

RFID EDGER SERVER IMPROVEMENTS

Inventor:
Kenneth R. Traub

[0001] The following Appendixes concern RFID Edge Server Improvements:

DynamicReaderConfiguration
EdgeDeviceConfiguration
ConfigurationDesign

[0002] These three documents relate to features for creating and modifying configuration information for RFID readers and other devices under the control of an RFID Edge Server. These documents disclose a metadata-driven description of configuration information, and the ability to dynamically define or change configuration without restarting the Edge Server or interrupting the activities of applications receiving data from the Edge Server. The first document is the main design doc, the second doc is an earlier design that was implemented but never released (but much of which was carried over to the present implementation), and the third is some supporting software that was contained in earlier product releases.

[0003] The features described in the first two documents include:

GUI-API mapping
Ualegui-mvc
Admin Console 1.3 Design
UALEGUI Visual Design
Unified ALEGUI-AI
Unified ALEGUI - Code Design

[0004] These six documents relate to two new features. One is a GUI that presents the dynamic configuration capability as described above. The other is a feature by which users can graphically define filtering and other processing operations for RFID data.

EPC-PS-Design
EPCPSWebGUIDesign
EPC-PS-Design

[0005] These three documents describe a new component of an RFID Enterprise Server, called the EPC Provisioning Service. It provides a way to centrally coordinate the assignment of unique serial numbers that are encoded by the RFID Edge Server onto RFID tags.

[0006] One embodiment may be implemented using a conventional general purpose or a specialized digital computer or microprocessor(s) programmed according to the teachings of the present disclosure, as will be apparent to those skilled in the computer art. Appropriate software coding can readily be prepared by skilled programmers based on the teachings of the present disclosure, as will be apparent to those skilled in the software art. The invention may also be implemented by the preparation of integrated circuits or by interconnecting an appropriate network of conventional component circuits, as will be readily apparent to those skilled in the art.

[0007] One embodiment includes a computer program product which is a storage medium (media) having instructions stored thereon/in which can be used to program a computer to perform any of the features presented herein. The storage medium can include, but is not limited to, any type of disk including floppy disks, optical discs, DVD, CD-ROMs, micro drive, and magneto-optical disks, ROMs, Rams, EPROM's, EPROM's, Drams, Rams, flash memory devices, magnetic or optical cards, Nano systems (including molecular memory ICs), or any type of media or device suitable for storing instructions and/or data.

[0008] Stored on any one of the computer readable medium (media), the present invention includes software for controlling both the hardware of the general purpose/specialized computer or microprocessor, and for enabling the computer or microprocessor to interact with a human user or other mechanism utilizing the results of the present invention. Such software may include, but is not limited to, device drivers, operating systems, execution environments/containers, and user applications.

[0009] The foregoing description of preferred embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations will be apparent to one of ordinary skill in the relevant arts. For example, steps performed in the embodiments of the invention disclosed can be performed in alternate orders, certain steps can be omitted, and additional steps can be added. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications that are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the claims and their equivalents.



ALE Configuration API

Version of 2004-06-29 at 11:54

This document is Company Confidential

Table of Contents

ALE Configuration API	1
1 Introduction	3
2 Use Cases	3
2.1 Administrative Console	3
2.1.1 Initial Product Evaluation	43
2.1.2 Tagging Station Reconfiguration	4
2.2 Discovery System	54
2.2.1 New Reader Discovered	5
2.3 Control Server	5
2.3.1 New Reader Discovered	5
2.3.2 Edge Server Decommissioned	5
2.4 Batch-Mode Configuration	6
3 Architecture Overview	7
4 Configuration Model	7
4.1 Referential Integrity	8
5 The ALE Configuration API	9
5.1 Physical Reader Methods	9
5.1.1 Helper objects	10
5.2 Logical Reader Methods	1140
5.3 Filter Methods	1244
5.4 Composite Reader Methods	1344
5.5 Trigger Methods	1442
6 Persistent Store	1543
7 Back/Forward Compatibility	1543
7.1 Upgrade	1543

Version of 29 June 2004 at 1154 hours 29 June 2004 at 1044 hours ~~ConnectTerra Confidential/Proprietary~~
~~C:\DOCUME~1\TMuir\LOCALS~1\Temp\EdgeDeviceConfiguration.docZ:\vsswrkdir\doc\design\Compe~~
~~nents\RFTagAware-Sep04\EdgeDeviceConfiguration_impl_design.doc~~

7.2	Downgrade	1514
8	Implementation Overview	1614
8.1	Components and Packages	1614
8.1.1	Component CONFIGAPI	1614
8.1.2	Component CONFIGAXIS	1715
8.1.3	Component CONFIGCLIENT.....	1715
8.1.4	Component CONFIGSERVICE	1715
8.1.5	Component ALEENGINE.....	1715
8.2	config.init Command File.....	1715
8.3	Changes Affecting In-Use Entities	1816
8.3.1	Defining New Entities	1816
8.3.1.1	Effect on Event Cycle processing.....	1816
8.3.1.2	Effect on Programming Cycle processing	1917
8.3.1.3	Effect on Logical Readers	1917
8.3.1.4	Effect on Composite Readers	1917
8.3.1.5	Effect on Telemetry.....	1917
8.3.2	Undefining an Entity	1917
8.3.2.1	Logical Reader Placeholder.....	1917
8.3.2.2	Composite Reader Placeholder.....	2018
8.3.2.3	Filter Placeholder	2018
8.3.2.4	Trigger Placeholder	2018
8.3.3	Redefining an Entity.....	2018
8.3.3.1	Physical Reader	2018
8.3.3.2	Composite Reader	2118
8.3.3.3	Filter	2119
8.3.3.4	Trigger.....	2119

Revision history: latest at the top please			
Version	Author	History	Date(yyyy-mm-dd)
<u>003</u>	<u>smr</u>	Added section on implementation design. Also added "redefine" methods. Change bars relative to 002.	<u>2004-06-29</u>

002	smr	Changes from review. Change bars relative to 001.	2004-06-23
001	smr	Initial draft	2004-06-15

1 Introduction

In RFTagAware V1.1, the edge server is configured by means of an edge .props file, which is a Java-properties formatted file that configures:

- The name of the edge server and the port on which its Soap server listens for connections.
- Internal matters such as thread concurrency and queue depths.
- Location(s) of EPC company prefix index translation tables.
- Other settings for: persistence, telemetry, notification drivers and default notification delivery failure handling.
- Physical readers: driver class, and per-driver configuration items.
- Composite readers.
- Filters for logical readers.
- Triggers: driver class, and per-driver configuration items.

There are major problems with relying on an edge .props file for configuring all of these aspects of the edge server. First, editing this text file isn't viewed as being a user-friendly part of the RFTagAware experience. Second, the file-based configuration is static, only being read by the edge server at start up time. If any changes are made to the configuration file, the edge server must be stopped and restarted, which is not only inconvenient but also unacceptable in an operational setting. Third, there is no easy way to remotely administer the edge .props file.

The purpose of this project is to define an object model and remote API to allow the edge server to be configured, supplanting at least part of the edge .props configuration items. In particular, the aspects of edge server configuration having to do with readers (physical, logical, composite), filters, and triggers will be subjects of this new "ALE Configuration" API.

2 Use Cases

2.1 Administrative Console

The "Admin Console" is currently the primary user-visible manifestation of a RFTagAware deployment. This is where all edge servers are listed and monitored, where ECSpecs and PCSpecs are (someday) authored and deployed, and where telemetry and alerts are marshalled.

One can envision that the Admin Console will be extended to allow the configuration of an edge server to be viewed and manipulated via the ALE Configuration API, for example allowing new readers to be defined and configured, triggers to be created, etc.

The Use Cases in this section concern how the Admin Console might manipulate an edge server's configuration through the ALE Configuration API.

2.1.1 Initial Product Evaluation

In this use case, RFTagAware is being installed by a potential customer who is evaluating the product in a development lab.

The main flow is as follows:

1. The customer uses the RFTagAware installer application to install RFTagAware. An `edge.props` is created by the installer that configures the edge server's name, listening port, etc., and establishes good defaults for various parameters that remain under `edge.props`' control.

The initial edge server configuration includes no readers (except perhaps for the reader simulator).

2. The customer brings up the edge server and the admin console.
3. The customer uses the admin console to define and configure one or more readers. The admin console "knows about" the supported readers, and presents configuration screens tailored for each kind of reader.
 - 3.1. The admin console implementation uses the ALE Configuration API to carry out the various operations on the edge server to define and configure the readers as directed by the customer's interaction with the admin console.
 - 3.2. As each reader is defined and configured through the ALE Configuration API, the edge server persists this state so that the reader definitions survive restarts of the edge server.

Variations:

For step 3, the admin console also allows "other" kinds of reader to be configured, by specification of an driver class and driver-specific properties.

For step 3, the customer can also use the admin console to define and configure composite readers, filters for logical readers, and triggers.

2.1.2 Tagging Station Reconfiguration

In this use case, an existing RFTagAware deployment in an operational setting is updated by a site administrator to reflect a change to a tag reading (or writing) station.

The main flow is as follows:

1. A decision is made to add a reader to a tag reading (or writing) station.
2. The site administrator uses the admin console to view/manipulate the state of the edge server that is associated with the tag reading station to be reconfigured.
3. The site administrator uses the admin console to define and configure the new reader.
4. The site administrator uses the admin console to redefine any existing ECSpecs (or PCSpecs) that should be using the new reader.

Variations:

In step 1, the change could instead be to remove a reader or replace a reader with one of the same or different type.

In step 4, if the change involves replacing a reader but keeping the same logical reader names (such as when upgrading a reader), then it should not be necessary to update any existing ECSpec (or PCSpec) definitions.

In step 4, if instead the ECSpecs (or PCSpecs) had been authored to work in terms of a composite reader instead of an explicit list of logical readers, then the site administrator could instead modify the composite reader to include the new reader, and then all ECSpecs (and PCSpecs) defined in terms of the composite reader would automatically pick up the new reader.

2.2 Discovery System

A "Discovery System" is a piece of network management infrastructure that is responsible for noticing whenever a reader has been added to or taken away from a deployment site's network (how the discovery process works is beyond the scope of this document).

The Use Case in this section concerns how a Discovery System might manipulate an edge server's configuration through the ALE Configuration API.

2.2.1 New Reader Discovered

In this use case, an existing RFTagAware deployment in an operational setting is updated by the Discovery System in response to a new reader being plugged into the network.

The main flow is as follows:

1. A site technician installs a new reader, plugs it in to the site network, and powers up the reader.
2. The Discovery System sees the new reader.
3. The Discovery System connects to the RFTagAware edge server.
 - How does the Discovery System know *which* edge server to contact? For this use case, maybe there's only one edge server, and the Discovery System is statically configured to know where to find this edge server.

The determination of which edge server should be configured for a newly discovered reader could be based on factors such as: subnet affinity; location-based rules, where location is determined by sentinel tags.

4. The Discovery System uses the ALE Configuration API to define and configure the new reader in the edge server.
 - And how does the Discovery System know *how* to configure the reader? What name to give it? How to name its logical reader(s)? What polling interval to define? What filters to configure? What composite(s) to add it to?

Some kind of reader-definition-template mechanism will be useful, where once the appropriate template is identified, filling in the reader configuration information becomes possible. Generally, adding the N+1'th reader to an existing group of N readers will be automatable; configuring a uniquely new reader will generally require human intervention.

2.3 Control Server

A "Control Server" is an RFTagAware component that oversees a network of multiple RFTagAware edge servers.

The Use Cases in this section concern how a Control Server manipulates the edge servers' configuration through the ALE Configuration API.

2.3.1 New Reader Discovered

This is very similar to the use case of 2.2.1, but where the question of *which* edge server to configure to use the new reader is central.

2.3.2 Edge Server Decommissioned

In this use case, an admin console associated with the Control Server is used to take an edge server out of service, migrating all of its workload to another edge server.

The main flow is as follows:

1. A decision is made to permanently decommission an edge server (as opposed to temporarily failing over to a stand-by server).
2. The site administrator uses the Control Server's admin console to designate the to-be-decommissioned edge server ("expiring") and another edge server ("uptaking") to which its work will be transferred.
3. The Control Server enumerates all of the ECSpecs, PCSpecs, reader definitions, subscriptions, EPC caches, etc. of the expiring edge server, and creates corresponding definitions in the uptaking edge server. In the uptaking edge server, the ECSpecs and PCSpecs are defined in their suspended state.
4. The Control Server suspends all of the ECSpecs and PCSpecs in the expiring edge server, and shuts down the expiring edge server. Then, the Control Server un-suspends the corresponding ECSpecs and PCSpecs in the uptaking edge server.

Variations:

In step 2, the site administrator can designate multiple edge servers to take up the decommissioned server's work. The need to distribute the assets of the expiring edge server to the multiple uptaking edge servers highlights the need for the ALE Configuration API to expose the various referential relationships among the various entities being migrated.

In step 4, the three phases (suspend expiring specs, shut down expiring edge server, un-suspend uptaking specs) want to be treated as a distributed transaction.

2.4 Batch-Mode Configuration

In this use case, an enterprise administrator rolls out a configuration change to a large number of deployed edge servers. At the time the changes are to be deployed, the edge servers are not necessarily running or available to be manipulated via the ALE Configuration API.

The main flow is as follows:

1. The enterprise administrator prepares a `config.init` file for each edge server to be configured. Each such file contains one or more commands, in some human-readable syntax, where each such command is equivalent to an ALE Configuration API call.
2. The enterprise administrator copies each `config.init` file to its corresponding edge server's `etc/` directory.
3. The next time each edge server starts, it processes the contents of its `config.init` file.
4. After processing `config.init`, each edge server deletes that file.

