SPAWAR®

Systems Center
ATLANTIC

License Plate Recognition Data (LPRD)
System Design Document

Prepared By:

Scientific Research Corporation

Version #2.3 Delivered on 10-19-2010

Distribution of this document authorized to federal, state, local, and tribal government agencies and their contractors only for administrative or operational use, December 2010. Other requests for this document shall be referred to the Space and Naval Warfare Systems Center, Code 54150, P.O. Box 190022, North Charleston, SC 29419-9022.

Acknowledgements

Role	Name	E-Mail Address
Developer	(b)(4), (b)(6)	
QA Manager		
Reviewer		
Senior Engineer		
Senior Engineer		
Developer		
Project Manager		
Reviewer		

Document Number	1
Document Name	LPR SSD 2.2.docx
Date Created (Draft)	05-06-2009
Date Approved	

Document Number	2
Document Name	LPR SSD 2.2.docx
Date Created (Draft)	08-23-2010
Date Approved	

Version Control

Version	Date	Author	Change Description
1.0	05-06-09	(b)(4), (b)(6)	Document created
1.1	05-12-09		Added sections 4 and 5 First release candidate
1.2	05-28-09		Grammatical changes Phrasing changes Formatting changes
1.3	06-04-09		Updates to UI Section Updates to Introduction
1.4	06-05-09		Grammatical changes Screenshot changes
1.5	06-09-09		Response to customer feedback Added table references Added XSD/XML examples Clarified language
2.0	06-16-09		Refactoring of layout Added more information on replicators Removed section 2, renumbered other sections Removed requirements traceability matrix Added system overview image
2.1	06-29-09		Greatly expanded the Data Replicator section Responded to many user comments Added UML sequence diagram for UI interaction
2.2	06-30-09		Proofreading edits
2.3	10-15-10		Updated document IAW new system installed at ARJIS and SLED. Final edits and proofreading.

Table of Contents

1. Introduction 1

1.1 Summary 1

1.2 Purpose 1

1.3 Scope 1

1.4 Methods 2

2. System Architectural Design 3

2.1 System Architecture 3

2.2 Data Types Description 4

2.3 Interface Description 6

2.4 Database Interface Description 6

3. Detailed Component Overview 7

3.1 Components Overview 7

3.2 Data Replicators 8

3.3 User Interface (UI) 13

3.4 JRA Router 14

3.5 JLEIE 14

4. User Interface Design 17

4.1 Header 17

4.2 Menu 18

4.3 UI Layout 20

Table of Figures

Figure 1 - Three Tier LPR System Architecture 3
Figure 2 - LPR System Component 7
Figure 3 - UML diagram for JLEIE query handling service 14
Figure 4 - UML diagram for JLEIE service for adding new data 15
Figure 5 - LPR Header for an authenticated user 18
Figure 6 – Menu Segment 18
Figure 7 – Navigation Menu composed of Menu Segments 19
Figure 8 - Layout Overview 20

Table Reference

Table 1 - LPR XML Example 4
Table 2 - Replicator Configuration in "replicator.properties" 8
Table 3 - ChangeLog Hibernate Mapping 12
Table 4 - NDI Reference Implementation new records Hibernate Query 12
Table 5 - Transforming a LicensePlateRead object 12
Table 6 - Example JLEIE Success Message 16
Table 7 - Examle JLEIE Error Message 16

1. Introduction

1.1 Summary

In the past, Law Enforcement Agencies (LEAs) have selected License Plate Recognition solutions aligned with their budgetary constraints and with the individual needs of their agency. This has resulted in the need for LEAs to directly interface and share data.

In response to the need to share information across agencies, the National Institute of Justice (NIJ) funded the Navy's Space and Naval Warfare Systems Center, Atlantic (SPAWAR) to develop a standardized LPR data sharing solution as an operational proof of concept. This solution implements the Department of Justice's (DOJ) Justice Reference Architecture (JRA) model.

The LPR solution includes:

- A cost-effective solution that integrates into existing proprietary LPR systems
- A capturing process for proprietary LPR data from individual agency locations
- Formatting/standardization procedures for the captured datasets using NIEM XML standards
- Data handling processes for uploading the standardized data into the centralized repository
- Data sharing regulations for the centralized repository including user permissions and incorporating required security measures
- Querying capabilities of the centralized repository
- A user-friendly, open-sourced, web User Interface (UI) aligned with agency-specific business procedures and regulations
- Secure data storage to prevent any modification of data outside of the source agency's repository

This LRP prototype enables the sharing of existing LPR data by converting it into standard format. The system supports a web based user interface format that can be easily used. It embraces free, open source, and web-enabled design methodologies that do not prevent third parties from interacting with the system.

1.2 Purpose

This document provides a detailed design specification of the LPRD Service Oriented Architecture (SOA) software application including architectural design requirements/components, data formatting procedures, and comprehensive descriptions/illustrations of LPRD software clearinghouse processes. This document details the system architecture, system requirements, front-end/back-end specifications, component/sub component specifications, and the software interface requirements.

Also provided in this document are future processes relating to continual improvement of the system, maintaining upgrades to the open-source coding (based on the National Information Exchange Model (NIEM) Extensible Markup Language (XML) standards/framework), and system processes to ensure effective application/maintenance of LPRD according to current accepted standards and regulations.

1.3 Scope

This document covers the components that comprise the LPRD operational prototype. It also details external and internal interfaces.

