore, this warning should be ignored.

Defects

The EFM meets this requirement. The following defects have been identified and closed:

Request ID	Summary	Open Date	Priority	Status
935313	Missing Java files for UUIDs and TimestampIDs	2004-04-14 17:16	5	Closed
926363	Incorrect Use of //Update	2004-03-30 16:13	5	Closed
926361	Deprecated Methods	2004-03-30 16:11	3	Closed
926357	Invalid Web.Xml ENTITY	2004-03-30 16:00	5	Closed

3. Execute the unit tests successfully.

counterclaim provides a test report including the results of all unit tests. The SQA environment should produce the same results running the unit tests.

All of the JUnit test case files were executed by using the command:

```
# cd <openefm-repository>/build
# ant tests
```

The following table contains a list of unit tests that report a failure or error:

Test Class	Unit Test	Error Message	Notes
OpenEFMControl-lerTestCase	testFilingMethods	java.security.PrivilegedActionException: com.sun.xml.messaging.saaj.SOAPExceptionImpl: Message send failed. Caused by: java.net.ConnectException: Connection refused.	See Note b.
OXCIProcessor TestCase	testFilingMethods	java.security.PrivilegedActionException: com.sun.xml.messaging.saaj.SOAPExceptionImpl: Message send failed. Caused by: java.net.ConnectException: Connection refused.	See Note b.

Notes

a. The execution of unit tests in the SQA environment was generally consistent with the test report provided by counterclaim.

b. Regarding the two failed tests, counterclaim includes the following note in the OpenEFM-Test-Plan.txt document that may explain these errors:

```
Note that some tests will fail unless you have started the cms-dms-stub
Web app' on port 8080, as some tests rely on that service.
```

Defects

No defects have been identified in this area.

4. Deploy the application successfully.

J2EE applications should be portable across platform, application server, and database environments. As described in subsection II.B., the SQA environment includes a subset of the range of components targeted in the OXCI architecture. The EFM supports the components in the SQA environment as follows:

Component	Sub-component	Summary	Products	Supported
Platform	Operating System	The operating system provides the software environment in which the other components run.	Fedora Core Release 2 Linux	✓
Web Server	EFM GUI	The EFM GUI represents the server-side code used to support clerk review and EFM administration.	Tomcat	✓
	MSH	This represents the MSH that connects the EFSP with the EFM application.	freebXML MSH	See Note a.
Application Server	EFM Application	The EFM application subcomponent provides the EJB container for the core business logic for the application.	JBoss 3.2.5	✓
			JBoss 4.0	See Note b.
	XML Parser	The XML parser provides a service for traversing and extracting data stored in XML.	Xerces	✓
Database Server	Database	The database server provides persistence for the MSH and the EFM application.	MySQL 4.23	✓
			Oracle 9i	See Note c.
			SQL Server 2000	✓

Notes

a. In order to eliminate the need for an ebXML MSH such as the freebXML MSH, counter-claim chose to incorporate ebMS support in the EFM and support the ebMS header elements with the SOAP w/Attachments API.

b. The EFM supports the JBoss 3.2.5 application server but not the recently released JBoss 4.x application server.

c. The EFM supports MySQL and SQL Server 2000 but not Oracle 9i due to issues with Oracle storing binary long objects (BLOBs) necessary to store images and binary attachments. This may be a flaw in the JDBC drivers provided by Oracle.

Defects

The EFM does not fully meet this requirement. The following defects have been identified:

Request ID	Summary	Open Date	Priority	Status
1038634	Oracle Issue Storing BLOBs	2004-10-01 11:21	5	Open
1036310	JBoss 4.0 Incompatibilities	2004-09-28 09:45	5	Open
1013660	VARCHAR BINARY Datatype Incompatible With SQL Server	2004-08-21 21:35	5	Closed
1002399	Hardcoded Configuration Options	2004-08-02 20:38	5	Closed
961256	Hardcoded File and Directory References	2004-05-26 16:46	5	Closed

5. Execute the use cases defined in the architecture successfully.

The OXCI architecture defines a set of six use cases that the EFM must support. These use cases describe how the key users interact with the EFM. The OXCI use cases are supported by the EFM as follows:

Use Case	Description	Support
Submit Filing	A filer submits a filing with the court.	✓
Review Filing	A clerk reviews and accepts or rejects filings submitted to the court.	Partial. See Notes a, b, c.
Query Filing	A user requests information on a specific case from the court CMS.	See Note d.
Query Policy	A user obtains the policies specific to a court.	See Note d.