Variations:

In step 1, the enterprise administrator prepares the `config.init` files in any of a number of ways:

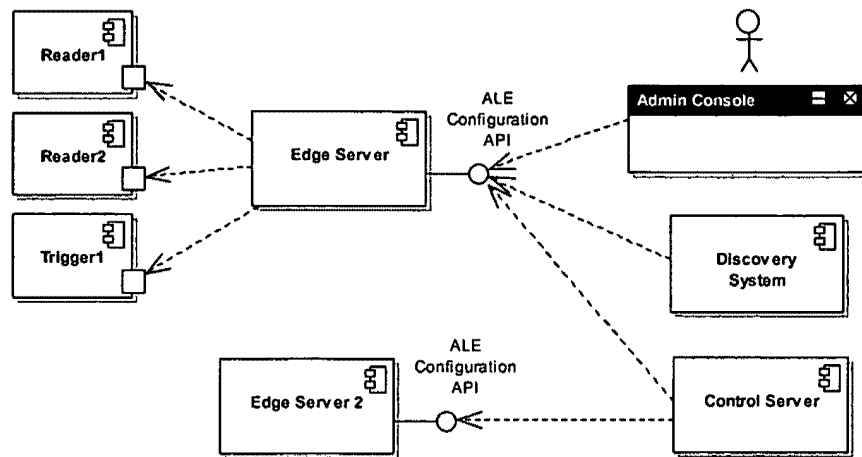
- hand-editing
- using a text-processing tool such as Perl scripts
- using a dedicated tool that presents a task-appropriate user interface, and then emits the file(s)

In step 2, copying the `config.init` files out to the edge servers' `etc/` directories can be accomplished several ways:

- `scp`
- `ftp`, etc.
- any of the above, automated by the same dedicated tool used to create the `config.init` files.

3 Architecture Overview

The following component diagram shows the relationship between the readers/triggers, edge servers, the admin console, discovery system, and control server that were mentioned in the use cases.



Using the ALE Configuration API, an edge server is configured to work with one or more readers and zero or more triggers. The ALE Configuration API is accessible to the admin console, so that an administrator may view and manipulate the reader/trigger definitions in an edge server.

The ALE Configuration API is also available to a discovery system that is responsible for noting the comings and goings of readers on the site network. The discovery system uses the ALE Configuration API to configure an edge server to use a discovered reader.

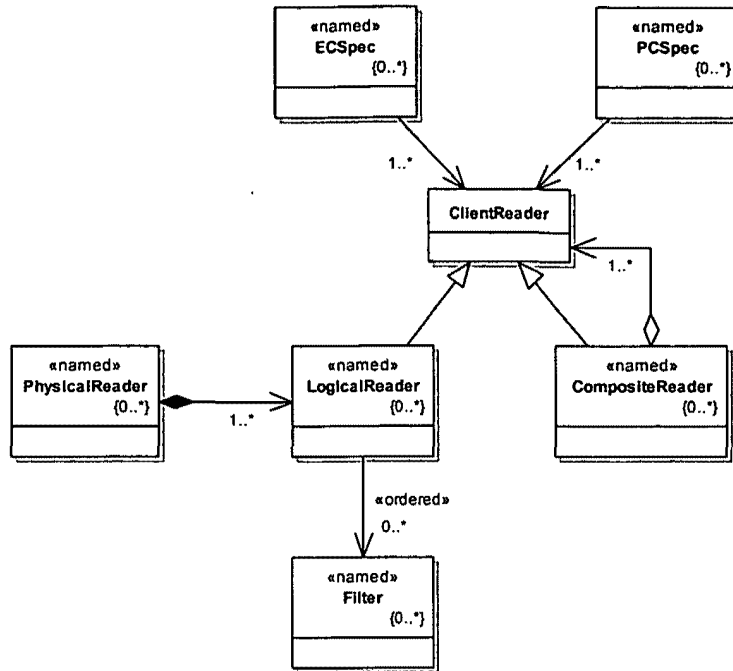
A control server uses the ALE Configuration API to manipulate the configuration of multiple edge servers.

4 Configuration Model

The ALE Configuration API provides methods that expose and manipulate various aspects of an edge server's configuration. This API may be thought of as a public interface to an underlying object model.

We present this model as an entity-relationship diagram in this section. **Important:** this diagram is specifically for illustrating the ALE Configuration API, and does not reflect the underlying ALE implementation object model, despite the presence of similarly named entities such as "PhysicalReader", "LogicalReader", etc.

In the interest of simplicity, triggers are not shown.



The «named» stereotype indicates that the entity is a collection of instances, each of which has a distinct name (akin to the primary key in a relational model). For example, an edge server can have any number of ECSpec instances, each of which has a unique name. In the ALE Configuration API, such entities are always reference by their name. When one entity references another, it does so via the name of the referenced entity (like a foreign key).

ECSpecs and PCSpecs are created via the ALE and ALEPC APIs, respectively. Each ECSpec (or PCSpec) refers to one or more “client” readers, each of which is either a logical reader or a composite reader.

A composite reader is created by use of the ALE Configuration API, and is simply a collection of logical readers and/or composite readers.

A physical reader entity is created by use of the ALE Configuration API, by specifying a driver implementation class and a set of properties, interpreted by the driver, that configure the physical reader instance. Each physical reader defines one or more logical readers.

Logical readers are not created directly by the ALE Configuration API, but instead are owned by a physical reader.

Filters are created by use of the ALE Configuration API. Each logical reader may have an ordered sequence of filters configured for use during read cycle processing.

4.1 Referential Integrity

As can be seen in the entity-relationship diagram above, there are many referencing relationships among the objects manipulated by the ALE Configuration API. This raises the question of referential integrity:

- May a referencing entity be created if the referenced entity does not (yet) exist? For example, may an ECSpec be defined if it refers to a non-existent logical reader?
- May a referenced entity be deleted? For example, may a filter that is part of a logical reader’s filter chain be deleted?

Elsewhere, the various ALE APIs maintain referential integrity. For example, an ECSpec definition is rejected if it references a non-existent logical reader. As another example, an EPC cache may not be undefined if any PCSpec references that EPC cache.

Strongly enforcing referential integrity could make certain configuration manipulations very difficult. For example, suppose a physical reader with two logical readers (antennas) is to be replaced with two physical readers, each with a single logical reader. The replacement logical readers will have the same names as the original logical readers, so that existing ECSpecs (or PCSpecs) don't need to be modified. In order to carry out this replacement, while preserving referential integrity, it would be necessary to treat the individual operations (undefine the old physical reader, define the first new physical reader, define the second new physical reader) as belonging to a single transaction.

Instead, a simpler approach is to permit referential integrity to be (temporarily) broken, as when the old physical reader has been undefined but the new physical readers has not yet been defined.

5 The ALE Configuration API

Like the other ALE APIs, the ALE Configuration API is organized as a set of top-level methods, helper objects, and exceptions. In this section we present the ALE Configuration API, with the top-level methods organized and grouped according to function.

The API is presented using pseudo-Java syntax. Fundamentally, the API will be exposed as a SOAP-based remote invocation wire protocol, with a Java client binding provided to encapsulate the SOAP layer.

In addition to the exceptions listed below, all methods also throw `EngineException` and `RemoteException`.

5.1 Physical Reader Methods

List `getAllPhysicalReaderNames()`

```
void definePhysicalReader(String physicalReaderName, EntitySpec spec)
    throws DuplicatePhysicalReaderNameException,
           DuplicateLogicalReaderNameException,
           EntitySpecValidationException,
           NoSuchDriverException
```

```
void undefinePhysicalReader(String physicalReaderName)
    throws NoSuchPhysicalReaderException
```

```
void undefinePhysicalReader(String physicalReaderName, ReferentialIntegrity riControl)
    throws NoSuchPhysicalReaderException,
           InUseException
```

```
void redefinePhysicalReader(String physicalReaderName, EntitySpec spec)
    throws NoSuchPhysicalReaderException,
           DuplicateLogicalReaderNameException,
           EntitySpecValidationException,
           NoSuchDriverException
```

```

void redefinePhysicalReader(String physicalReaderName, EntitySpec spec,
                           ReferentialIntegrity riControl)
    throws NoSuchPhysicalReaderException,
           DuplicateLogicalReaderNameException,
           EntitySpecValidationException,
           NoSuchDriverException,
           InUseException

```

```
EntitySpec getPhysicalReaderSpec(String physicalReaderName)
```

```

List getLogicalReaderNames(String physicalReaderName)
    throws NoSuchPhysicalReaderException

```

```
List getPhysicalReaderDriverNames()
```

```

// TBD: methods to list ECSpecs/PCSpecs that depend on a given physical reader?
//      Not here-- such methods should be part of the ALE and ALEPC
//      APIs.

```

5.1.1 Helper objects

The following helper object is used by all of the methods that define entity instances and return information about them:

```

class EntitySpec
{
    String driverName;
    Map config;
}

```

The `driverName` identifies the driver class responsible for this entity. For now, this is just the fully-qualified Java class name.

The `config` is a set of name/value pairs, and is interpreted by the driver class. Each driver class should specify what names it looks for in the map and how it treats their values.

The following enum is used by methods that could possibly break inter-entity references:

```

enum ReferentialIntegrity
{
    ENFORCE,    // throw InUseException rather than break reference
    RELAX       // allow reference to dangle
    // UPDATE   // remove the reference
}

```

If `ENFORCE` is specified, a method such as `undefinePhysicalReader()` will throw `InUseException` if the operation would cause a reference to be broken (such as an `ECSpec` that references a logical reader defined by the to-be-undefined physical reader).

If `RELAX` is specified, then enforcement of referential integrity constraints is not performed. This introduces the possibility of dangling references. For example, if a physical reader is undefined, and an

ECSpec lists a logical reader which no longer exists as a result of undefining the physical reader, then we say that the ECSpec has a dangling reference to a non-existent logical reader. In all such cases, we need to carefully specify the resulting behavior. In the present example, the ECSpec might operate as if the missing logical reader exists but always produces read errors (say).

UPDATE is another possibility that we omit for now. This would maintain referential integrity by modifying a referencing entity so that it no longer references the deleted entity. For example, ECSpecs and PCSpecs would be altered to remove an undefined reader from their list of logical readers.

- But what if an ECSpec's final logical reader is removed by an UPDATE-mode operation? We would either have to change the rules to permit an ECSpec to have no logical readers (currently invalid), or we'd have to undefine the ECSpec altogether to maintain referential integrity. It is considerations such as this, coupled with a lack of crisp customer requirements, that lead us to defer the UPDATE mode.

5.2 Logical Reader Methods

```
List getAllLogicalReaderNames()
```

```
boolean supportsReading(String logicalReaderName)  
    throws NoSuchLogicalReaderException
```

```
boolean supportsWriting(String logicalReaderName)  
    throws NoSuchLogicalReaderException
```

```
List getFilterChain(String logicalReaderName)  
    throws NoSuchLogicalReaderException
```

```
void setFilterChain(String logicalReaderName, List filterNames)  
    throws NoSuchLogicalReaderException,  
           NoSuchFilterException
```

```
void prependFilterToChain(String logicalReaderName, String filterName)  
    throws NoSuchLogicalReaderException,  
           NoSuchFilterException
```

```
void appendFilterToChain(String logicalReaderName, String filterName)  
    throws NoSuchLogicalReaderException,  
           NoSuchFilterException
```

```
void removeFilterFromChain(String logicalReaderName, String filterName)  
    throws NoSuchLogicalReaderException,  
           NoSuchFilterException
```

```
String getPhysicalReaderName(String logicalReaderName)  
    throws NoSuchLogicalReaderException
```

```
// This method should be part of ALE: getECSpecsForReader
//List getECSpecNamesUsingReader(String clientReaderName)
//    throws NoSuchClientReaderException

// This method should be part of ALEPC: getPCSpecsForReader
//List getPCSpecNamesUsingReader(String clientReaderName)
//    throws NoSuchClientReaderException

List getCompositeReaderNamesUsingReader(String clientReaderName)
    throws NoSuchClientReaderException
```

5.3 Filter Methods

```
List getAllFilterNames()

void defineFilter(String filterName, EntitySpec spec)
    throws DuplicateFilterNameException,
           EntitySpecValidationException,
           NoSuchDriverException

void undefineFilter(String filterName)
    throws NoSuchFilterException

void undefineFilter(String filterName, ReferentialIntegrity riControl)
    throws NoSuchFilterException,
           InUseException

void redefineFilter(String filterName, EntitySpec spec)
    throws NoSuchFilterException,
    EntitySpecValidationException,
    NoSuchDriverException

void redefineFilter(String filterName, EntitySpec spec, ReferentialIntegrity riControl)
    throws NoSuchFilterException,
    EntitySpecValidationException,
    NoSuchDriverException,
    InUseException

EntitySpec getFilterSpec(String filterName)

List getLogicalReaderNamesUsingFilter(String filterName)
    throws NoSuchFilterException
```

```
List getFilterDriverNames()
```

5.4 Composite Reader Methods

```
List getAllCompositeReaderNames()
```

```
void defineCompositeReader(String compositeReaderName,  
                           List clientReaderNames, boolean reportFailedMembers)  
    throws DuplicateCompositeReaderNameException,  
           NoSuchClientReaderException,  
           CyclicDefinitionException
```

```
void undefineCompositeReader(String compositeReaderName)  
    throws NoSuchCompositeReaderException
```

```
void undefineCompositeReader(String compositeReaderName, ReferentialIntegrity riControl)  
    throws NoSuchCompositeReaderException,  
           InUseException
```

```
void redefineCompositeReader(String compositeReaderName,  
                             List clientReaderNames, boolean reportFailedMembers)  
    throws NoSuchCompositeReaderException,  
           NoSuchClientReaderException,  
           CyclicDefinitionException
```

```
void redefineCompositeReader(String compositeReaderName,  
                             List clientReaderNames, boolean reportFailedMembers,  
                             ReferentialIntegrity riControl)  
    throws NoSuchCompositeReaderException,  
           NoSuchClientReaderException,  
           CyclicDefinitionException,  
           InUseException
```

```
void addReaderToComposite(String compositeReaderName, String clientReaderName)  
    throws NoSuchCompositeReaderException,  
           CyclicDefinitionException
```

```
void addReadersToComposite(String compositeReaderName, List clientReaderNames)  
    throws NoSuchCompositeReaderException,  
           CyclicDefinitionException
```

```
void removeReaderFromComposite(String compositeReaderName, String clientReaderName)  
    throws NoSuchCompositeReaderException
```

```
boolean getReportFailedMembers(String compositeReaderName)
    throws NoSuchCompositeReaderException
```

```
List getClientReaderNames(String compositeReaderName)
    throws NoSuchCompositeReaderException
```

5.5 Trigger Methods

```
List getAllTriggerNames()
```

```
void defineTrigger(String triggerName, EntitySpec spec)
    throws DuplicateTriggerNameException,
           EntitySpecValidationException,
           NoSuchDriverException
```

```
void undefineTrigger(String triggerName)
    throws NoSuchTriggerException
```

```
void undefineTrigger(String triggerName, ReferentialIntegrity riControl)
    throws NoSuchTriggerException,
           InUseException
```

```
void redefineTrigger(String triggerName, EntitySpec spec)
    throws NoSuchTriggerException,
    EntitySpecValidationException,
    NoSuchDriverException
```

```
void redefineTrigger(String triggerName, EntitySpec spec, ReferentialIntegrity riControl)
    throws NoSuchTriggerException,
    EntitySpecValidationException,
    NoSuchDriverException,
    InUseException
```

```
EntitySpec getTriggerSpec(String triggerName)
```

```
// This method should be part of ALE: getECSpecsForTrigger
//List getECSpecNamesUsingTrigger(String triggerName)
//    throws NoSuchTriggerException
```

```
// This method should be part of ALEPC: getPCSpecsForTrigger
//List getPCSpecNamesUsingTrigger(String triggerName)
//    throws NoSuchTriggerException
```

Version of 29 June 2004 at 1154 hours ~~29 June 2004 at 1044 hours~~ *ConnecTerra Confidential/Proprietary*
C:\DOCUME~1\TMuir\LOCALS~1\Temp\EdgeDeviceConfiguration.doc ~~Z:\vsswrk\dir\doc\design\Compo~~
nents\RFTagAware-Sep04\EdgeDeviceConfiguration_impl_design.doc

6 Persistent Store

The definitions for readers, filters, triggers, etc. are all persisted in the `var/edgestate` repository along with ECSpecs, subscriptions, EPC caches, etc.