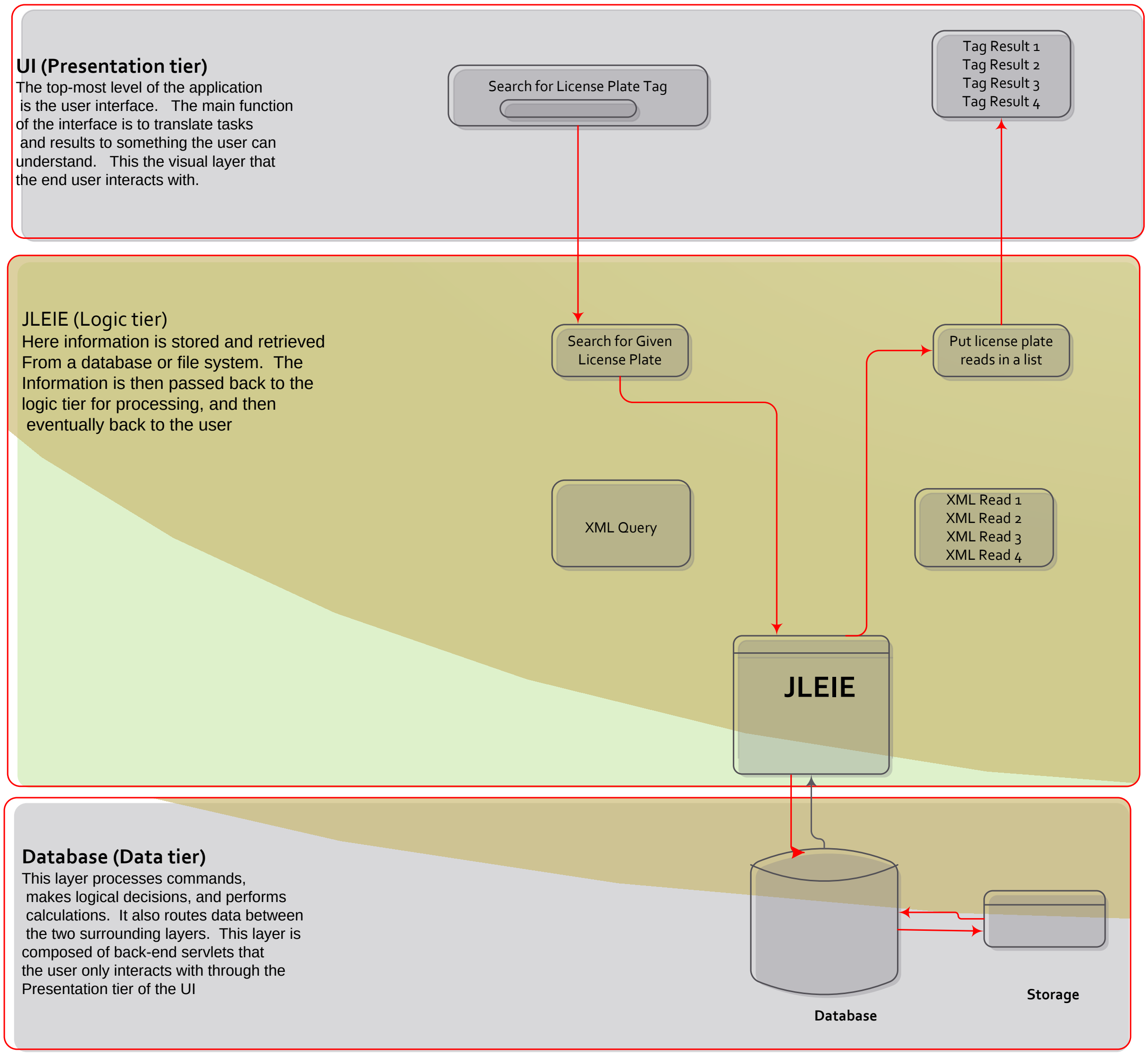
1.4 Methods

This document is based on the IEEE 1016-1998 standard for software design documents. In this document UML 2.0 will be used to model object interaction. The Extensible Markup Language (XML) format will be used to describe data interchange between components. The XML Schema Definition (XSD) format will be used to constrain and define the format of the XML messages.

2. System Architectural Design

2.1 System Architecture

The LPRD prototype is comprised of three major components: The User Interface (UI), the Java Law Enforcement Integration Engine (JLEIE), and the database. This is a classic Three Tier architectural pattern.



UI (Presentation tier)
The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand. This the visual layer that the end user interacts with.

JLEIE (Logic tier)
Here information is stored and retrieved From a database or file system. The Information is then passed back to the logic tier for processing, and then eventually back to the user

Database (Data tier)
This layer processes commands, makes logical decisions, and performs calculations. It also routes data between the two surrounding layers. This layer is composed of back-end servlets that the user only interacts with through the Presentation tier of the UI

Figure 1 - Three Tier LPR System Architecture

This architecture was chosen because it offers a great deal of separation of concerns. Each application layer is a band of responsibility which is key to the LPR system's speed. For example: the UI layer is not concerned with the process of submitting new license plate reads.

The UI is only concerned with where it submits the data and is ignorant of any other processes except the acknowledgement that the process completed.

2.2 Data Types Description

LPRD communicates by using the LPR IEPD. The IEPD serves as a contract between two endpoints sharing IEPD data. The IEPD specifies a format for License Plate Read events which is expressed in XML. This format is expressed in XML Schema Definition (XSD). The XSD includes:

Mandatory Fields:

- Event Date
- Event Time
- LPR System ID
- Activity Reason
- LPR Event ID
- Organization ID (ORI)
- LPR Vehicle
- Vehicle License Plate ID
- LPR Vehicle Plate Photo
- Geographic Coordinates
- Document Control Data

Optional Fields:

- LPR Vehicle Plate Text Correction
- LPR Vehicle Plate State Correction
- LPR Additional Photo
- LPR Direction
- LPR Camera ID
- LPR Recorded Lane

The below message produces an XML representation example.