Use Case	Description	Support
Manage Accounts	An administrator creates, modifies, and removes accounts and privileges on the EFM.	✓
View Logs	An administrator reviews the system and application logs for the EFM.	✓

Notes

a. When a clerk tries to review a filing with a PaymentRequest attachment, he/she receives "An error occurred while attempting to retrieve filings" message.

b. When a clerk rejects a filing, the EFSP does not receive an asynchronous FilingConfirmation with the disposition of "rejected."

c. When the filing is docketed, the filing status does not change to "filed," and the EFSP does not receive an asynchronous FilingConfirmation with the disposition of "filed."

d. The current version of the EFM does not implement the query interfaces.

Defects

The EFM does not fully meet this requirement. The following defects have been identified:

Request ID	Summary	Open Date	Priority	Status
1054330	Can't Review Filings With Payment Requests	2004-10-25 20:48	5	Open
1038627	efspID Not Being Stored in the Database	2004-10-01 11:07	5	Closed
1038625	EFSP Interface Does Not Support Callbacks	2004-10-01 11:05	9	Open
1013266	Received Disposition But Not Really Received	2004-08-20 21:55	9	Closed

6. Support the OXCI XML Interface Specifications.

The OXCI XML Interface Specifications defined schemas for each XML interface in the EFM. The OXCI XML Interface Specifications are supported in the EFM as follows:

XML Schema	Version	Use Cases	Supported
Court Filing	0.9.1	Submit Filing, Review Filing	Partial. See Note a.
Query Response	0.9.1	Query Filing, Query Policy	See Note b.
Court Policy	0.9.1	Query Policy	See Note b.
Payment	0.4	Submit Filing, Review Filing	Partial. See Note c.

Notes

a. FilingConfirmation elements returned by the EFM only include the FilingDisposition. According to the XML Interface Specifications, FilingConfirmations must also include the following elements: j:SubmissionSubmittedDate, j:SubmissionSubmittedTime, j:SubmissionReceivedDate, j:SubmissionReceivedTime, cf:FilingSubmissionID, j:CaseTrackingID, and cf:DocumentConfirmation.

b. The current version of the EFM does not implement the query/response and query policy interfaces.

c. The EFM does not recognize PaymentRequest attachments.

Defects

The EFM does not fully meet this requirement. The following defects have been identified:

Request ID	Summary	Open Date	Priority	Status
1054904	FilingConfirmations are Incomplete	2004-10-26 15:28	9	Open
1049747	Liberty DMS Adapter Needs to Use Latest Schemas	2004-10-18 20:30	5	Closed
1013078	Use Latest OXCI Schemas	2004-08-20 12:15	5	Closed

C. AUXILIARY REQUIREMENTS

The OXCI EFM Software Development Plan document requires two auxiliary components that are not part of the core EFM. In addition, system testing necessitated a third auxiliary component. These additional requirements include:

- Provide a generic CMS adapter.

- Provide an adapter for the Liberty DMS.
- Provide an EFSP simulator.

1. Provide a generic CMS adapter.

The OXCI EFM Software Development Plan requires a generic CMS adapter intended to provide a template for developers of future CMS and DMS adapters.

The EFM includes a generic CMS adapter.

Component	Description	Supported
Generic CMS Adapter	Pseudo Code for an OXCI CMS or DMS Adapter	✓

Notes

There are no relevant notes in this area.

Defects

No defects have been identified in this area.

2. Provide an adapter for the Liberty DMS.

The OXCI EFM Software Development Plan requires an adapter for the Liberty DMS. This adapter must accept XML data and documents from the EFM, transform the XML data into Liberty's XML format, and upload the data and documents into the Liberty DMS.

The Liberty DMS adapter is provided in a separate distribution from the EFM.

Component	Description	Supported
Liberty DMS Adapter	OXCI Adapter for the Liberty DMS	✓

Notes

a. The adapter builds successfully with the provided build script.

b. The adapter has not been tested with the Liberty DMS to verify interoperability.

Defects

The EFM meets this requirement. The following defects have been identified and closed:

Request ID	Summary	Open Date	Priority	Status
1036312	No Libery-dms Build Script	2004-09-28 09:47	5	Closed
1012311	Unbundle Chelan DMS Adapter	2004-08-19 09:35	5	Closed

3. Provide an EFSP simulator.

In order to perform system-level testing on the EFM, counterclaim agreed to provide an EFSP simulator that submits filings to the EFM and receives synchronous and asynchronous filing confirmations from the EFM.