It is likely that the `ClearEdgeServerState` utility installed as part of `RFTagAware` needs to grow options to control *which* persistent state is cleared.

7 Back/Forward Compatibility

In `RFTagAware V1.1` (and `V1.0`), `edge.props` is the sole repository for edge server configuration information for physical readers, filters, triggers, etc. In `R3`, much of this information no longer resides in `edge.props` but is instead managed via the ALE Configuration API and is persisted in the edge server's `var/edgestate` repository.

In this section we examine how a `V1.1` installation is upgraded to `R3`, and also explore how we might support export of an `R3` installation's configuration back to `V1.1` format.

7.1 Upgrade

In the upgrade scenario, an existing `V1.1` deployment is upgraded to `R3`. The goal is that any readers, triggers, filters, etc. defined in the `V1.1` deployment should be carried over to the `R3`-upgraded deployment and should thereafter be fully manipulable via the ALE Configuration API. Essentially, we need to remove the relevant configuration items from `edge.props` and put them into the persistent store.

1. Initial condition: an `RFTagAware V1.1` deployment exists. Its `edge.props` file describes the configuration of its readers, triggers, filters, etc. Whenever the `V1.1` edge server is started, it reads the `edge.props` file and configures itself accordingly.
 2. The `V1.1` edge server is shut down.
 3. The `R3` installer is run in "upgrade" mode over the existing `V1.1` deployment. All trampolines, jars, etc. are replaced with the `R3` versions. The existing `edge.props` file is modified to remove any definitions for readers, triggers, filters, etc., and to append any new properties introduced in `R3`. In addition, a `config.init` file is created containing commands to define the readers, triggers, filters, etc. that were previously defined in the `edge.props` file.
 4. The `R3` edge server is started for the first time after the upgrade install.
 5. The `R3` edge server processes the contents of the `var/edgestate` repository, restoring ECSpecs, subscriptions, etc. Because this is the first time the edge server has run after being upgraded to `R3`, the persistent repository does not yet hold any definitions for readers, filters, triggers, etc.
 6. The `R3` edge server looks for and finds the `config.init` file and processes its contents, defining readers, triggers, filters, etc. as directed by commands therein.
 7. After processing the `config.init` file, the `R3` edge server deletes it.
- The contents of `config.init` should have idempotent semantics, so that if the edge server should crash while processing it, then the next time the edge server processes its contents it will be OK that some of the commands have already been executed.

7.2 Downgrade

- This section is retained for reference only; current thinking is that we will not be supporting downgrade of the sort described here.

In this scenario, the customer decides to down-version an R3 deployment back to V1.1.6, the last stable release. Let's just say there's something about R3 that isn't cutting the mustard for this customer, and let's assume that we provide installer support for down-versioning an existing installation.

Suppose the customer has invested some effort in configuring the edge server while running R3, and wants to carry this configuration to the down-versioned V1.1.6 deployment. The R3 `edge.props` doesn't have many of the configuration items, because these now reside in the R3-specific persistent repository.

Prior to running the V1.1.6 installer in the "upgrade" mode over the R3 installation, the customer invokes an "export to V1.1 `edge.props`" utility command. This does the following:

1. Starts the R3 edge server, which causes it to process all of its persisted definitions for readers, filters, triggers, etc.
2. Copies the R3 `edge.props` file to `edge.props.export.v1.1`.
3. Appends configuration properties for all currently defined readers, filters, triggers, etc. to the `edge.props.export.v1.1` file.
4. Shuts down the R3 edge server.

At this point the customer can run the V1.1.6 installer to "upgrade" the R3 installation back to V1.1.6, and then move or copy the `edge.props.export.v1.1` file to `edge.props`.

- Again, this depends on the ability to write out an `edge.props` file using our property-based configuration abstractions.

8 Implementation Overview

This section provides a roadmap for the implementation of the ALE Configuration API. The goal is not to cover every aspect in excruciating detail, but rather to broadly map out the territory to be covered by the implementation, and most importantly, to identify any parts of the ALE implementation that need to be altered to accommodate the ALE Configuration API.

8.1 Components and Packages

All config-related classes will be in package `com.connecterra.ale.config` or sub-packages thereof. For each VCC component described below, we identify the Java packages to which the component contributes, and explain the functionality being provided.

We are patterning the componentization and packaging of the ALE Configuration API after that of the ALE Telemetry API.

8.1.1 Component CONFIGAPI

This component provides the XML schema (.xsd file) that defines the XML formatting for the datatypes used by the ALE Configuration API. These are the `EntitySpec` struct and the `ReferentialIntegrity` enum.

This component also provides the WSDL for the Soap invocation mechanism of the ALE Configuration API methods.

This component contributes to the following Java packages:

`com.connecterra.ale.config.api`: Defines interfaces and exception classes for the ALE Configuration API. The main interfaces are `Config.java`, defining the various top-level methods of the configuration API; `EntitySpec.java`; and `ReferentialIntegrity.java` (type-safe enum class).

com.connectterra.ale.config.encoding: Defines interfaces and exception classes related to XML serialization and deserialization of API-related objects (just `EntitySpec`).

8.1.2 Component CONFIGAXIS

This component hosts the Axis-generated XML serialization code together with ConnecTerra-authored adapter classes and implementations of the API serialization interfaces. This component contributes to the following Java packages:

com.connectterra.ale.config.axis: provides `AxisConfigFactory.java`, an implementation of the `ConfigFactory` interface of the API.

com.connectterra.ale.config.axis.adapters: provides adapter classes that implement the API data type interfaces in terms of Axis-provided implementations.

com.connectterra.ale.config.axis.encoding: provides implementations of the API interfaces for XML serialization, implemented in terms of the corresponding adapter classes. Also provides refined implementations of Axis serialization and deserialization (to tweak things such as namespace treatment).

8.1.3 Component CONFIGCLIENT

This component hosts the Java-over-Soap client API binding for the ALE Configuration API. This component contributes to the following Java package:

com.connectterra.ale.config.client: provides the `AxisConfigClient.java` class, which implements the `Config` API by proxying the methods over Axis/Soap to the ALE engine implementation.

8.1.4 Component CONFIGSERVICE

This component hosts the Axis-generated implementation of the server side of the Soap implementation as described in the WSDL. This component contributes to the following Java package:

com.connectterra.ale.config.axis.service: Contains WSDL-described generated code, and the ConnecTerra-authored `ConfigServiceBindingImpl.java` class which receives Soap-proxied client API calls and delegates them to the ALE engine implementation of the configuration API.

8.1.5 Component ALEENGINE

This component provides the ALE engine implementation of the `Config` API. This component contributes to the following Java package:

com.connectterra.ale.config: Contains implementation classes, including `EngineConfigImpl.java`, which implements the `Config` API interface.

In addition, existing ALE engine logic related to ECSpecs and PCSpecs will be affected by the ALE Configuration API, because configuration changes will now be possible in a “live” edge server, affecting ongoing processing activities in the ALE engine logic. See 8.3 below.

8.2 config.init Command File

A `config.init` file may be placed in the edge server's `etc/` directory, as part of:

- batch-mode configuration update of edge server
- upgrade from V1.1 installation's `edge.props`

In either case, the `config.init` file consists of a series of human-readable commands or directives, which cause one or more readers, filters, triggers, etc. to be configured. After the edge server processes the contents of `config.init`, it removes the file. And just in case an edge server is re-started while it is processing this file, the commands or directives therein should be interpreted with idempotent semantics.

How are the contents of `config.init` formatted? At least a few choices are:

1. A scripting language syntax, such as Jython or Groovy, executing in a context where the ALE Configuration API methods are available.
2. Serialized remote invocations, as defined by the WSDL for the ALE Configuration API.
3. `edge.props` style Java properties.

Pros and cons are copious and mostly obvious. I'm not going to try to decide this here; I'd like to hear some discussion on the topic and hopefully a consensus will emerge.

8.3 Changes Affecting In-Use Entities

In V1.1, an edge server establishes its configuration at startup time, and its configuration is unchanging thereafter. In R3, an edge server also establishes its configuration at startup time, but thereafter its configuration may be changed at any time by invoking operations of the ALE Configuration API.

This section explores what effect dynamic changes to an edge server's configuration have on that edge server's ongoing processing activities.

The general strategy is twofold. First, we notify affected entities of a change to something they refer to. So for example, if a filter is changed, then affected logical readers are notified. If a reader is changed, then affected event cycles and/or programming cycles are notified.

Second, if an entity is undefined in RELAX mode, then other entities that refer to it replace their reference with a reference to a "placeholder" object of the same name, where the placeholder object provides an appropriate neutral behavior (for example, a placeholder filter's behavior would be probably be pass-through). Later, if an entity is defined again to the same name, then such placeholder references will be replaced (likely in the change-notification logic) with a reference to the newly defined entity of the same name as the previously undefined entity.

8.3.1 Defining New Entities

Normally, defining a new physical reader (and so one or more logical readers), filter, composite reader, or trigger would not affect any existing ECSpecs or PCSpecs, because the ECSpecs and PCSpecs are already bound to existing entities and don't care about any additional readers etc.

However, a RELAX-mode change to configuration could have left an ECSpec or PCSpec with a "dangling" reference to a reader or trigger (or a logical reader could have a dangling reference to a filter, etc.), in which case defining a "new" entity could really be re-establishing a previous referential relationship.

Thus, even new entity definitions need to be treated as potentially affecting ongoing processing activity.

8.3.1.1 Effect on Event Cycle processing

All event cycles are told (by queuing a change notification event) whenever:

- a new client reader (logical or composite) is defined. The change notification is decorated with the reader's name and a reference to the reader instance.
- a new trigger is defined. The change notification is decorated with the trigger's name and a reference to the trigger instance.

- a composite reader is updated due to a new logical reader being defined, with the effect of replacing a placeholder reference in the composite reader (ah, so the pattern is that whenever an entity is affected by a change, notification bubbles up through all levels of referrers).

When the event cycle processes the queued change notification, it checks its client reader list (or trigger specs) to see if any placeholder references thereon are to be replaced with a reference to a newly defined reader (or trigger). If so, the event cycle re-performs any of its validations or computations related to its reader list (or trigger specs).

8.3.1.2 Effect on Programming Cycle processing

Pretty much the same as with event cycles.

8.3.1.3 Effect on Logical Readers

Whenever a new filter is defined, the reader manager notifies all logical readers (via a new method to be defined in the LogicalReader interface), identifying the name of the new filter and providing the filter definition (factory, in the parlance of our implementation). Each logical reader determines whether it has any placeholder references that should be re-bound to an instance of the newly defined filter, and if so, creates an instance of the new filter and replaces the placeholder with it, and commences using the new filter in the logical reader's filter chain.

8.3.1.4 Effect on Composite Readers

If a new logical reader is defined (indirectly, by (re)defining a physical reader), then the CompositeReaderManager is notified so that it can update its tables, replacing placeholder references as appropriate. In addition, all event/programming cycles are notified of the change, as mentioned above.

8.3.1.5 Effect on Telemetry

Whenever a new entity is defined, the telemetry subsystem needs to be updated.

➤ Won't this "just work" because whenever an entity is defined it registers itself with telemetry?

8.3.2 Undefined an Entity

Whenever an entity is undefined in RELAX mode, all entities that reference it are notified, so that the references to the undefined entity can be replaced with "placeholder" references. Placeholder references are used so that if the just-undefined entity is later re-defined, the redefined entity will take the place of the previously undefined original in all referencing entities.

When an entity is undefined, the telemetry subsystem is notified. Any TelemetrySpecs that reference the just-undefined entity will need to replace the entity with a placeholder, and perhaps also generate a report to subscribers notifying them that the device has been undefined.

➤ The concept of de-registering an entity from telemetry is new, and so there is likely to be work done in the telemetry subsystem.

8.3.2.1 Logical Reader Placeholder

When a physical reader is undefined in RELAX mode, resulting in one or more logical readers being undefined, then any referencing ECSpecs or PCSpecs (and composite readers) replace their reference to the just-undefined logical reader with a reference to a placeholder logical reader.

A placeholder logical reader is not visible through the ALE Configuration API methods, and so for example placeholder logical readers are not part of the getAllLogicalReaderNames() collection. But an ECSpec (or PCSpec, or composite) that references a placeholder logical reader will include the logical reader name among its list of client readers.

When an active event cycle (or programming cycle) attempts to use a placeholder logical reader, the placeholder logical reader will complete its operation with an “undefined logical reader” failure status, and the event cycle (or programming cycle) will behave as it normally does when a logical reader fails.