Table 1 - LPR XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<lpr:LicensePlateRead
        xmlns:lpr="http://www.it.ojp.gov/jxdm/3.0.3/license-plate-reader"
        xmlns:j="http://www.it.ojp.gov/jxdm/3.0.3"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <j:EventDate>2006-05-04</j:EventDate>
  <j:EventTime>01:01:01.001</j:EventTime>
  <lpr:LPRSystemID>EXAMPLEORI</lpr:LPRSystemID>
  <lpr:ActivityReason>Law Enforcement Use</lpr:ActivityReason>
  <lpr:LPREventID>123456</lpr:LPREventID>
  <j:OrganizationORIID>
    <j:ID>EXAMPLEORI</j:ID>
  </j:OrganizationORIID>
  <lpr:LPRVehicle>
    <j:VehicleMakeCode>MAK</j:VehicleMakeCode>
    <j:VehicleModelCode>MOD</j:VehicleModelCode>
    <j:VehicleModelYearDate>2006</j:VehicleModelYearDate>
    <j:VehicleModelCodeText>Model</j:VehicleModelCodeText>
    <j:VehicleColorPrimaryText>Color</j:VehicleColorPrimaryText>
    <j:VehicleColorPrimaryCode>COL</j:VehicleColorPrimaryCode>
    <j:VehicleColorSecondaryText>Color</j:VehicleColorSecondaryText>
    <j:VehicleColorSecondaryCode>COL</j:VehicleColorSecondaryCode>
    <j:VehicleDoorQuantity>2</j:VehicleDoorQuantity>
    <j:VehicleLicensePlateID>
      <j:ID>TAG123</j:ID>
      <j:IDIssuingAuthorityText>SC</j:IDIssuingAuthorityText>
    </j:VehicleLicensePlateID>
  </lpr:LPRVehicle>
  <lpr:LPRVehiclePlatePhoto>ZGVmYXVsdA==</lpr:LPRVehiclePlatePhoto>
  <lpr:LPRGeographicCoordinate>
    <j:GeographicCoordinateLatitude>
      <j:LatitudeDegreeValue>80</j:LatitudeDegreeValue>
      <j:LatitudeMinuteValue>3</j:LatitudeMinuteValue>
      <j:LatitudeSecondValue>80</j:LatitudeSecondValue>
    </j:GeographicCoordinateLatitude>
    <j:GeographicCoordinateLongitude>
      <j:LongitudeDegreeValue>40</j:LongitudeDegreeValue>
      <j:LongitudeMinuteValue>12.2</j:LongitudeMinuteValue>
      <j:LongitudeSecondValue>18</j:LongitudeSecondValue>
    </j:GeographicCoordinateLongitude>
  </lpr:LPRGeographicCoordinate>
  <lpr:LPRMetadata>
    <j:MetadataFieldName></j:MetadataFieldName>
    <j:MetadataFieldValueText></j:MetadataFieldValueText>
  </lpr:LPRMetadata>
  <j:DocumentControlData>
    <j:DocumentCountryCode.fips10-4></j:DocumentCountryCode.fips10-4>
  </j:DocumentControlData>
  <lpr:LPRVehiclePlateTextCorrection></lpr:LPRVehiclePlateTextCorrection>
  <lpr:LPRVehiclePlateStateCorrection> </lpr:LPRVehiclePlateStateCorrection>
  <lpr:LPRAdditionalPhoto>ZGVmYXVsdA==</lpr:LPRAdditionalPhoto>
  <lpr:LPRDirection></lpr:LPRDirection>
  <lpr:LPRCameraID></lpr:LPRCameraID>
  <lpr:LPRRecordedLane> </lpr:LPRRecordedLane>
</lpr:LicensePlateRead>
```

This format is used within the system when an LPR event is communicated.   When multiple LPR events are transmitted a list element that contains multiple “lpr:LicensePlateRead” elements is given.

The data exchange formats are provided within the LPR commons library as part of the LPR source package. The Java objects (Plain Old Java Objects or POJOS) that store the data exchange messages when they are de-serialized are also included in the LPR source package.

2.3 Interface Description

This service's interface for this service is defined in Web Service Descriptor Language (WSDL) and is included in the IEPD documentation. This WSDL is attached below for convenience.

LPRReplicationRouterService.wsdl

The message portion of this WSDL is sent in the "sendLicensePlateRead" parameter and consists of a single License Plate Read event as described in section 2.2.

2.4 Database Interface Description

The LPRD database is designed to conform to MySQL Server version 5.1. The primary mechanism that JLEIE uses to access the database is Hibernate. Hibernate is a Plain Old Java Object (POJO) to Relational Database mapping tool. One of Hibernate's capabilities is to communicate in different database dialects; thus, LPRD is database agnostic. In order to use a different backend database with JLEIE only the configuration files would need to be changed to support a different dialect than MySQL.

It is also possible to communicate with the database using any other method that supports MySQL connections such as ODBJ or JDBC. There are no restrictions against these methods of access. There is no proprietary data or obfuscated table/column names to prevent direct data access.

3. Detailed Component Overview

3.1 Components Overview

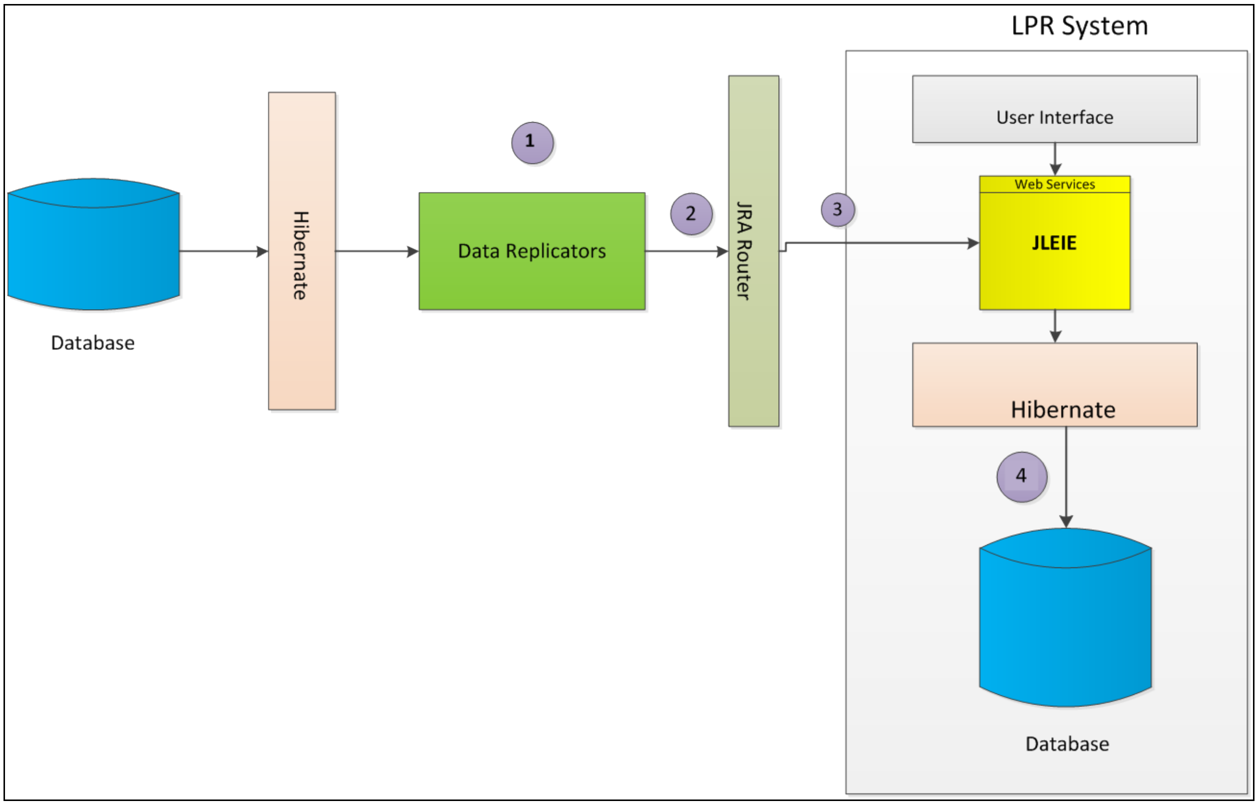


Figure 1 - LPR System Component

The LPRD component overview is shown in Figure 1. Label 1 illustrates the data replicators. Each data source (agency or otherwise) must have a data replicator attached to its data source (as described in section 3.2) to be able to access the data warehouse of the LPRD system. In the future there will be no replicators and LPRD will be dependent on the vendors to build it into their software. This will be accomplished through the web services that LPRD provides.

Label 2 shows the connection that every data replicator must make. The format for this exchange is described in section 2.3.

Label 3 marks the interface between the User Interface (UI) and Java Law Enforcement Integration Engine (JLEIE). The JLEIE provides a common way to store and share all law enforcement data. This is described in section 3.5. Like the replicators, JLEIE uses hibernate to map to data objects that will be inserted as rows in the database

Label 4 marks a Java Data Base Connectivity (JDBC) that is used by Hibernate to connect to whatever type or dialect of database is used. This mechanism is further described in section 2.4.

3.2 Data Replicators

LPRD is created and populated by data replicators installed at participating agencies. Once the necessary replicator software is installed, the data collected from the agencies is replicated to a central warehouse database. This “replication” is accomplished by sending messages to the JRA Router using the various data exchange formats described in section 2.

The reference implementation for the replicator works in the following manner as seen in the “replicator” source package. The application is started and polls the database for new records by using the hibernate configuration. New records are selected and transformed to LicensePlateRead objects through Hibernate. These LPR objects are then transformed into the XML format in section 2.2 with the transformer found in the “commons” package. The XML messages are then sent to the JRA-Router using the format described in the Web Service Definition Language (WSDL) (see section 2.3).