The OXCI EFM includes an EFSP simulator:

Component	Description	Supported
EFSP Simulator	OXCI Adapter for the Liberty DMS	✓

Notes

a. The EFSP simulator submits filings and receives the synchronous filing confirmations. However, the asynchronous filing confirmations are actually received and stored with the EFM Web application.

b. The EFSP simulator is limited to three attachments, besides the lead document and payment, per filing. Several courts participating in OXCI have expressed the need for up to 10 or more attachments per filing.

Defects

The EFM meets this requirement. The following defect has been identified and closed:

Request ID	Summary	Open Date	Priority	Status
926364	Missing Java Source for filing-client.sh	2004-03-30 16:15	5	Closed

APPENDIX A
SQA TEST SCRIPTS

SQA TEST SCRIPTS

The following three scripts were developed for the SQA environment to help automate repeated testing:

- build.xml
- test-received.xsl
- test-infected.xsl

The ANT script, build.xml, submits several types of filings to the EFM using the EFSP simulator client provided by counterclaim and collects and processes the synchronous responses. The types of filings tested include:

- Normal filings (including a lead document, payment, and one attached document).
- Filings with large documents.
- Infected filings.
- Filings with payment requests.
- Filings with multiple attachments.

For infected filings, the script uses the stylesheet in test-infected.xsl to verify the response includes an infection warning from the antivirus scanning engine. For all other filings, the script uses the stylesheet in test-received.xsl to verify the response includes a FilingDisposition of "received."

To run the tests, use the "ant tests" command.