8.3.2.2 Composite Reader Placeholder

When a composite reader is undefined in RELAX mode, then any referencing ECSpecs or PCSpecs (or other composite readers) replace their reference to the just-undefined composite reader with a reference to a placeholder composite reader.

A placeholder composite reader is not visible through the ALE Configuration API methods, and so for example placeholder composite readers are not part of the `getAllCompositeReaderNames()` collection. But an ECSpec (or PCSpec, or another composite) that references a placeholder composite reader will include the composite reader name among its list of client readers.

A placeholder composite reader simply behaves as a composite reader with no entries.

8.3.2.3 Filter Placeholder

When a filter is undefined in RELAX mode, then any referencing logical readers replace their reference to the just-undefined filter with a reference to a placeholder filter.

A placeholder filter is not visible through the ALE Configuration API methods, and so for example placeholder filters are not part of the `getAllFilterNames()` collection. But a logical reader that references a placeholder filter will include the filter name in its chain of filters.

A placeholder filter behaves as a pass-through (identity) function, leaving the `EPCBuffer` contents unchanged.

8.3.2.4 Trigger Placeholder

When a trigger is undefined in RELAX mode, then any referencing ECSpecs or PCSpecs replace their reference to the just-undefined trigger with a reference to a placeholder trigger.

A placeholder trigger is not visible through the ALE Configuration API methods, and so for example placeholder triggers are not part of the `getAllTriggerNames()` collection. But a ECSpec (or PCSpec) that references a placeholder trigger will continue to include the trigger name in its specification.

A placeholder trigger behaves as a “dead” trigger, one that can be subscribed/unsubscribed, but never actually fires.

8.3.3 Redefining an Entity

8.3.3.1 Physical Reader

When a physical reader is redefined, this can cause new logical readers to be defined, or existing logical readers to be undefined (because the replacement physical reader’s spec can define a different set of logical readers as the original physical reader spec). Handling of newly defined (or undefined) logical readers is handled as described above.

In addition, the read/write capability of an existing logical reader may be changed as a result of redefining the physical reader that defines the logical reader. So affected ECSpecs and PCSpecs must re-evaluate the usability of their logical readers whenever such a change occurs.

- What if logical reader is redefined (by redefining a physical reader)—is the original logical reader’s filter chain retained? If ‘yes’, then do we also do so for a logical reader that is undefined and later re-defined?

8.3.3.2 Composite Reader

When a composite reader is redefined, affected ECSpecs and PCSpecs are notified so that they can re-expand the composite and continue working with the resulting list of logical readers.

8.3.3.3 Filter

When a filter is redefined, affected logical readers are notified so that they can replace their corresponding filter chain entries with a reference to a replacement filter instance.

8.3.3.4 Trigger

When a trigger is redefined, affected ECSpecs and PCSpecs are notified so that they can replace their reference to the previous trigger with a reference to the new trigger.



ConnectTerra EPC Provisioning Service Design

Version of 2005-04-18 at 13:26

This document is Company Confidential

Table of Contents

ConnectTerra EPC Provisioning Service Design	1
1 Introduction	3
2 Key Terminology	3
3 Feature Overview	3
4 Use Cases	4
5 Architectural Overview	4
6 Design Overview	5
6.1 EPC Provisioning Service	6
6.2 RFTA Integration Component	7
6.2.1 VCC Components	7
6.3 Admin Web Client	7
7 Design Detail	8
7.1 EPC Provisioning Service	8
7.1.1 Package com.connectterra.epcps.api– key classes UML	8
7.1.2 Package com.connectterra.epcps.api.policyConfigurations– key classes UML	9
7.1.3 Package com.connectterra.epcps.engine– key classes UML	10
7.1.4 Dependencies on ALEEPC component	11
7.1.5 Dependencies on ALEAPI component	11
7.1.6 Schema Design	11
7.2 RFTA Integration Component	14
7.3 Admin Web Client	14
8 External Changes	14
8.1 Specific add/update to Java API	14
8.2 Specific add/update to XML API	14
8.3 Specific add/update to Web Services API	14

8.4	Specific add/update to Configuration	14
8.5	Specific add/update to Scripts	14
8.6	Specific add/update to GUI	15
8.7	Specific add/update to Command Line Interface	15
8.8	Specific add/update to Protocol.....	15
8.9	Specific add/update to Syntax	15
8.10	Specific add/update to Database/Persistence.....	15
8.11	Specific add/update to Installer	15
8.12	Specific add/update to Samples.....	15
8.13	Specific add/update to jar files and other libraries	15
8.14	Security considerations.....	15
8.15	Compatibility Considerations.....	15
8.16	Performance Considerations.....	15
8.17	Testing considerations.....	15
8.18	Deployment considerations	15

Revision history: latest at the top please			
Version	Author	History	Date(yyyy-mm-dd)
001	Manju James	Initial version	2004-04-12

1 Introduction

This document outlines the design for EPC Provisioning Service in the ConnecTerra Enterprise Server 1.0 product.

2 Key Terminology

This section lists key terms and definitions as utilized in the design:

EPC Pool: A named list of EPCs maintained in the provisioning system. EPCs may be checked out or checked in to the pool by outside applications.

Identity Type: As used in EPC Tag Data Standards document. Examples of some identity types used in this design doc are GTIN, SSCC, DoD Construct.

Pool Name, Cache Name: This design assumes that the name of a cache in the Edge Server is the same as the pool name in the provisioning service. Therefore the terms pool name and cache name are used interchangeably in this document.

3 Feature Overview

Tag writing systems need a central authority from which they can receive unique EPCs. The EPC Provisioning Service is designed to satisfy this need.

In addition to providing a service to give out unique EPCs, a design goal is also to make it easy for administrators to define and maintain pools. Administrators will be able to specify parameters based on which naming of pools and EPC allocations to pool can be done automatically

The service must be generic enough to work with different ConnecTerra products as well as third party products. However, we will provide specific sub-components to integrate smoothly with ConnecTerra products. For example, we will provide sub-components that will help integrate with RFTagAware Edge Server and ConnecTerra's local workflow components.

4 Use Cases

1. Local workflow checks out EPCs from the provisioning service by specifying the parameters for the checkout. An example of the parameters specified is (GTIN=10037000308048, TAG_LENGTH=64, LOCATION_ID=tag@ship-station-1). The provisioning service will check to see if the set of parameters has access to a pool and then returns EPCs to the local workflow. In this process, as required, the provisioning service may create a new pool and allocate EPCs to it. The EPCs given out will be marked as 'allocated' by the provisioning service.
2. Provisioning system receives a tag-cache-low report from the RFTagAware Edge Server. The provisioning service will checkout EPCs from the correct pool and replenish the cache in the Edge Server.
3. Administrator finds out that a product or station must no longer be allowed to get EPCs from the provisioning service. Administrator makes changes to the provisioning service and requests with those parameters will be denied access.
4. An edge server tag cache is reset and the EPCs left in the cache is released back to the provisioning service. The provisioning service removes these from allocated list and puts them back in the available list.
5. Administrator finds out that a tagging station needs to switch from using 64 bit tags to 96 bit tags. Administrator sets up pool naming policy and pool population policy to enable that.

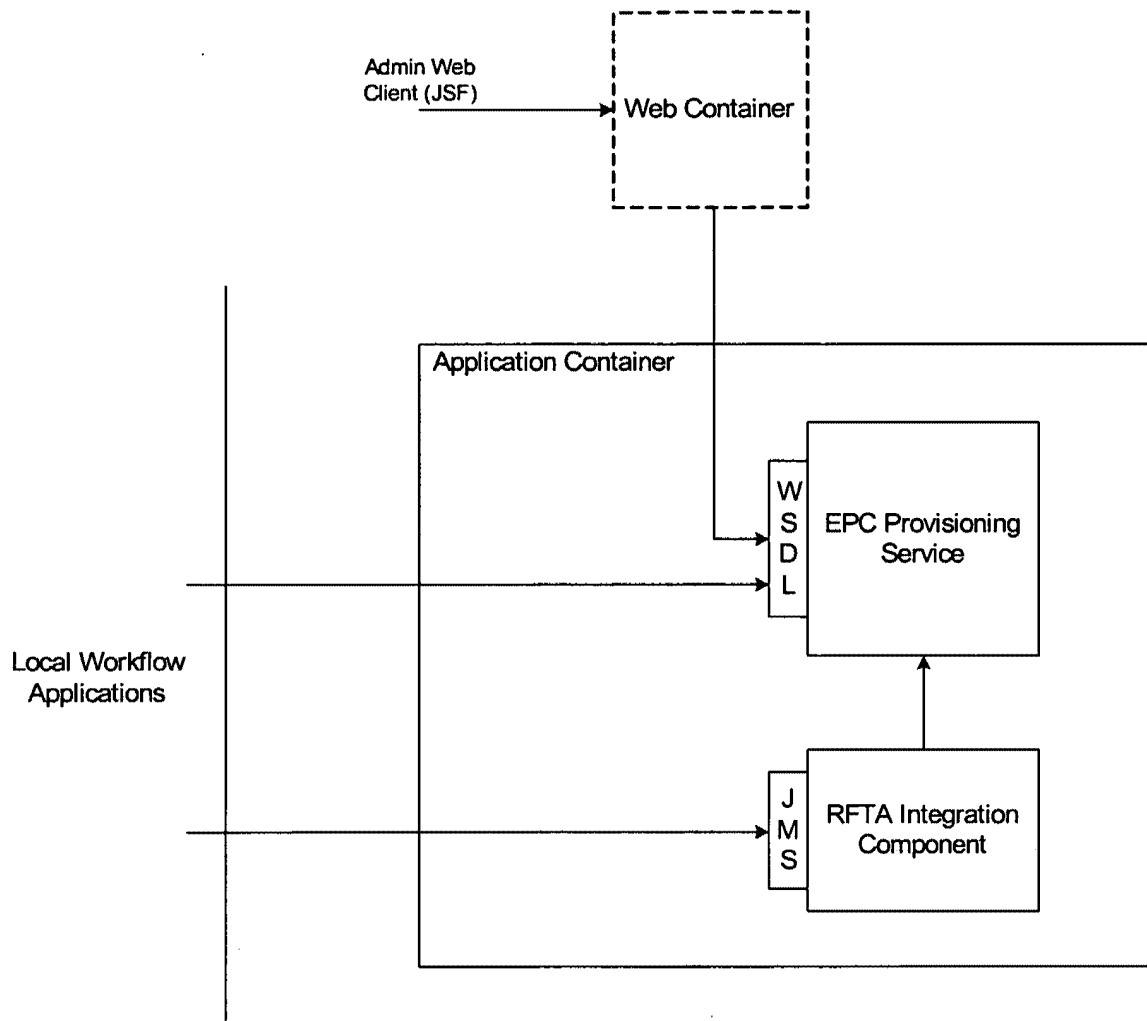
5 Architectural Overview

The provisioning service system will be composed of 2 main components:

1. EPC Provisioning Service: This is a web service that will provide generic provisioning related operations like administering EPC pools, checkout or check-in from EPC pools etc.
2. RFTA Integration Component: This component is used to integrate between RFTagAware Edge Server and the provisioning service. For release 1.0, responsibility of this component is to accept cache-low notifications from the edge server and replenish the cache. For 1.0, this component will support only a JMS interface.

There will be a web based GUI for administering the provisioning system. This GUI will be developed using JSF technology and communication with the provisioning service will be through the web service interface.

Local workflows may communicate with the EPC Provisioning Service or RFTA Integration Component as needed.



6 Design Overview

Figure 1 below shows interaction between local workflow components, edge server and EPC provisioning system when the local workflow encounters a new item for which it may need to define a cache at the edge server.

Figure 2 shows the interactions between the edge server, RFTA Integration Component and EPC Provisioning Service when the edge server detects that a cache's size has reached the low threshold value.

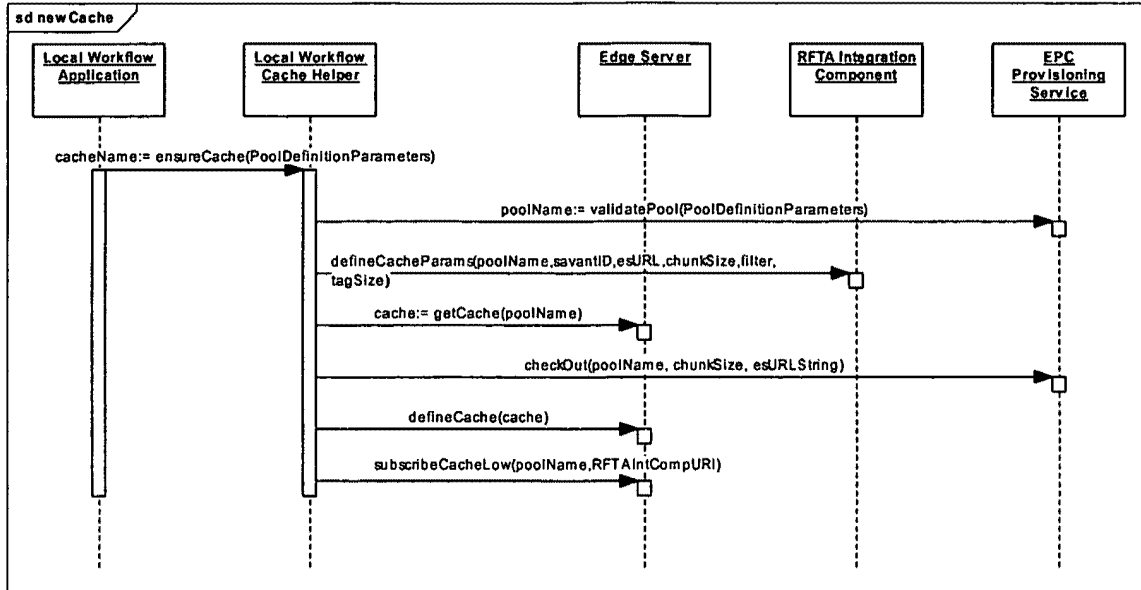


Figure 1

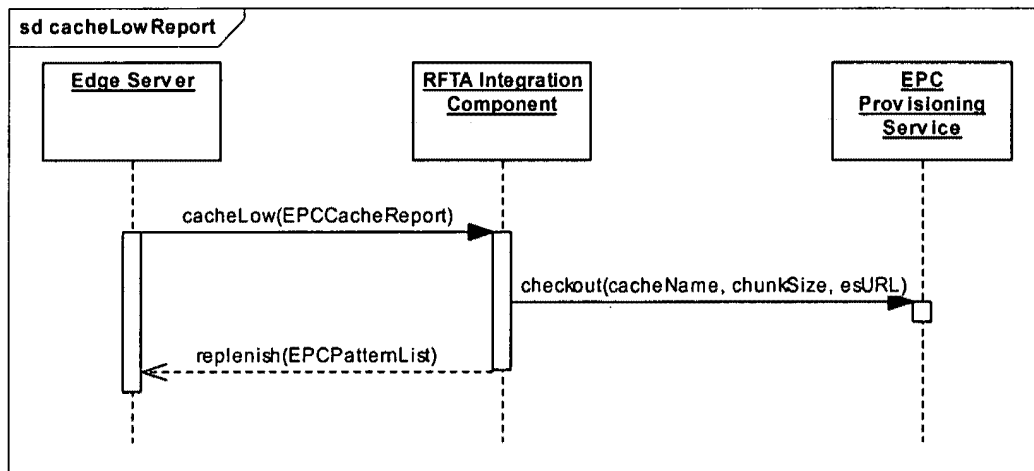


Figure 2

6.1 EPC Provisioning Service

The design overview diagram for the web service will be similar to other web services in Enterprise Server (see [EPCIS Design.doc](#) for details).