The reference replicator is configured using a single configuration file: “replicator.properties.” This file is generated by the replicator configuration utility, which is run by the replicator installer. Once configured, the replicator maps vendor-specific classes to the backend database, and each of these classes are able to transform themselves into and out of the common LicensePlateRead object.

Table 2 - Replicator Configuration in "replicator.properties"

reload.interval	How often to reload this configuration file, in milliseconds. Default: 10000
lpr.replicator.interval	How long to wait between attempts at polling the database. Default: 5000
lpr.replicator.exception.interval	How long to wait between attempts at polling the database after an exception is caught. Default: 60000
lpr.replicator.healthy.message.interval	How long to wait before printing a message that no new records have arrived. Default: 3600000 (1 hour)
lpr.replicator.max.age	How new records must be, in days, to be considered for replication.
lpr.replicator.inputs	Comma-separated list of databases which will be replicated. This is a list of prefixes that is used below. For example, lpr.replicator.inputs=ndi, then replace {prefix} with ndi, below.
{prefix}.vendor	Vendor of the LPR system being replicated, i.e. ndi, pips, elsag
{prefix}.type	Type of backend. Default: hibernate
{prefix}.max.results	How many records to replicate at one time from the database.
{prefix}.ws.security.enabled	Whether to use JRA-Router and encrypted xml or not. true/false.
{prefix}.ws.secure.repl.url	URL for JRA-Router, i.e. http://127.0.0.1:8080/router-jra/LPRReplicationRouterService?wsdl. Only used if {prefix}.ws.security.enabled is true.
{prefix}.ws.unsecure.repl.url	URL for JLEIE, i.e. http://127.0.0.1:9090/jleie/LPRReplicatorService?wsdl. Only used if {prefix}.ws.security.enabled is false.
{prefix}.xml.enc.keystore	The location of the keys used for communicating with the JRA Router, i.e. c:\origins\keys\replicator.xml.enc.keystore. This file is created when jra-router is first run.
{prefix}.org.oriid	The ORI of the agency that the replicator is installed at.

{prefix}.org.name	The long name of the agency that the replicator is installed at.
{prefix}.hibernate.dialect	Hibernate dialect of the database being replicated, i.e. org.hibernate.dialect.SQLServerDialect
{prefix}.hibernate.connection.driver_class	Classname of the JDBC driver, i.e. net.sourceforge.jtds.jdbc.Driver
{prefix}.hibernate.connection.url	JDBC URL of the database, i.e. jdbc:jtds:sqlserver://127.0.0.1:1433;databaseName=NDI_VBOF_SYNC
{prefix}.hibernate.connection.username	The user to use to login to the database.
{prefix}.hibernate.connection.password	The password for the database.
{prefix}.hibernate.current_session_context_class	Default: thread
elsag.images.root	(Elsag only) Folder storing images, i.e. d:\\data\\OperationalCenter\\I

Each replicator maps a different class to the backend database, and each of these classes are able to transform themselves into and out of our LicensePlateRead object. The code in Table 3 shows an example of this mapping.

Table 3 – NDI Vehicle Hibernate Mapping

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="gov.usdoj.nij.ilp.infosharing.origins.adapters.ndi.dao.Vehicles" table="VEHICLES">
    <id name="vehicleId" type="int">
      <column name="VehicleID"/>
      <generator class="identity"/>
    </id>
    <timestamp column="ImportDate" name="importDate" source="db"/>
    <property name="sessionId" type="int">
      <column name="SessionID" not-null="true"/>
    </property>
    <property name="vrm" type="string">
      <column length="12" name="VRM" not-null="true"/>
    </property>
    <property name="dt" type="timestamp">
      <column length="23" name="DT"/>
    </property>
    <property name="confidence" type="java.lang.Short">
      <column name="Confidence"/>
    </property>
    <property name="lane" type="string">
      <column length="50" name="Lane"/>
    </property>
    <property name="camera" type="string">
      <column length="50" name="Camera"/>
    </property>
    <property name="direction" type="string">
      <column length="50" name="Direction"/>
    </property>
    <property name="manualEntry" type="boolean">
      <column name="ManualEntry" not-null="true"/>
    </property>
  <property name="active" type="boolean">
   <column name="Active" not-null="true"/>
  </property>
  <property name="lat" type="java.lang.Double">
   <column name="Lat" precision="53" scale="0"/>
  </property>
  <property name="long_" type="java.lang.Double">
   <column name="Long" precision="53" scale="0"/>
  </property>
  <property name="stateCode" type="string">
   <column length="2" name="StateCode"/>
  </property>
  <property name="nationCode" type="string">
   <column length="2" name="NationCode"/>
  </property>
  <property name="vehicleMake" type="string">
   <column length="20" name="VehicleMake"/>
  </property>
  <property name="vehicleModel" type="string">
   <column length="20" name="VehicleModel"/>
  </property>
  <property name="vehicleType" type="string">
   <column length="20" name="VehicleType"/>
  </property>
  <property name="vehicleColor" type="string">
   <column length="20" name="VehicleColor"/>
  </property>
  <property name="plateRegionX" type="java.lang.Float">
   <column name="PlateRegionX" precision="24" scale="0"/>
  </property>
  <property name="plateRegionY" type="java.lang.Float">
   <column name="PlateRegionY" precision="24" scale="0"/>
  </property>
  <property name="plateRegionWidth" type="java.lang.Float">
   <column name="PlateRegionWidth" precision="24" scale="0"/>
  </property>
  <property name="plateRegionHeight" type="java.lang.Float">
   <column name="PlateRegionHeight" precision="24" scale="0"/>
  </property>
  <property name="crimeSweepId" type="int">
   <column name="CrimeSweepID"/>
  </property>
  <property name="notes" type="string">
   <column length="500" name="Notes"/>
  </property>
  <property name="systemId" type="string">
   <column length="50" name="SystemID"/>
  </property>
  <property name="orgId" type="string">
   <column length="50" name="OrgID"/>
  </property>
  <property name="modifiedDate" type="timestamp">
   <column length="23" name="ModifiedDate"/>
  </property>
  <set cascade="all-delete-orphan" inverse="true" name="images">
   <key column="VehicleID"/>
   <one-to-many class="gov.usdoj.nij.ilp.infosharing.origins.adapters.ndi.dao.Images"/>
  </set>
 </class>
</hibernate-mapping>
```