build.xml:

```
<project name="OXCIQA" default="tests" basedir=".">
    <description>
     OXCI QA tests
   </description>
    <!-- set global properties for this build -->
    <property name="qa" location="/home/oxci/qa"/>
    <property name="samples" location="${qa}/samples"/>
    <property name="logs" location="${qa}/log"/>
    <property name="src" location="/home/oxci/openefm"/>
    <property name="dist" location="${src}/out/dist"/>
    <property name="Webapp" location="${dist}/Web/admin"/>
    <property name="endpoint" value="http://localhost:8080/admin/receiver/efsp/ebxml"/>
    <property name="username" value="gollum"/>
    <property name="password" value="precious"/>
    <!-- set default properties -->
    <property name="deffiling" location="${samples}/FilingSubmissions.xml"/>
    <property name="defdocument" location="${samples}/TestDocument.pdf"/>
    <property name="defpayment" location="${samples}/Payment.xml"/>
    <property name="defattachment1" value="${samples}/TestDocument.pdf"/>
    <property name="defattachment2" value=""/>
    <property name="defattachment3" value=""/>
    <property name="defsoaptest" value="${samples}/test-received.xsl"/>
    <target name="clean" description="clean up">
        <delete dir="${logs}"/>
    </target>
    <target name="init" description="initialize the tests">
        <!-- Create the time stamp -->
        <tstamp/>
        <!-- Create the tests directory structure used by compile -->
        <mkdir dir="${logs}"/>
    </target>
    <target name="tests" depends="init" description="run all tests">
        <antcall target="normal"/>
        <antcall target="largedoc"/>
        <antcall target="infecteddoc"/>
        <antcall target="paymentrequest"/>
        <antcall target="attachments"/>
    </target>
    <target name="normal" description="test normal filing">
        <antcall target="filingtest">
            <param name="test" value="normal"/>
            <param name="filing" value="${deffiling}"/>
            <param name="document" value="${defdocument}"/>
            <param name="payment" value="${defpayment}"/>
            <param name="attachment1" value="${defattachment1}"/>
            <param name="soaptest" value="${defsoaptest}"/>
        </antcall>
    </target>
    <target name="largedoc" description="test large document">
        <antcall target="filingtest">
            <param name="test" value="largedoc"/>
            <param name="filing" value="${deffiling}"/>
            <param name="document" value="${samples}/TestDocument.pdf"/>
            <param name="payment" value="${defpayment}"/>
            <param name="attachment1" value="${defattachment1}"/>
            <param name="soaptest" value="${defsoaptest}"/>
        </antcall>
    </target>
    <target name="infecteddoc" description="test infected filing">
        <antcall target="filingtest">
            <param name="test" value="infecteddoc"/>
            <param name="filing" value="${deffiling}"/>
            <param name="document" value="/home/oxci/qa/samples/eicar.com"/>
            <param name="payment" value="${defpayment}"/>
            <param name="attachment1" value="${defattachment1}"/>
```

```
            <param name="soaptest" value="${samples}/test-infected.xsl"/>
        </antcall>
    </target>
    <target name="paymentrequest" description="test payment request">
        <antcall target="filingtest">
            <param name="test" value="paymentrequest"/>
            <param name="filing" value="${deffiling}"/>
            <param name="document" value="${defdocument}"/>
            <param name="payment" value="${samples}/PaymentRequest.xml"/>
            <param name="attachment1" value="${defattachment1}"/>
            <param name="soaptest" value="${defsoaptest}"/>
        </antcall>
    </target>
    <target name="attachments" description="test multiple attachment">
        <antcall target="filingtestattachments">
            <param name="test" value="attachments"/>
            <param name="filing" value="${deffiling}"/>
            <param name="document" value="${defdocument}"/>
            <param name="payment" value="${defpayment}"/>
            <param name="attachment1" value="${defattachment1}"/>
            <param name="attachment2" value="${defattachment1}"/>
            <param name="attachment3" value="${defattachment1}"/>
            <param name="soaptest" value="${defsoaptest}"/>
        </antcall>
    </target>
    <target name="filingtest">
        <delete dir="${logs}/${test}"/>
        <mkdir dir="${logs}/${test}"/>
        <java classname="com.counterclaim.openefm.client.efspsim.SubmitFiling" fork="true" output="${logs}/${test}/output.log">
            <arg value="-f"/>
            <arg file="${filing}"/>
            <arg value="-d"/>
            <arg path="${logs}/${test}"/>
            <arg value="-c"/>
            <arg file="${document}"/>
            <arg value="-i"/>
            <arg file="${payment}"/>
            <arg value="-e"/>
            <arg value="${endpoint}"/>
            <arg value="-u"/>
            <arg value="${username}"/>
            <arg value="-p"/>
            <arg value="${password}"/>
            <arg value="-a"/>
            <arg value="${attachment1}"/>
            <jvmarg value="-DOpenEFM.Web.dir=${Webapp}"/>
            <!--    <jvmarg value="-Djava.protocol.handler.pkgs=com.sun.net.ssl.internal.www.protocol" />
    <jvmarg value="-Djavax.net.ssl.trustStore=${dist}/WEB-INF/keys/client.keystore"  /> -->
            <classpath>
                <fileset dir="${Webapp}/WEB-INF/lib">
                    <include name="**/*.jar"/>
                </fileset>
            </classpath>
        </java>
        <xslt basedir="${logs}/${test}" includes="filing-*/soap-response.xml" style="${soaptest}" destdir="${logs}/${test}"
out="response.txt"/>
    </target>
    <target name="filingtestattachments">
        <delete dir="${logs}/${test}"/>
        <mkdir dir="${logs}/${test}"/>
        <java classname="com.counterclaim.openefm.client.efspsim.SubmitFiling" fork="true" output="${logs}/${test}/output.log">
            <arg value="-f"/>
            <arg file="${filing}"/>
            <arg value="-d"/>
            <arg path="${logs}/${test}"/>
            <arg value="-c"/>
            <arg file="${document}"/>
            <arg value="-i"/>
```

```
            <arg file="${payment}"/>
            <arg value="-e"/>
            <arg value="${endpoint}"/>
            <arg value="-u"/>
            <arg value="${username}"/>
            <arg value="-p"/>
            <arg value="${password}"/>
            <arg value="-a"/>
            <arg value="${attachment1}"/>
            <arg value="-a"/>
            <arg value="${attachment2}"/>
            <arg value="-a"/>
            <arg value="${attachment3}"/>
            <jvmarg value="-DOpenEFM.Web.dir=${Webapp}"/>
            <!--     <jvmarg value="-Djava.protocol.handler.pkgs=com.sun.net.ssl.internal.www.protocol" />
    <jvmarg value="-Djavax.net.ssl.trustStore=${dist}/WEB-INF/keys/client.keystore"  /> -->
            <classpath>
                <fileset dir="${Webapp}/WEB-INF/lib">
                    <include name="**/*.jar"/>
                </fileset>
            </classpath>
        </java>
        <xslt basedir="${logs}/${test}" includes="filing-*/soap-response.xml" style="${soaptest}" destdir="${logs}/${test}"
out="response.txt"/>
    </target>
</project>
```