VCC components for this are:

1. EPCPS_API: This will contain the public facing interfaces (java, wsdl and xsd)
2. EPCPS_Axis: Axis generated classes common to client and server sides of an axis service. Also contains adapter classes for axis generated classes.
3. EPCPS_Client: Implementation of public facing java interface.
4. EPCPS_Engine: Server side implementation of public facing java interfaces which will be used by all the different web service implementations.

5. EPCIS_Service: Different web-service implementations (for example: Axis, BEA) of the provisioning service.

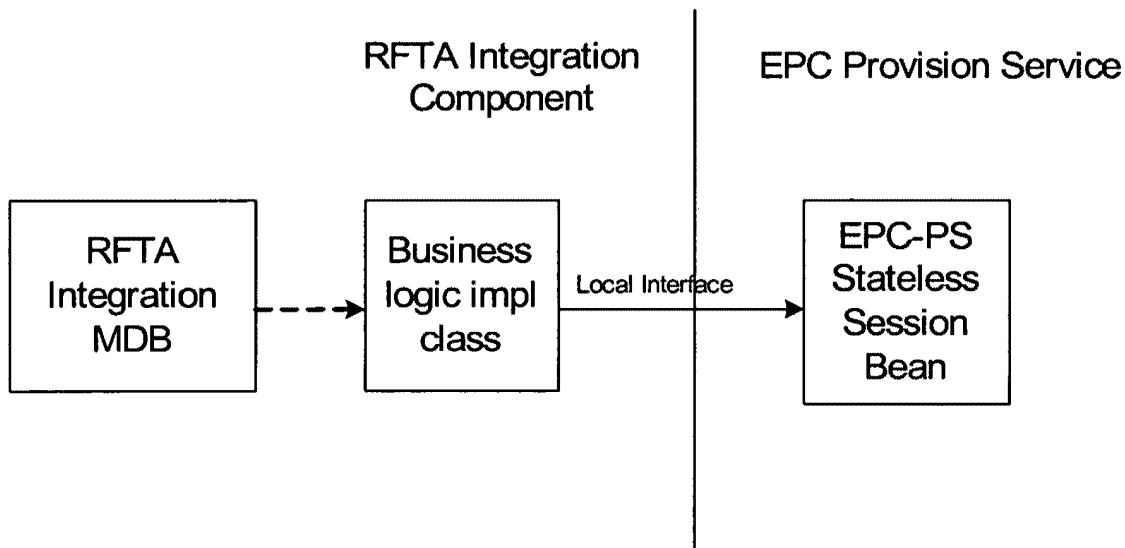
6.2 RFTA Integration Component

The only responsibility of this component for 1.0 is to accept EPC Cache Low report from the edge server and replenish the corresponding cache. This can be accomplished using a message driven bean that will accept 2 kinds of messages:

1. Define cache parameters: This message associates a cache with its edge server URL, filter value, tag size and chunk size. A cache is identified using the cache name and savant ID.
2. EPC Cache low report: This report will be sent by the edge server when the cache falls below a certain pre-specified value. The message driven bean will get EPCs for replenishment from the provisioning service and replenish the cache (the URL for the edge server is specified by message 1).

The message driven bean will delegate all the business logic work to a simple java class, so that supporting other communication mechanisms in the future will be easy.

RFTA integration component will communicate with the EPC Provisioning Service through local EJB interfaces to avoid communication overheads.



6.2.1 VCC Components

One VCC component – RFTA_PS (Or may be will just use EPCPS_Broker component created last year)

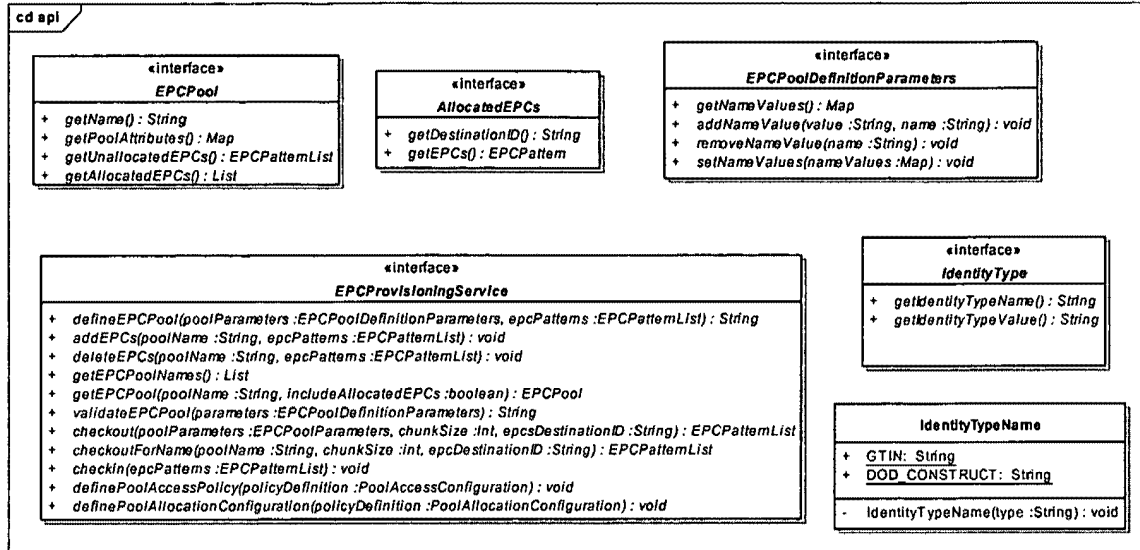
6.3 Admin Web Client

TBD

7 Design Detail

7.1 EPC Provisioning Service

7.1.1 Package com.connecterra.epcps.api– key classes UML



7.1.1.1 com.connecterra.epcps.api.EPCPool

Pool attributes represent attributes like which GTIN this pool was created for ([GTIN, <gtin-value] will be the name-value pair in this case) or what tag-length the pool is for etc.

Allocated EPCs gives EPCs that were allocated and opaque id of the destination of the EPCs.

An EPC Pool cannot contain EPCs of more than one tag-type. The EPCs will be stored as pure identities.

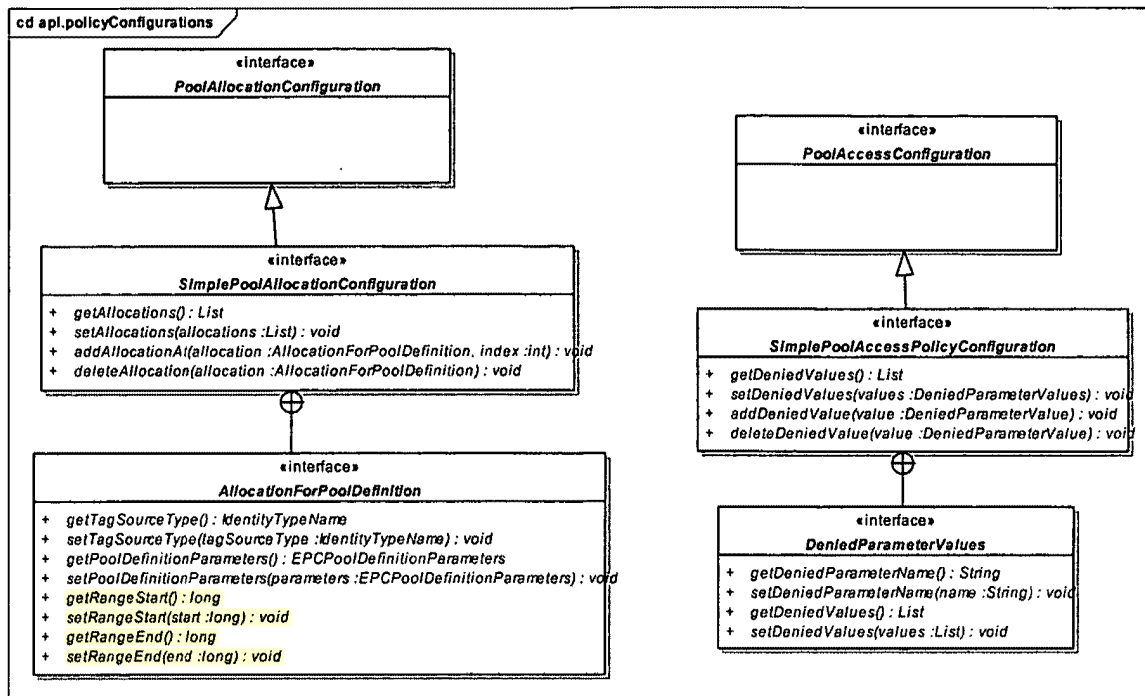
7.1.1.2 com.connecterra.epcps.api.EPCPoolDefinitionParameters Class

This defines the list of parameters for which a pool may be requested. The parameters are a set of name-value pairs.

7.1.1.3 com.connecterra.epcps.api.EPCProvisioningService Class

This is the public interface to the provisioning service.

7.1.2 Package com.connecterra.epcps.api.policyConfigurations— key classes UML



7.1.2.1 com.connecterra.epcps.api.policyConfigurations.PoolAllocationConfiguration Class

This serves as the super-type for different pool allocation configurations that may be supported by the provisioning service. Using the pool allocation configuration administrator can define which pool gets allocated for a given set of pool definition parameters and also define how the pool may get populated.

7.1.2.2 com.connecterra.epcps.api.policyConfigurations.SimplePoolAllocationConfiguration Class

This allows you to make associations between an EPCPoolDefinitionParameters, identity type, and a serial number range. This association defines how to select a pool and how the pool gets populated initially. The order of the associations is maintained and is used when matching an input request with the definitions.

On receiving a request for an EPC pool based on an input EPCPoolDefinitionParameters, the first AllocationForPoolDefinition that matches the input EPCPoolDefinitionParameters will be selected. Using this, you can get the pool name and if needed, the initial range for the pool.

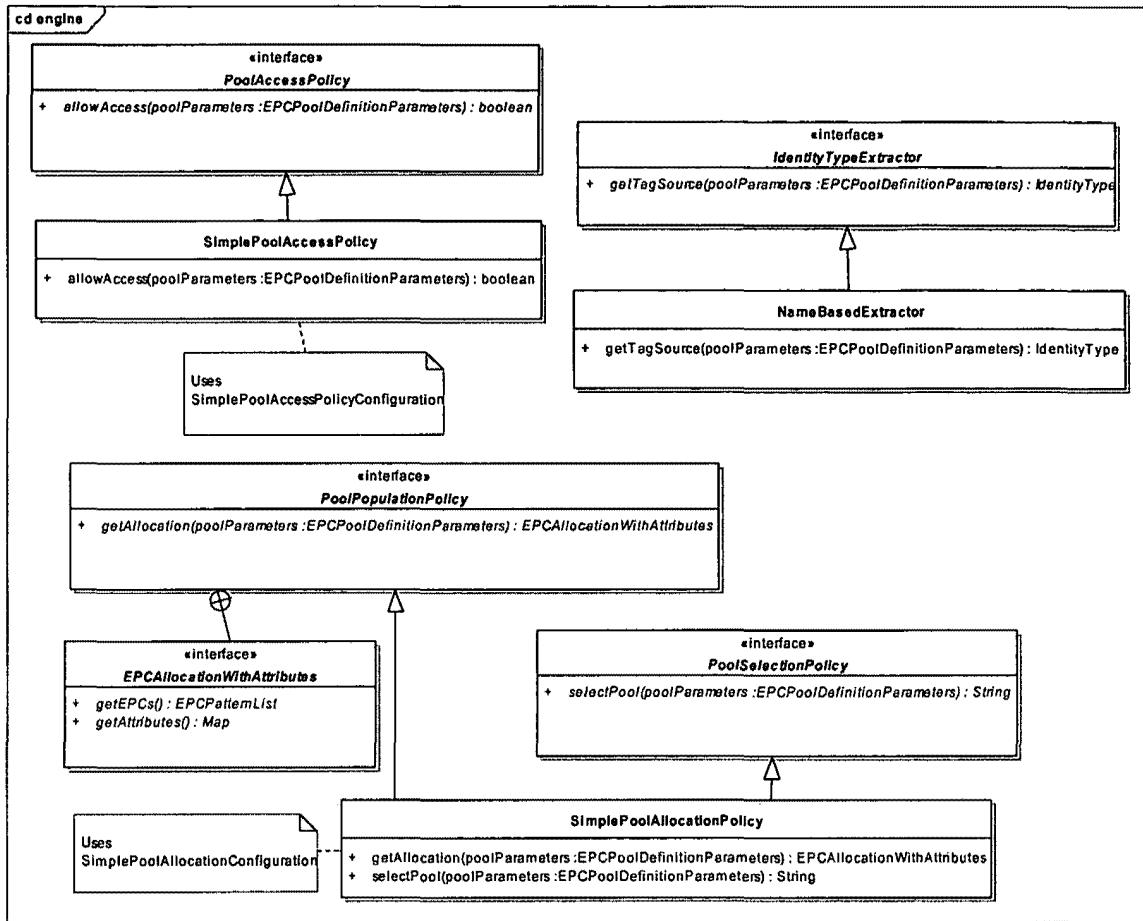
7.1.2.3 com.connecterra.epcps.api.policyConfigurations.PoolAccessConfiguration Class

Super-type for different pool access configurations that may be supported by the provisioning service.