This indicates that the table “vehicles” in the configured database will be mapped to the class “Vehicles”.

The relationship between the Vehicles object and the LicensePlateRead object can be seen in the toLicensePlateRead method within Vehicles.java:

```
   /**
    * Converts this Vehicles object to a LicensePlateRead object.
    * @param defaultOriId The ORI to set in the LicensePlateRead object.
    * @param withImages Whether or not to include images in the LicensePlateRead
    * @return A new LicensePlateRead initialized to the values in this Vehicles object.
    */
   public LicensePlateRead toLicensePlateRead(String defaultOriId, boolean withImages) {
…
}
```

In order to use another database or vendor data store, new hibernate mappings and classes would need to be generated, then the classes modified to have a toLicensePlateRead method. The former can be accomplished with hibernate tools such as hbm2ddl.

The reference replicator uses the ChangeLog object to detect changes in the source database. The replicator then checks that table according to the interval described in 2. Table 4 describes the ChangeLog object’s mapping to the lprd_log table.

Table 1 - ChangeLog Hibernate Mapping

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="gov.usdoj.nij.ilp.infosharing.origins.core.data.ChangeLog" table="lprd_log">
    <id name="id">
      <column name="id"/>
      <generator class="native"/>
    </id>
    <property name="entityId" type="string">
      <column length="60" name="entityid"/>
    </property>
    <property name="activityType" type="string">
      <column length="20" name="activitytype"/>
    </property>
    <property name="changeType" type="string">
      <column length="1" name="changetype"/>
    </property>
    <property name="processed" type="string">
      <column length="1" name="processed"/>
    </property>
    <property name="tstamp" type="timestamp">
      <column length="23" name="tstamp"/>
    </property>
  </class>
</hibernate-mapping>
```

The reference NDI implementation uses the code in Table 5 to query the database for records that have changed. The implementation can detect if the records are either new or changed as the ‘ChangeLog’ table is first queried to get the maximum timestamp of the last-sent record.

Table 2 - NDI Reference Implementation new records Hibernate Query

```
            value = new Integer(lastChange.getEntityId());
             field = "vehicleId";

             Criteria criteria = session.createCriteria(Vehicles.class)
                                                  .add(Restrictions.gt(field, value))
                                                  .addOrder(Order.asc("vehicleId")) // always order by vehicleId
                                                  .setMaxResults(maxResults);
            List<Vehicles> newRecords = criteria.list();
```

The code above loads the first fifty data results which are newer than the last record (from checking the max id with the ‘ChangeLog’ object) that was replicated by the system. Hibernate then creates a list of Vehicles objects from the results of this query, which are then transformed to ‘LicensePlateRead’ objects.

Once the list of ‘LicensePlateRead’ objects is populated, each object can be individually transformed (as seen in Table 6) and sent to the JRA Router for storage in the central warehouse databse; see Replicator.java for details.

Table 3 - Transforming a LicensePlateRead object

```
String xml = TransformerManager.getTransformer(this).toXML(licensePlateRead);
```

3.3 User Interface (UI)

The UI is implemented using JavaServer Faces 2.0 (JSF2). JSF2 is a Model-View-Controller architecture (MVC). The model consists of a number of Java beans which talk to JLEIE by calling its web services. The view consists of a number of xhtml pages which display the data. JSF2 itself is the controller which connects the two.

Security is provided in the Logic tier through Spring Security with LDAP extensions. The currently logged in user is always available to the Logic Tier by calling User.getUser(). If this is called when no user is logged in, the result will be User.ANONYMOUS_USER. This provides strong container level security. It also provides security that enforces permissions based on URL. These measures prevent any unauthorized viewing of data and/or modification of settings that has not been authorized by the security layer.

The Spring Security layer contacts an Apache Directory Server which is an LDAP v3 compliant LDAP server. The LDAP entries are organized into two sections (each an ‘ou’ or organizational unit), Groups and Users. The Groups unit has four sub-units. Three of these are the main groups within the application: user, admin, and system admin. The fourth sub-unit is the Modules unit. Currently there is only one module available: the LPR module. The LPR module has one sub-unit, the LPR_READ unit. The other section, the Users unit, is where user ids are stored. A user either belongs to an organization (such as a police agency) or is a system administrator. Each organization serves as a parent unit to the users who are members. An organization is also a member of all the modules that its users should have access to. For example: an organization ‘ORG01’ would be a member of the LPR module if its users could have LPR_READ access to the application. Each user, then, is a member of the sub-units of the modules. A user ‘user01’ who is a member of ‘ORG01’ would then be able to be a member of the LPR_READ module unit.

The design of this LDAP layout is intended to provide maximum flexibility in adding new modules and allowing all user information to reside in the LDAP server.

A typical successful chain of interaction for an Investigator class User (a User that can only perform read and query actions) involves the following steps. First, the user logs on through Spring Security, which creates a java.security.Principal for the current session. This principal consists of an OriginsUserDetails object which stores the user’s username, ORI, and roles. Spring consults this object automatically to determine which URLs the user is allowed to view. This is configured in applicationContext-security as follows:

```
<intercept-url pattern="/login*" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
<intercept-url pattern="/lpr/*" access="ROLE_USER"/>
<intercept-url pattern="/admin/*" access="ROLE_ADMIN,ROLE_SYSTEM_ADMIN"/>
<intercept-url pattern="/systemadmin/*" access="ROLE_SYSTEM_ADMIN"/>
```

A user logged in with the ROLE_USER role may view any pages in the lpr subdirectory, such as the “Last 50 Hits” page or the “Search” page. These pages use JSF managed beans which call the JLEIE web services. The beans store lists of LicensePlateRead objects which are then displayed using JSF tags in the xhtml pages. As the UI is displaying the details of each LicensePlateRead it will make a HTTP GET request to the ImageController servlet which will return the binary image data for the current web browser to display.

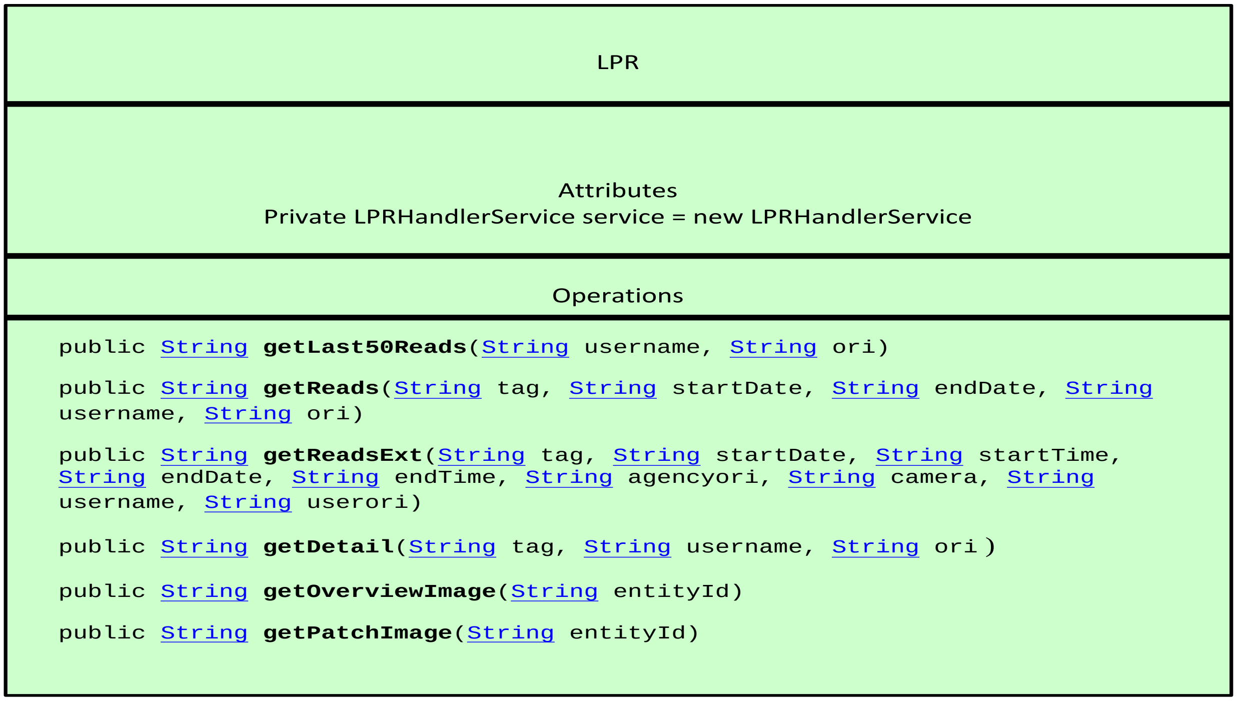
3.4 JRA Router

JRA-Router uses Apache XML Security to encrypt and decrypt XML. Encryption and decryption is accomplished with a 128-bit symmetric Advanced Encryption Standard (AES) key which is automatically generated the first time the JRA-Router service is initialized. This key is then distributed to clients who wish to use the JRA-Router service.

The JRA Router accepts messages as described in section 2.3.

3.5 JLEIE

The JLEIE component serves as the implementation of the data level services for LPRD. It publishes services for the storage, query, and retrieval of license plate information. The JLEIE server publishes services that are known by the JRA Router and any UI clients, or custom clients written by third parties.



LPR

Attributes
Private LPRHandlerService service = new LPRHandlerService

Operations