`test-received.xsl:`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format"
xmlns:SOAP-ENV="http://schemas.xmlso
ap.org/soap/envelope/" xmlns:conv="http://www.openuri.org/2002/04/soap/conversat
ion/" xmlns:eb="http://www.ebxml.org/namespaces/messageHeader" xmlns:filing="htt
p://oxci.sourceforge.net/xml/CourtFiling" xmlns:wsdl="http://oxci.sourceforge.ne
t/xml/wsdl/Local">
    <xsl:output method="text"/>
    <xsl:template match="/">
        <xsl:if test="/SOAP-ENV:Envelope/SOAP-
ENV:Body/filing:FilingConfirmations/FilingConfirmation/FilingDisposition/FilingDispositionCode='received'">
            <xsl:text>ok</xsl:text>
        </xsl:if>
    </xsl:template>

</xsl:stylesheet>
```

`test-infected.xsl:`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format"
xmlns:SOAP-ENV="http://schemas.xmlso
ap.org/soap/envelope/" xmlns:conv="http://www.openuri.org/2002/04/soap/conversat
ion/" xmlns:eb="http://www.ebxml.org/namespaces/messageHeader" xmlns:filing="htt
p://oxci.sourceforge.net/xml/CourtFiling" xmlns:wsdl="http://oxci.sourceforge.ne
t/xml/wsdl/Local">
    <xsl:output method="text"/>
    <xsl:template match="/">
        <xsl:if test="/SOAP-ENV:Envelope/SOAP-
ENV:Body/filing:FilingConfirmations/FilingConfirmation/FilingDisposition/FilingError/FilingErrorCode='error'">
            <xsl:text>ok</xsl:text>
        </xsl:if>
    </xsl:template>

</xsl:stylesheet>
```

APPENDIX B
ISSUE LOG

ISSUE LOG

These tables, extracted from the issue tracking tool at *http://sourceforge.net/tracker/?atid=107304&group_id=7304*, are summaries of all open and closed defects reported to date. Each defect includes a summary, age, priority, and status.

Open Defects

Request ID	Summary	Open Date	Priority	Status
1054904	FilingConfirmations are Incomplete	2004-10-26 15:28	9	Open
1054900	Can't Review Attached Documents	2004-10-26 15:18	5	Open
1054330	Can't Review Filings With Payment Requests	2004-10-25 20:48	5	Open
1038634	Oracle Issue Storing BLOBs	2004-10-01 11:21	5	Open
1038625	EFSP Interface Does Not Support Callbacks	2004-10-01 11:05	9	Open
1036310	JBoss 4.0 Incompatibilities	2004-09-28 09:45	5	Open

Closed Defects

Request ID	Summary	Open Date	Priority	Status
1049747	Liberty DMS Adapter Needs to Use Latest Schemas	2004-10-18 20:30	5	Closed
1038627	efspID Not Being Stored in the Database	2004-10-01 11:07	5	Closed
1036312	No Libery-dms Build Script	2004-09-28 09:47	5	Closed
1013660	VARCHAR BINARY Datatype Incompatible With SQL Server	2004-08-21 21:35	5	Closed
1013266	Received Disposition But Not Really Received	2004-08-20 21:55	9	Closed
1013078	Use Latest OXCI Schemas	2004-08-20 12:15	5	Closed
1012311	Unbundle Chelan DMS Adapter	2004-08-19 09:35	5	Closed
1002407	Outdated Documentation	2004-08-02 20:54	5	Closed
1002406	Remove References to Ozone	2004-08-02 20:53	5	Closed
1002405	Virus Scanning is Currently Unimple-mented	2004-08-02 20:51	5	Closed

Request ID	Summary	Open Date	Priority	Status
1002402	IMPLEMENTME and FIXME	2004-08-02 20:46	4	Closed
1002400	Servlet vs. ReqResp vs. WebContainee vs. OpenEFMComponen	2004-08-02 20:42	5	Closed
1002399	Hardcoded Configuration Options	2004-08-02 20:38	5	Closed
961256	Hardcoded File and Directory References	2004-05-26 16:46	5	Closed
935313	Missing Java Files for UUIDs and TimestampIDs	2004-04-14 17:16	5	Closed
926365	Customize UI for OXCI EFM	2004-03-30 16:18	3	Closed
926364	Missing Java Source for filing-client.sh	2004-03-30 16:15	5	Closed
926363	Incorrect Use of //Update	2004-03-30 16:13	5	Closed
926361	Deprecated Methods	2004-03-30 16:11	3	Closed
926357	Invalid Web.xml ENTITY	2004-03-30 16:00	5	Closed

APPENDIX C
REVISION HISTORY

REVISION HISTORY

Version	Date	Revised By	Description
1.0	10/27/04	Mr. James Cabral	This is the initial version of this quality assurance report.