7.1.2.4 com.connecterra.epcps.api.policyConfigurations.SimplePoolAccessConfiguration Class

This interface lets you disallow access to EPC pools for certain pool definition parameters. This is a simple configuration where you only configure negative entries. If any of the parameters in an input request matches any of the denied parameters, the request will not be granted access to an EPC pool. (This is not a security mechanism; it is just an easy way to disallow workflows from tagging certain products, for example).

7.1.3 Package com.connecterra.epcps.engine– key classes UML



7.1.3.1 com.connecterra.epcps.engine.PoolAccessPolicy Class

This is the contract for pool access policies.

7.1.3.2 com.connecterra.epcps.engine.SimplePoolAccessPolicy Class

Class that implements pool access policy using the SimplePoolAccessConfiguration.

7.1.3.3 com.connecterra.epcps.engine.IdentityTypeExtractor Class

Interface that lets you get to a IdentityType from a given EPCPoolDefinitionParameters object.

7.1.3.4 com.connecterra.epcps.engine.NameBasedExtractor Class

Class that gets a IdentityType by looking for identity type names (defined in IdentityTypeName) in the parameter list.

7.1.3.5 com.connecterra.epcps.engine.PoolSelectionPolicy Class

This is the contract for policies that let you select a pool name from a given EPCPoolDefinitionParameter.

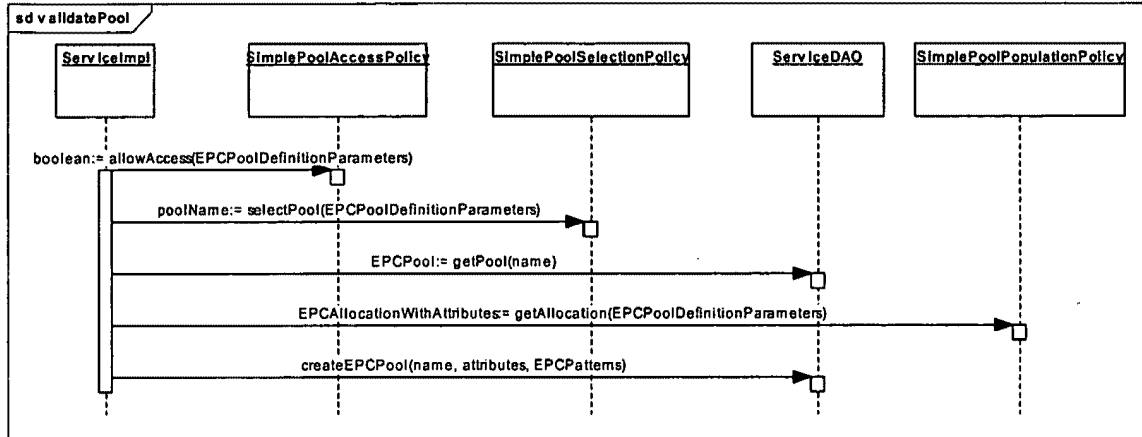
7.1.3.6 com.connecterra.epcps.engine.PoolPopulationPolicy Class

This is the contract for policies that gives you the range for populating a pool based on an EPCPoolDefinitionParamter.

7.1.3.7 com.connecterra.epcps.engine.SimplePoolAllocationPolicy Class

This class implements PoolSelectionPolicy and PoolPopulationPolicy. It uses the SimplePoolAllocationConfiguration that may be specified by the administrator.

Below is a sequence diagram showing how the different classes interact with each other on a validatePool call:



7.1.4 Dependencies on ALEPC component

Provisioning service design uses com.connecterra.ale.epc.EPCPatternList. The pattern list will have to be modified to support a new type of pattern for pure identity based patterns.

Details of this: TBD

7.1.5 Dependencies on ALEAPI component

Changes that may need to be made to the ALEPC interface:

1. Method to get ALEID from the ALEPC interface.
2. May need to modify PCSpec to include the filter to be used when programming a tag and probably change poll call as well to take filter parameter.

7.1.6 Schema Design

EPC Pool

Column Name	Column Type	Constraints
pool_id	integer	Primary Key
pool_name	varchar2(255)	Unique Index
creation_date	date	
last_modified_date	date	

Pool Attributes

Column Name	Column Type	Constraints
attr_row_id	Integer	Primary Key
pool_id	integer	Indexed
attr_name	varchar2(255)	Indexed

attr_value	varchar2(255)	Indexed
------------	---------------	---------

Separate tables will be used for allocated and unallocated EPCs in a pool. This is done so that searches on the unallocated EPC pattern list will not be slowed down because of all the allocated EPCs. The allocated EPC pattern list is assumed to grow very long.

Similarly, different tables will be used to store different types of tag, instead of one table storing the EPC pattern string. This way searching on individual fields in the pattern in the database will be efficient. We will have to search on the individual fields when assigning EPCs to a pool, allocating EPCs to a client, checking in EPCs etc.

Tables for allocated and unallocated EPCs for SGTIN patterns are shown below:

SGTIN Unallocated Epcs

Column Name	Column Type	Constraints
unallocated_epcs_id	integer	Primary Key
pool_id	integer	Indexed
company_prefix	varchar2(12)	Indexed
item_ref_and_ind	varchar2(7)	Indexed
serial_range_start	long	Indexed
serial_range_end	long	Indexed
creation_date	date	

SGTIN Allocations

Column Name	Column Type	Constraints
allocation_id	integer	Primary Key
pool_id	integer	Indexed
company_prefix	varchar2(12)	Indexed
item_ref_and_ind	varchar2(7)	Indexed
serial_range_start	long	Indexed
serial_range_end	long	Indexed

SGTIN Allocation Details

Column Name	Column Type	Constraints
allocation_details_id	integer	Primary Key
allocation_id	integer	Indexed
destination_id	varchar2(255)	Indexed
serial_range_start	long	Indexed
serial_range_end	long	Indexed
allocation_date	date	

7.1.6.1 Pool Access Policy Configuration

Access Parameters

Column Name	Column Type	Constraints
parameter_name	varchar2(255)	Primary Key

Access Parameter Values

Column Name	Column Type	Constraints
row_id	integer	Primary Key
parameter_name	varchar2(255)	Indexed
parameter_value	varchar2(255)	

7.1.6.2 Pool Allocation Policy Configuration

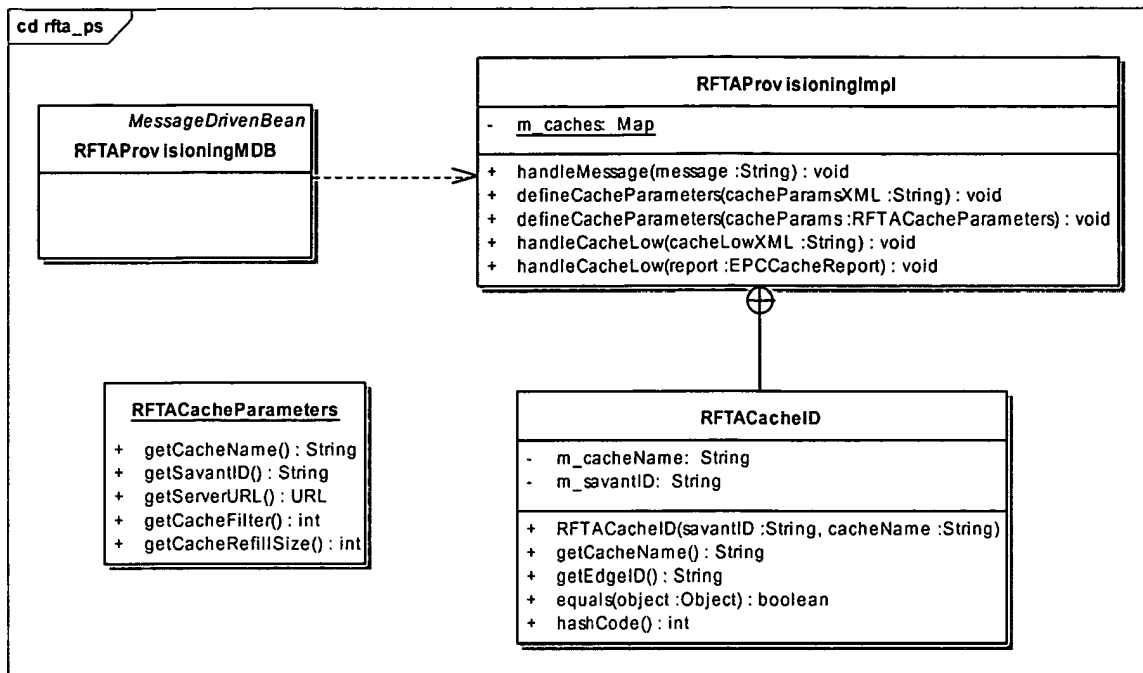
Pool Allocation Configuration

Column Name	Column Type	Constraints
config_row_id	integer	Primary Key
identity_type_name	varchar2(255)	Indexed
range_start	long	
range_end	long	
ordinal	int	

Pool Allocation Parameters

Column Name	Column Type	Constraints
row_id	integer	Primary Key
config_row_id	integer	Indexed
parameter_name	varchar2(255)	Indexed
parameter_value	varchar2(255)	Indexed

7.2 RFTA Integration Component



7.3 Admin Web Client

TBD

8 External Changes

8.1 Specific add/update to Java API

None.

8.2 Specific add/update to XML API

1. EPC-PS XSD
2. Any changes to ALEPC?

8.3 Specific add/update to Web Services API

1. WSDL for EPC Provisioning service
2. ALEPC changes?

8.4 Specific add/update to Configuration

Configuration parameters for EPC-PS:
Configuration parameters for RFTA-PS:

8.5 Specific add/update to Scripts

None.

8.6 Specific add/update to GUI

TBD.

8.7 Specific add/update to Command Line Interface

None.

8.8 Specific add/update to Protocol

None.

8.9 Specific add/update to Syntax

Any restriction or rule that customers must obey when choosing names or similar artifacts

8.10 Specific add/update to Database/Persistence

Refer to database schema section above. There will be no supported public view of the schema.

8.11 Specific add/update to Installer

TBD.

8.12 Specific add/update to Samples

TBD.

8.13 Specific add/update to jar files and other libraries

8.14 Security considerations

We may need authentication from the admin client – TBD.

Other things to consider:

- Lower level of authorization for unauthenticated clients.
- Limit the # of checkouts from an edge
- Allowing access to only a certain list of known edges.

8.15 Compatibility Considerations

8.16 Performance Considerations

8.17 Testing considerations

1. All checkouts and checkins should be logged.

8.18 Deployment considerations



ConnectTerra EPC Provisioning Service Design

Version of 2005-04-18 at 13:26

This document is Company Confidential

Table of Contents

ConnectTerra EPC Provisioning Service Design	1
1 Introduction	3
2 Key Terminology	3
3 Feature Overview	3
4 Use Cases	4
5 Architectural Overview	4
6 Design Overview	5
6.1 EPC Provisioning Service	6
6.2 RFTA Integration Component	7
6.2.1 VCC Components	7
6.3 Admin Web Client	7
7 Design Detail	8
7.1 EPC Provisioning Service	8
7.1.1 Package com.connectterra.epcps.api– key classes UML	8
7.1.2 Package com.connectterra.epcps.api.policyConfigurations– key classes UML	9
7.1.3 Package com.connectterra.epcps.engine– key classes UML	10
7.1.4 Dependencies on ALEEPC component	11
7.1.5 Dependencies on ALEAPI component	11
7.1.6 Schema Design	11
7.2 RFTA Integration Component	14
7.3 Admin Web Client	14
8 External Changes	14
8.1 Specific add/update to Java API	14
8.2 Specific add/update to XML API	14
8.3 Specific add/update to Web Services API	14

8.4	Specific add/update to Configuration	14
8.5	Specific add/update to Scripts	14
8.6	Specific add/update to GUI	15
8.7	Specific add/update to Command Line Interface	15
8.8	Specific add/update to Protocol.....	15
8.9	Specific add/update to Syntax	15
8.10	Specific add/update to Database/Persistence.....	15
8.11	Specific add/update to Installer	15
8.12	Specific add/update to Samples.....	15
8.13	Specific add/update to jar files and other libraries	15
8.14	Security considerations.....	15
8.15	Compatibility Considerations.....	15
8.16	Performance Considerations.....	15
8.17	Testing considerations	15
8.18	Deployment considerations	15

Revision history: latest at the top please			
Version	Author	History	Date(yyyy-mm-dd)
001	Manju James	Initial version	2004-04-12

1 Introduction

This document outlines the design for EPC Provisioning Service in the ConnecTerra Enterprise Server 1.0 product.

2 Key Terminology

This section lists key terms and definitions as utilized in the design:

EPC Pool: A named list of EPCs maintained in the provisioning system. EPCs may be checked out or checked in to the pool by outside applications.

Identity Type: As used in EPC Tag Data Standards document. Examples of some identity types used in this design doc are GTIN, SSCC, DoD Construct.

Pool Name, Cache Name: This design assumes that the name of a cache in the Edge Server is the same as the pool name in the provisioning service. Therefore the terms pool name and cache name are used interchangeably in this document.

3 Feature Overview

Tag writing systems need a central authority from which they can receive unique EPCs. The EPC Provisioning Service is designed to satisfy this need.

In addition to providing a service to give out unique EPCs, a design goal is also to make it easy for administrators to define and maintain pools. Administrators will be able to specify parameters based on which naming of pools and EPC allocations to pool can be done automatically

The service must be generic enough to work with different ConnecTerra products as well as third party products. However, we will provide specific sub-components to integrate smoothly with ConnecTerra products. For example, we will provide sub-components that will help integrate with RFTagAware Edge Server and ConnecTerra's local workflow components.