```
public String getLast50Reads(String username, String ori)

public String getReads(String tag, String startDate, String endDate, String
username, String ori)

public String getReadsExt(String tag, String startDate, String startTime,
String endDate, String endTime, String agencyori, String camera, String
username, String userori)

public String getDetail(String tag, String username, String ori )

public String getOverviewImage(String entityId)

public String getPatchImage(String entityId)
```

Figure 2 - UML diagram for JLEIE query handling service

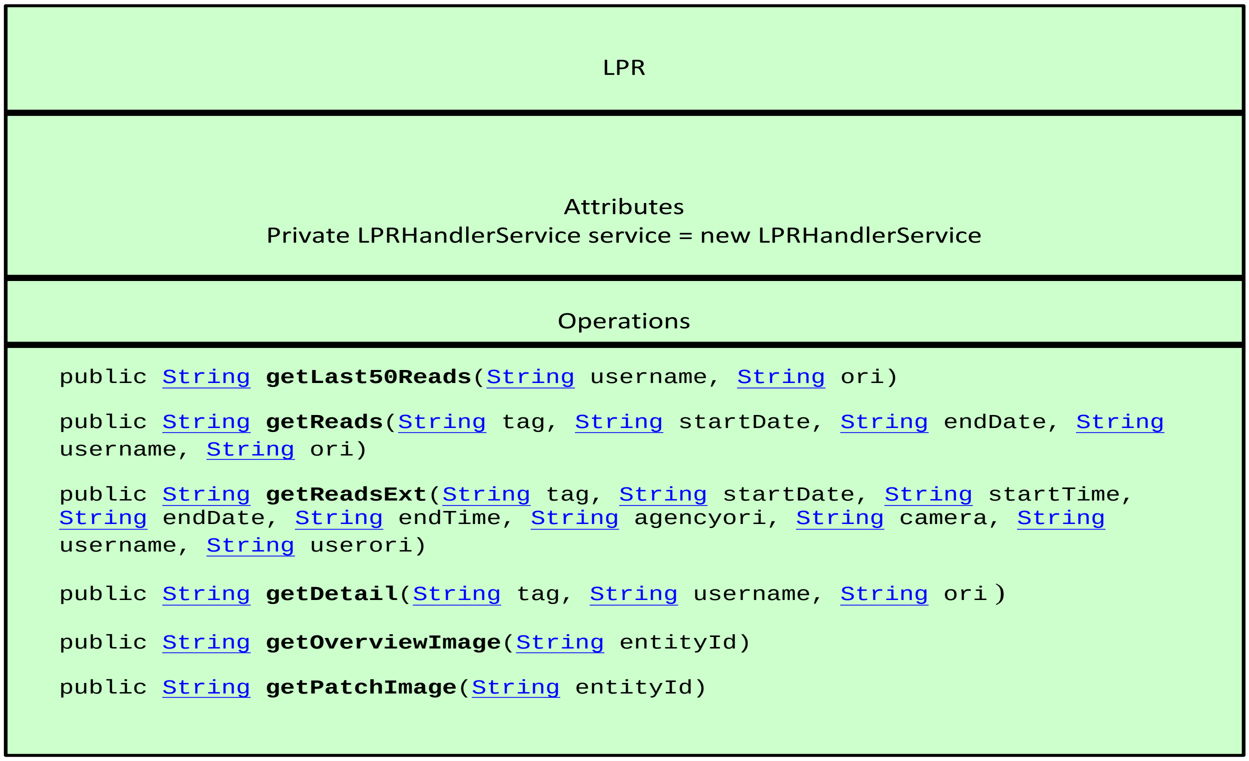
The LPRHandler service creates an XML response using data from the database that backs the data tier. It exists to provide participants (such as the UI) with data that can be presented to an end user.

The requests and the responses for this service are described here in WSDL format.

LPRHandler Service WSDL

LPRHandlerService.wsdl

The “getLast50Reads”, “getReads”, “getReadExt”, and “getDetail” methods all respond with a list of LPR event objects in the format described in section 2.2. The “getImage” method returns an LPR event with only the ID fields and the primary/secondary image fields populated. The “getOverviewImage” and “getPatchImage” methods return just the base64-encoded data for the particular image. These are intended to be called by the UI only when the image needs to be displayed to the user. Images are not returned by any of the other methods.



LPR

Attributes
Private LPRHandlerService service = new LPRHandlerService

Operations

```
public String getLast50Reads(String username, String ori)

public String getReads(String tag, String startDate, String endDate, String
username, String ori)

public String getReadsExt(String tag, String startDate, String startTime,
String endDate, String endTime, String agencyori, String camera, String
username, String userori)

public String getDetail(String tag, String username, String ori )

public String getOverviewImage(String entityId)

public String getPatchImage(String entityId)
```

Figure 3 - UML diagram for JLEIE service for adding new data

The LPRReplicator service manages incoming license plate read events. Third party participants' incoming messages are optionally routed to this service through the JRA Router. The format for incoming LPR events is described in section 2.2 and in the LPR IEPD. The JLEIE component then interacts with the backing database to store incoming license plate read events.

The WSDL for this service is attached here.

LPRReplicatorService.wsdl

The response from this service is either a “Success” message or an exception that message from the JLEIE component. At this time only code “0” and code “9999” are supported. Code “0” is the success code. Code “9999” is the generic exception code. Example messages are included.

Table 4 - Example JLEIE Success Message

```
<lpr:LicensePlateResult xmlns:lpr=\"http://www.it.ojp.gov/jxdm/3.0.3/license-plate-reader\"
targetNamespace=\"http://www.it.ojp.gov/jxdm/3.0.3/license-plate-reader\">
        <Code>0</Code>
        <Message>Success</Message>
</lpr:LicensePlateResult>
```

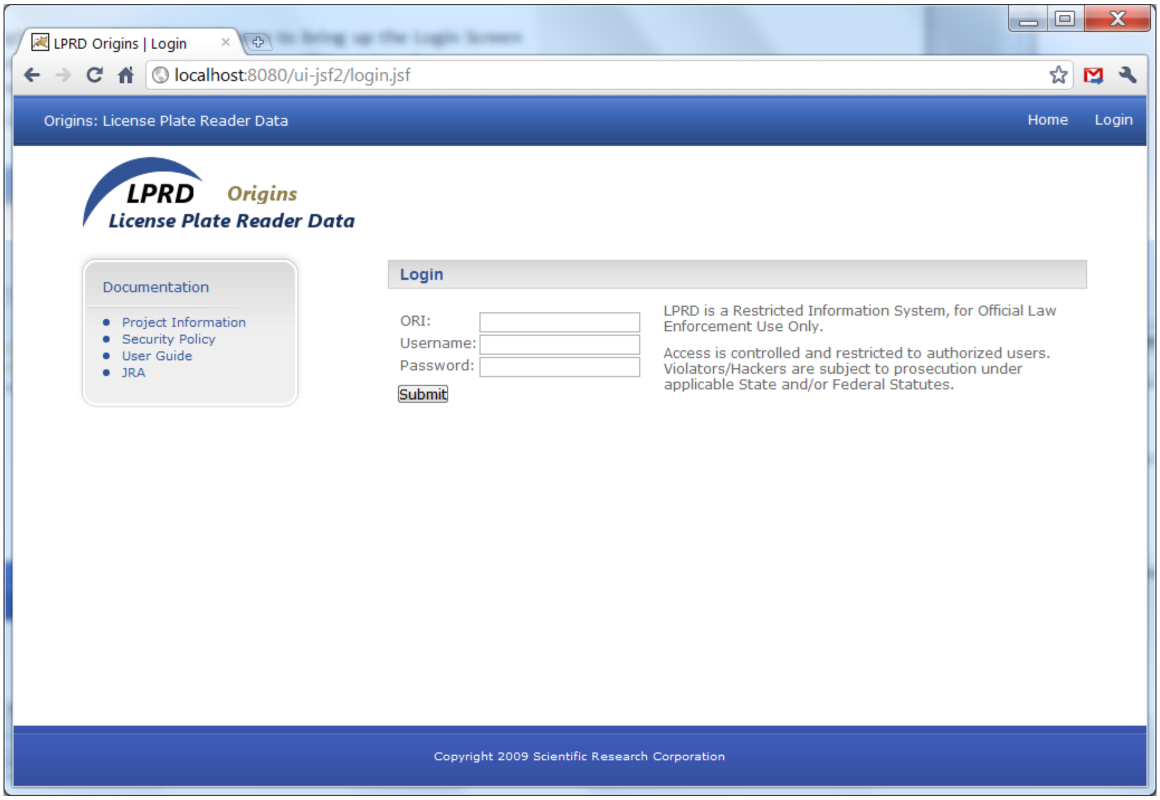
Table 5 - Examle JLEIE Error Message