4 Use Cases

1. Local workflow checks out EPCs from the provisioning service by specifying the parameters for the checkout. An example of the parameters specified is (GTIN=10037000308048, TAG_LENGTH=64, LOCATION_ID=tag@ship-station-1). The provisioning service will check to see if the set of parameters has access to a pool and then returns EPCs to the local workflow. In this process, as required, the provisioning service may create a new pool and allocate EPCs to it. The EPCs given out will be marked as 'allocated' by the provisioning service.
2. Provisioning system receives a tag-cache-low report from the RFTagAware Edge Server. The provisioning service will checkout EPCs from the correct pool and replenish the cache in the Edge Server.
3. Administrator finds out that a product or station must no longer be allowed to get EPCs from the provisioning service. Administrator makes changes to the provisioning service and requests with those parameters will be denied access.
4. An edge server tag cache is reset and the EPCs left in the cache is released back to the provisioning service. The provisioning service removes these from allocated list and puts them back in the available list.
5. Administrator finds out that a tagging station needs to switch from using 64 bit tags to 96 bit tags. Administrator sets up pool naming policy and pool population policy to enable that.

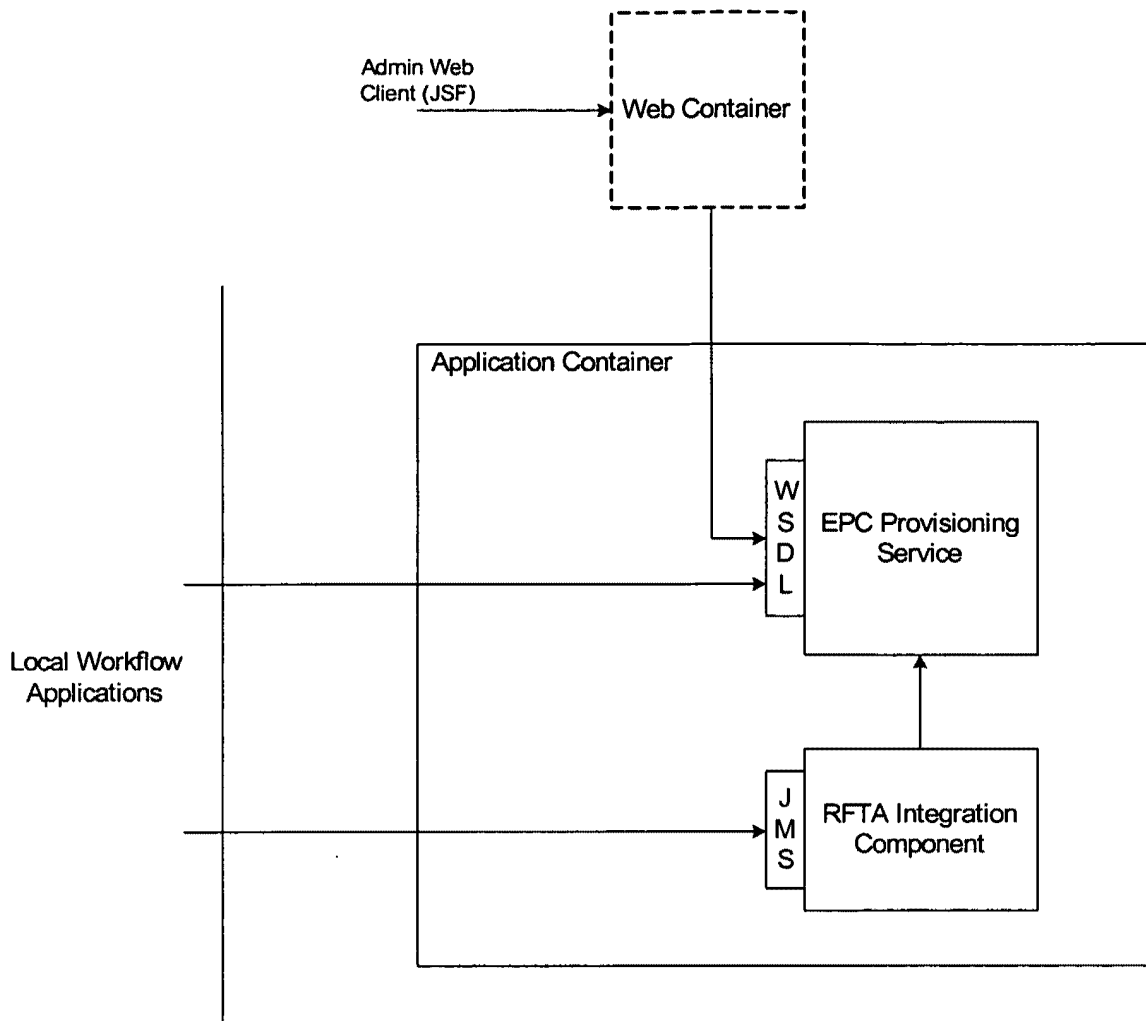
5 Architectural Overview

The provisioning service system will be composed of 2 main components:

1. EPC Provisioning Service: This is a web service that will provide generic provisioning related operations like administering EPC pools, checkout or check-in from EPC pools etc.
2. RFTA Integration Component: This component is used to integrate between RFTagAware Edge Server and the provisioning service. For release 1.0, responsibility of this component is to accept cache-low notifications from the edge server and replenish the cache. For 1.0, this component will support only a JMS interface.

There will be a web based GUI for administering the provisioning system. This GUI will be developed using JSF technology and communication with the provisioning service will be through the web service interface.

Local workflows may communicate with the EPC Provisioning Service or RFTA Integration Component as needed.



6 Design Overview

Figure 1 below shows interaction between local workflow components, edge server and EPC provisioning system when the local workflow encounters a new item for which it may need to define a cache at the edge server.

Figure 2 shows the interactions between the edge server, RFTA Integration Component and EPC Provisioning Service when the edge server detects that a cache's size has reached the low threshold value.

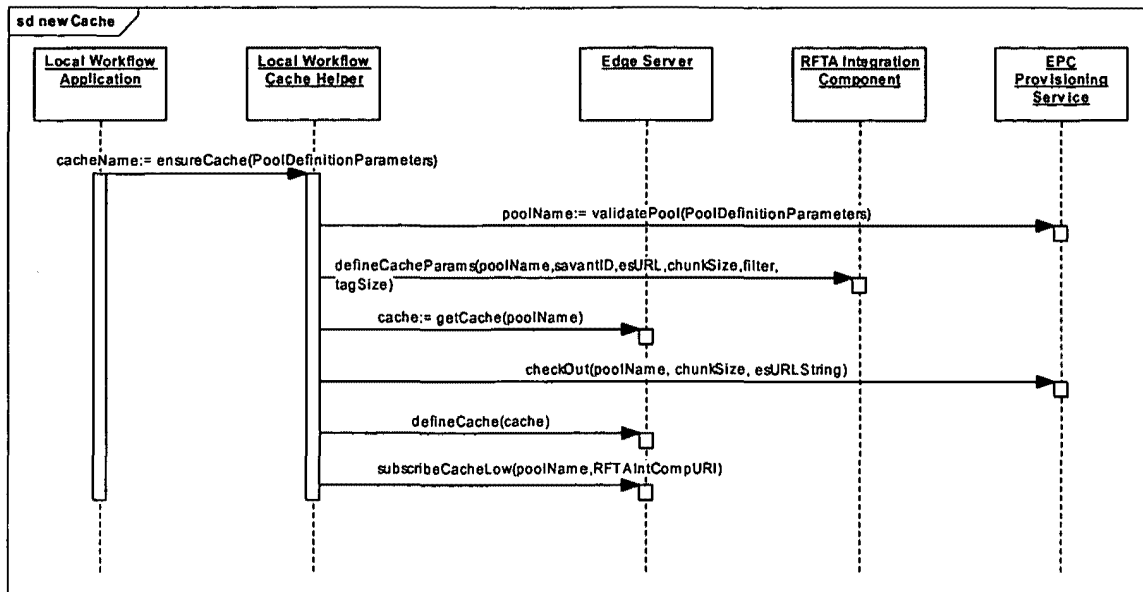


Figure 1

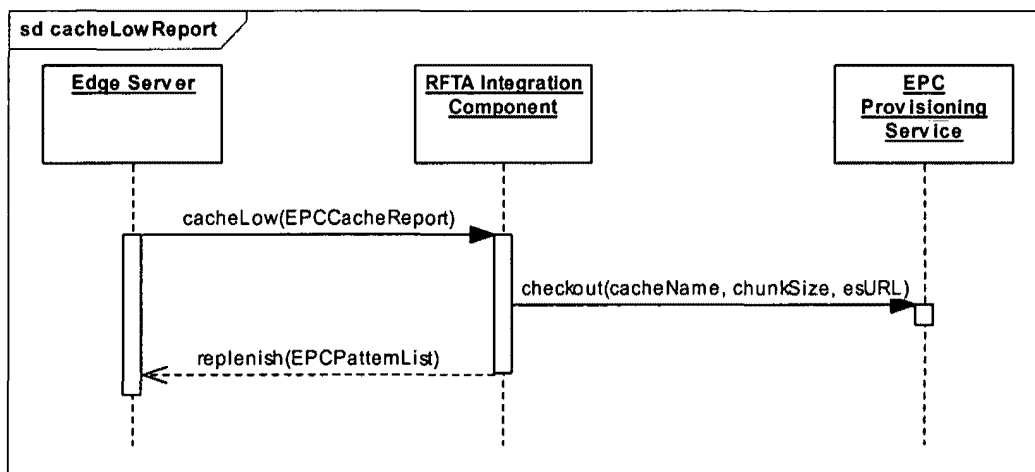


Figure 2

6.1 EPC Provisioning Service

The design overview diagram for the web service will be similar to other web services in Enterprise Server (see [EPCIS Design.doc](#) for details).

VCC components for this are:

1. EPCPS_API: This will contain the public facing interfaces (java, wsdl and xsd)
2. EPCPS_Axis: Axis generated classes common to client and server sides of an axis service. Also contains adapter classes for axis generated classes.
3. EPCPS_Client: Implementation of public facing java interface.
4. EPCPS_Engine: Server side implementation of public facing java interfaces which will be used by all the different web service implementations.

5. EPCIS_Service: Different web-service implementations (for example: Axis, BEA) of the provisioning service.

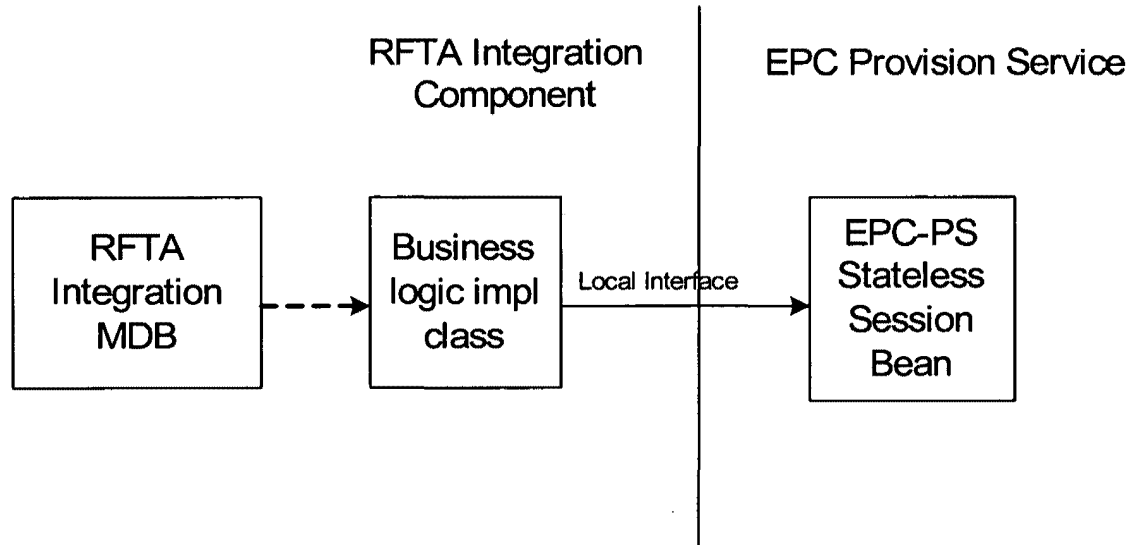
6.2 RFTA Integration Component

The only responsibility of this component for 1.0 is to accept EPC Cache Low report from the edge server and replenish the corresponding cache. This can be accomplished using a message driven bean that will accept 2 kinds of messages:

1. Define cache parameters: This message associates a cache with its edge server URL, filter value, tag size and chunk size. A cache is identified using the cache name and savant ID.
2. EPC Cache low report: This report will be sent by the edge server when the cache falls below a certain pre-specified value. The message driven bean will get EPCs for replenishment from the provisioning service and replenish the cache (the URL for the edge server is specified by message 1).

The message driven bean will delegate all the business logic work to a simple java class, so that supporting other communication mechanisms in the future will be easy.

RFTA integration component will communicate with the EPC Provisioning Service through local EJB interfaces to avoid communication overheads.