```
<lpr:LicensePlateResult xmlns:lpr=\"http://www.it.ojp.gov/jxdm/3.0.3/license-plate-reader\"
targetNamespace=\"http://www.it.ojp.gov/jxdm/3.0.3/license-plate-reader\">
        <Code>9999</Code>
        <Message>Generic Error Has Occurred: java.lang.NullPointerException</Message>
</lpr:LicensePlateResult>
```

4. User Interface Design

4.1 Header

The Header is made up of two different sections that can be modified by developers as shown in Figures 4 and 5.

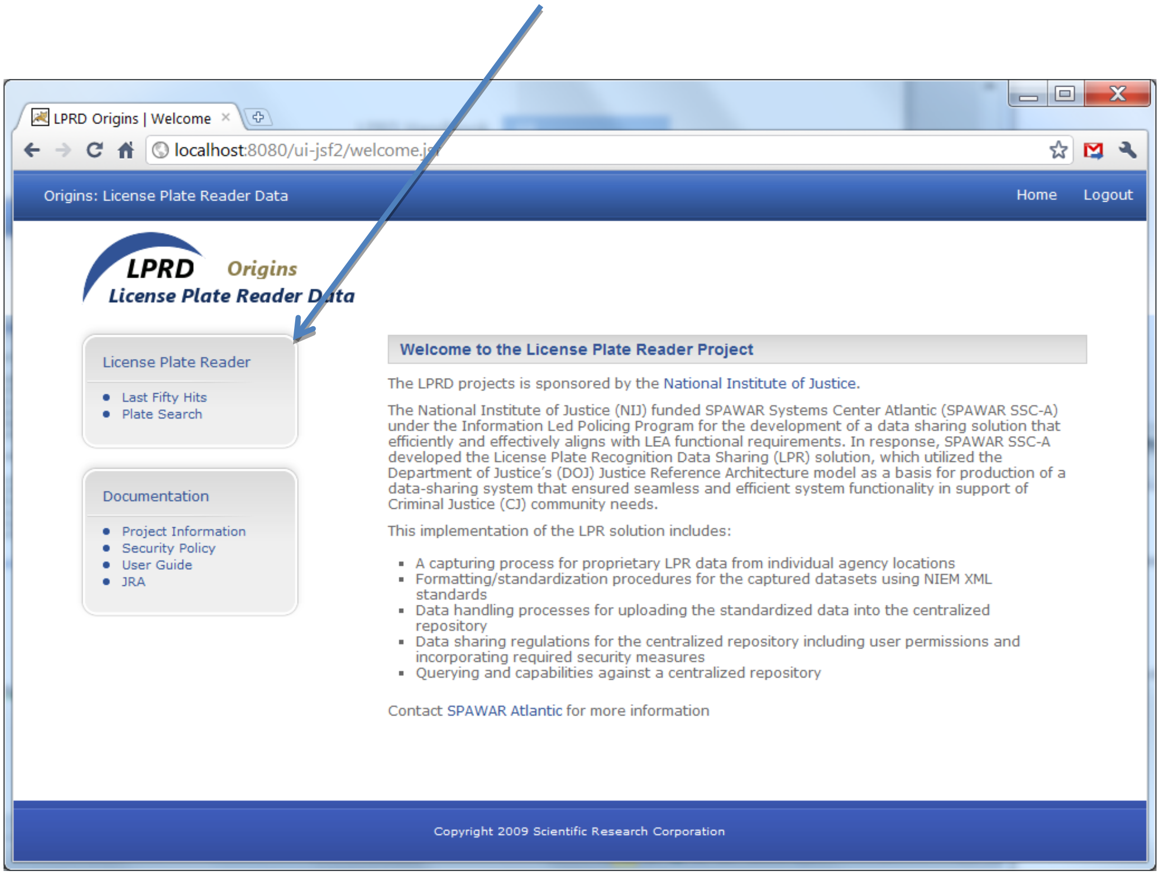


LPRD is a Restricted Information System, for Official Law Enforcement Use Only.

Access is controlled and restricted to authorized users. Violators/Hackers are subject to prosecution under applicable State and/or Federal Statutes.

Copyright 2009 Scientific Research Corporation

Figure 4 - LPR Header for a user who has not logged in



Welcome to the License Plate Reader Project

The LPRD projects is sponsored by the National Institute of Justice.

The National Institute of Justice (NIJ) funded SPAWAR Systems Center Atlantic (SPAWAR SSC-A) under the Information Led Policing Program for the development of a data sharing solution that efficiently and effectively aligns with LEA functional requirements. In response, SPAWAR SSC-A developed the License Plate Recognition Data Sharing (LPR) solution, which utilized the Department of Justice's (DOJ) Justice Reference Architecture model as a basis for production of a data-sharing system that ensured seamless and efficient system functionality in support of Criminal Justice (CJ) community needs.

This implementation of the LPR solution includes:

- A capturing process for proprietary LPR data from individual agency locations
- Formatting/standardization procedures for the captured datasets using NIEM XML standards
- Data handling processes for uploading the standardized data into the centralized repository
- Data sharing regulations for the centralized repository including user permissions and incorporating required security measures
- Querying and capabilities against a centralized repository

Contact SPAWAR Atlantic for more information

Copyright 2009 Scientific Research Corporation

Figure 5 - LPR Header for an authenticated user

The Header is made up of two different sections that can be modified by developers. The left hand side has a text container that can hold different messages, warnings or titles. The right hand side is for top level navigation links, such as logout or a direct return to the home page.

4.2 Menu

The UI menu changes depending on the permissions available to the currently logged in user. It is built with jsf ui:fragments that may be combined to present a variety of functionality to the user. Each fragment is composed of menu items that perform an action.

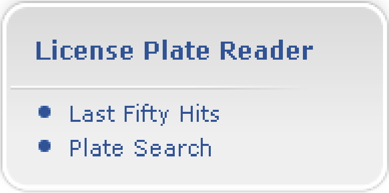


Figure 6 – Menu Segment

These fragments are built and composed in lpr-template.xhtml based on the permissions contained in the UserBean object.

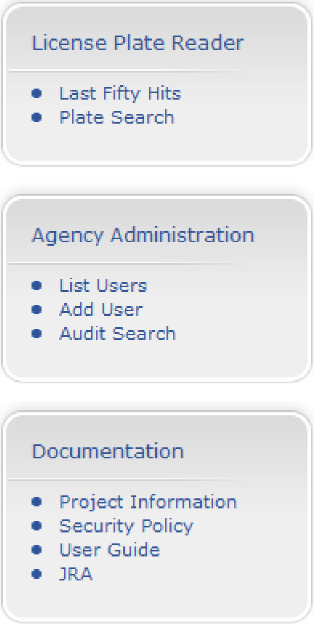
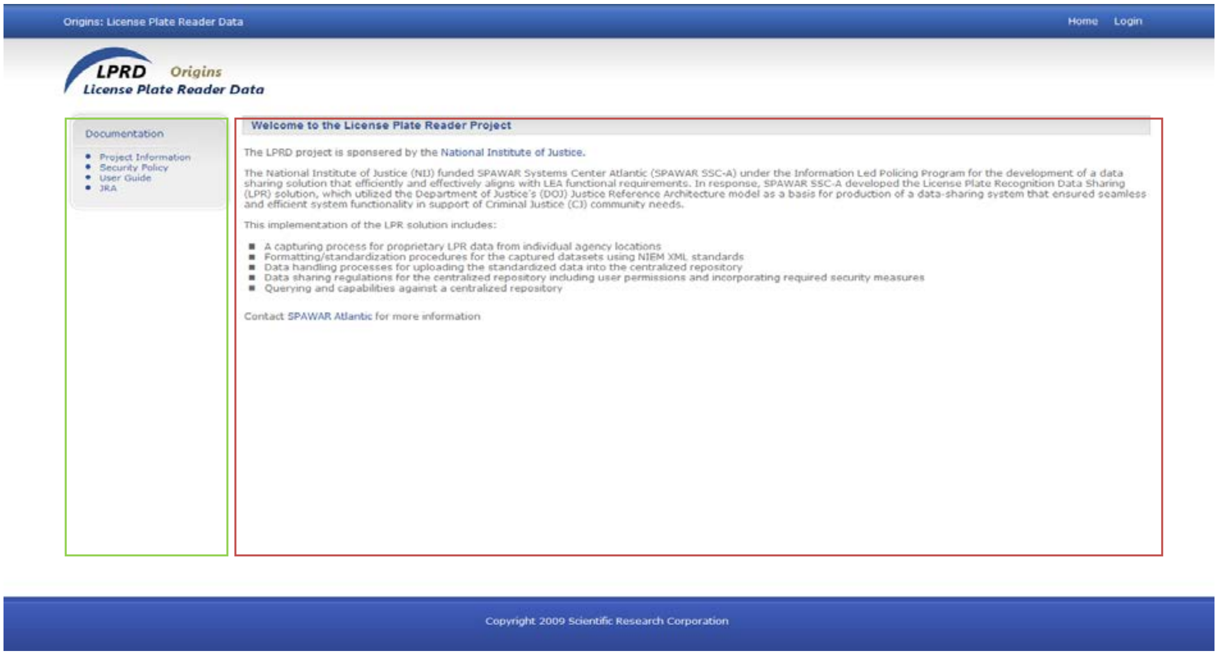


Figure 7 – Navigation Menu composed of Menu Segments

4.3 UI Layout



Welcome to the License Plate Reader Project

The LPRD project is sponsered by the National Institute of Justice.

The National Institute of Justice (NIJ) funded SPAWAR Systems Center Atlantic (SPAWAR SSC-A) under the Information Led Policing Program for the development of a data sharing solution that efficiently and effectively aligns with LEA functional requirements. In response, SPAWAR SSC-A developed the License Plate Recognition Data Sharing (LPR) solution, which utilized the Department of Justice's (DOJ) Justice Reference Architecture model as a basis for production of a data-sharing system that ensured seamless and efficient system functionality in support of Criminal Justice (CJ) community needs.

This implementation of the LPR solution includes:

- A capturing process for proprietary LPR data from individual agency locations
- Formatting/standardization procedures for the captured datasets using NIEM XML standards
- Data handling processes for uploading the standardized data into the centralized repository
- Data sharing regulations for the centralized repository including user permissions and incorporating required security measures
- Querying and capabilities against a centralized repository

Contact SPAWAR Atlantic for more information

Copyright 2009 Scientific Research Corporation

Figure 8 - Layout Overview

The UI uses a template “lpr-template.xhtml” to manage the content. The red box outlined in Figure 8 is the area that can be populated in JSF by the use of the tag <ui:define name=”content”>. The green outlined square shows the area where the Navigation Menu is composed. As described in section 4.2, it is built dynamically based on the roles of the logged in user.