6.2.1 VCC Components

One VCC component – RFTA_PS (Or may be will just use EPCPS_Broker component created last year)

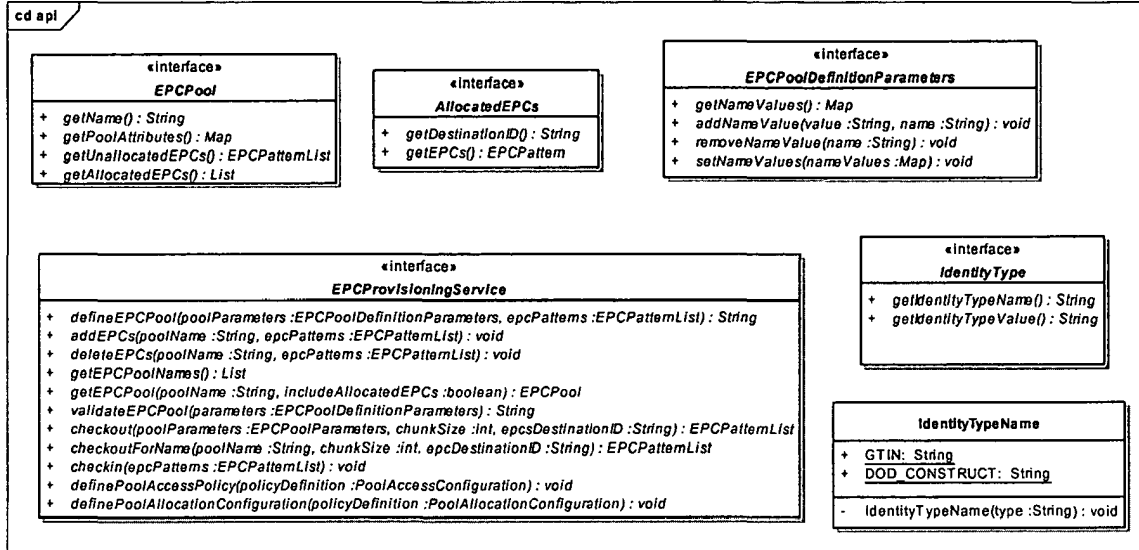
6.3 Admin Web Client

TBD

7 Design Detail

7.1 EPC Provisioning Service

7.1.1 Package com.connecterra.epcps.api– key classes UML



7.1.1.1 com.connecterra.epcps.api.EPCPool

Pool attributes represent attributes like which GTIN this pool was created for ([GTIN, <gtin-value] will be the name-value pair in this case) or what tag-length the pool is for etc.

Allocated EPCs gives EPCs that were allocated and opaque id of the destination of the EPCs.

An EPC Pool cannot contain EPCs of more than one tag-type. The EPCs will be stored as pure identities.

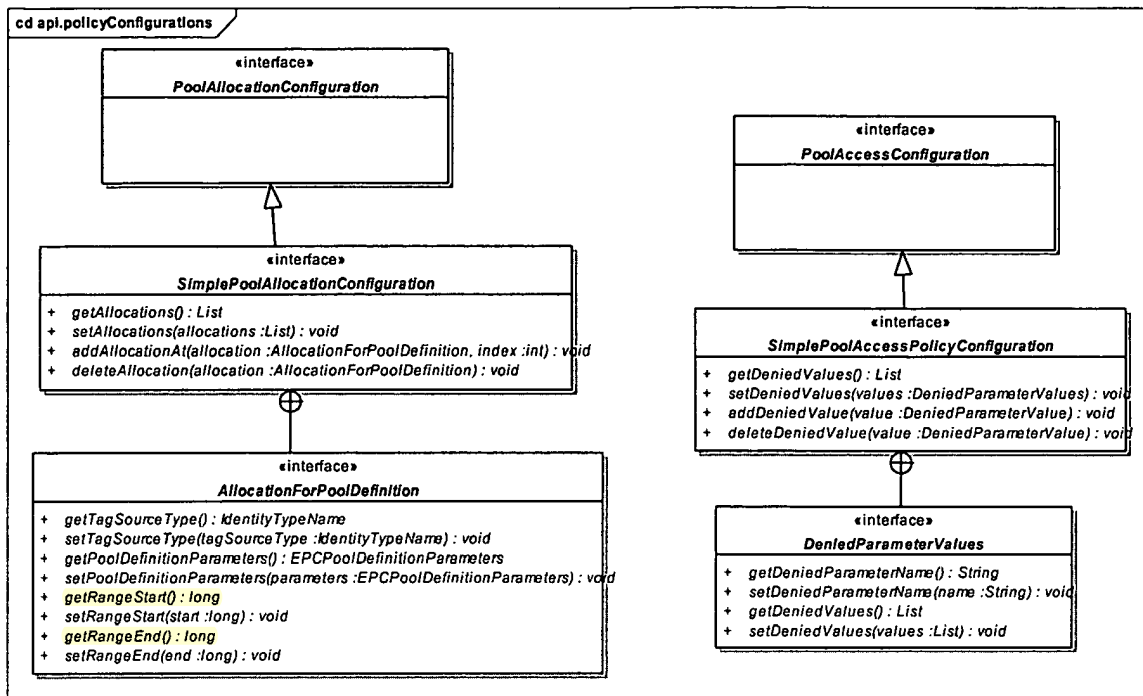
7.1.1.2 com.connecterra.epcps.api.EPCPoolDefinitionParameters Class

This defines the list of parameters for which a pool may be requested. The parameters are a set of name-value pairs.

7.1.1.3 com.connecterra.epcps.api.EPCProvisioningService Class

This is the public interface to the provisioning service.

7.1.2 Package com.connecterra.epcps.api.policyConfigurations– key classes UML



7.1.2.1 com.connecterra.epcps.api.policyConfigurations.PoolAllocationConfiguration Class

This serves as the super-type for different pool allocation configurations that may be supported by the provisioning service. Using the pool allocation configuration administrator can define which pool gets allocated for a given set of pool definition parameters and also define how the pool may get populated.

7.1.2.2 com.connecterra.epcps.api.policyConfigurations.SimplePoolAllocationConfiguration Class

This allows you to make associations between an EPCPoolDefinitionParameters, identity type, and a serial number range. This association defines how to select a pool and how the pool gets populated initially. The order of the associations is maintained and is used when matching an input request with the definitions.

On receiving a request for an EPC pool based on an input EPCPoolDefinitionParameters, the first AllocationForPoolDefinition that matches the input EPCPoolDefinitionParameters will be selected. Using this, you can get the pool name and if needed, the initial range for the pool.

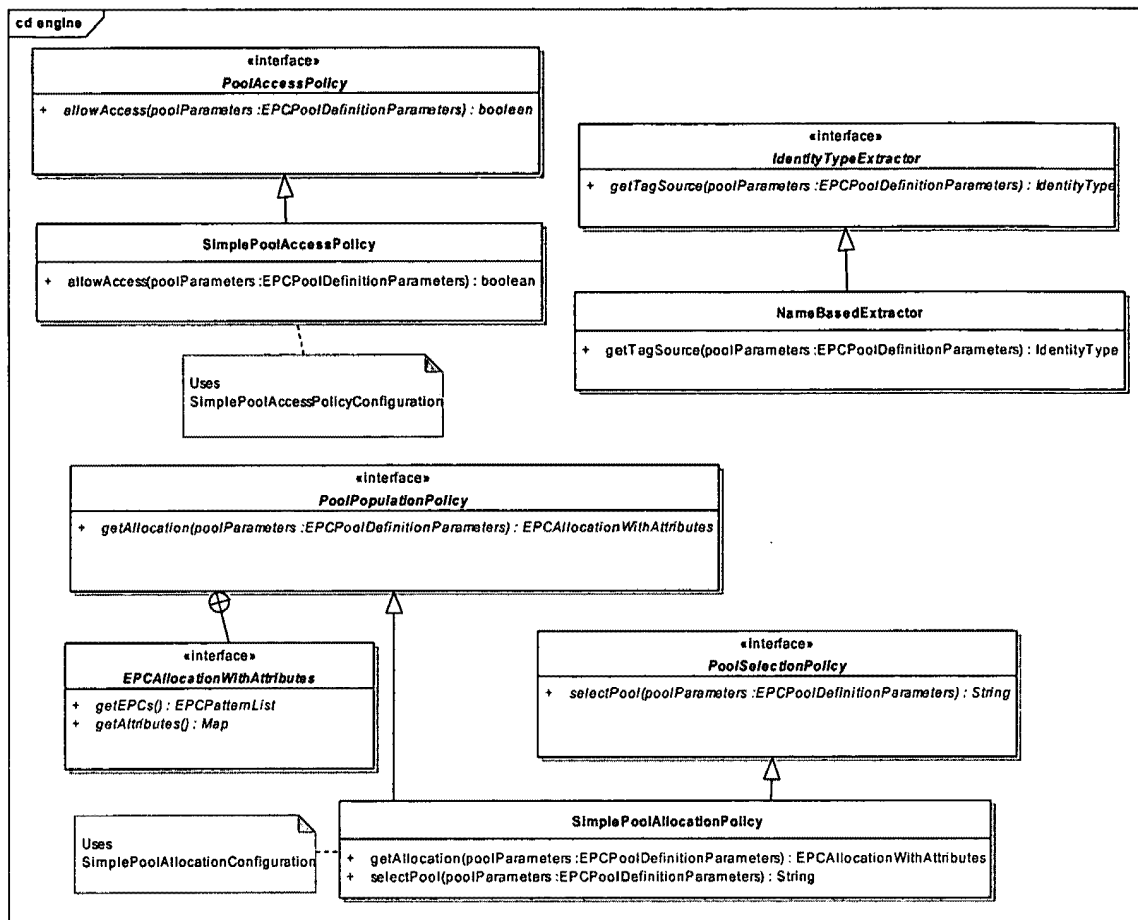
7.1.2.3 com.connecterra.epcps.api.policyConfigurations.PoolAccessConfiguration Class

Super-type for different pool access configurations that may be supported by the provisioning service.

7.1.2.4 com.connecterra.epcps.api.policyConfigurations.SimplePoolAccessConfiguration Class

This interface lets you disallow access to EPC pools for certain pool definition parameters. This is a simple configuration where you only configure negative entries. If any of the parameters in an input request matches any of the denied parameters, the request will not be granted access to an EPC pool. (This is not a security mechanism; it is just an easy way to disallow workflows from tagging certain products, for example).

7.1.3 Package com.connecterra.epcps.engine– key classes UML



7.1.3.1 com.connecterra.epcps.engine.PoolAccessPolicy Class

This is the contract for pool access policies.

7.1.3.2 com.connecterra.epcps.engine.SimplePoolAccessPolicy Class

Class that implements pool access policy using the SimplePoolAccessConfiguration.

7.1.3.3 com.connecterra.epcps.engine.IdentityTypeExtractor Class

Interface that lets you get to a IdentityType from a given EPCPoolDefinitionParameters object.

7.1.3.4 com.connecterra.epcps.engine.NameBasedExtractor Class

Class that gets a IdentityType by looking for identity type names (defined in IdentityTypeName) in the parameter list.

7.1.3.5 com.connecterra.epcps.engine.PoolSelectionPolicy Class

This is the contract for policies that let you select a pool name from a given EPCPoolDefinitionParameter.

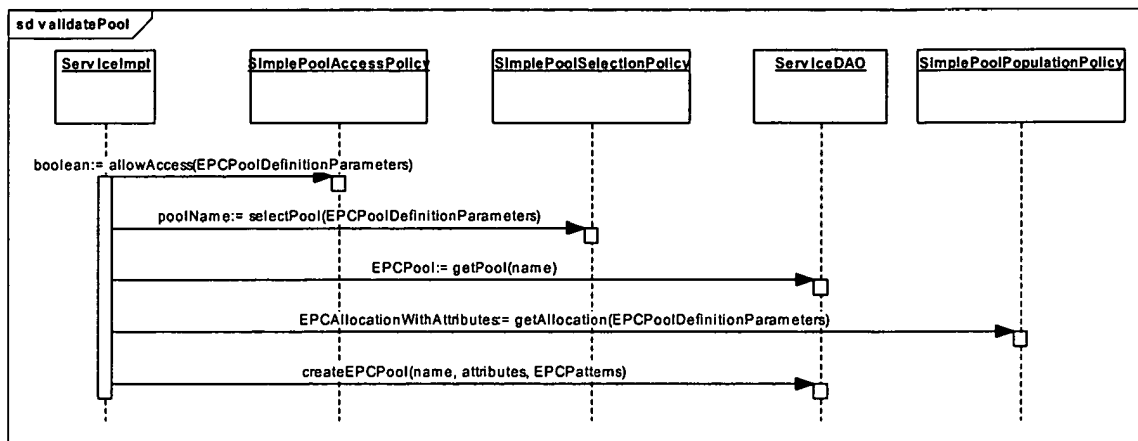
7.1.3.6 com.connecterra.epcps.engine.PoolPopulationPolicy Class

This is the contract for policies that gives you the range for populating a pool based on an EPCPoolDefinitionParamter.

7.1.3.7 com.connecterra.epcps.engine.SimplePoolAllocationPolicy Class

This class implements PoolSelectionPolicy and PoolPopulationPolicy. It uses the SimplePoolAllocationConfiguration that may be specified by the administrator.

Below is a sequence diagram showing how the different classes interact with each other on a validatePool call:



7.1.4 Dependencies on ALEPC component

Provisioning service design uses `com.connecterra.ale.epc.EPCPatternList`. The pattern list will have to be modified to support a new type of pattern for pure identity based patterns.

Details of this: TBD

7.1.5 Dependencies on ALEAPI component

Changes that may need to be made to the ALEPC interface:

1. Method to get ALEID from the ALEPC interface.
2. May need to modify PCSpec to include the filter to be used when programming a tag and probably change poll call as well to take filter parameter.

7.1.6 Schema Design

EPC Pool

Column Name	Column Type	Constraints
pool_id	integer	Primary Key
pool_name	varchar2(255)	Unique Index
creation_date	date	
last_modified_date	date	

Pool Attributes

Column Name	Column Type	Constraints
attr_row_id	Integer	Primary Key
pool_id	integer	Indexed
attr_name	varchar2(255)	Indexed

attr_value	varchar2(255)	Indexed
------------	---------------	---------

Separate tables will be used for allocated and unallocated EPCs in a pool. This is done so that searches on the unallocated EPC pattern list will not be slowed down because of all the allocated EPCs. The allocated EPC pattern list is assumed to grow very long.

Similarly, different tables will be used to store different types of tag, instead of one table storing the EPC pattern string. This way searching on individual fields in the pattern in the database will be efficient. We will have to search on the individual fields when assigning EPCs to a pool, allocating EPCs to a client, checking in EPCs etc.

Tables for allocated and unallocated EPCs for SGTIN patterns are shown below:

SGTIN Unallocated Epcs

Column Name	Column Type	Constraints
unallocated_epcs_id	integer	Primary Key
pool_id	integer	Indexed
company_prefix	varchar2(12)	Indexed
item_ref_and_ind	varchar2(7)	Indexed
serial_range_start	long	Indexed
serial_range_end	long	Indexed
creation_date	date	

SGTIN Allocations

Column Name	Column Type	Constraints
allocation_id	integer	Primary Key
pool_id	integer	Indexed
company_prefix	varchar2(12)	Indexed
item_ref_and_ind	varchar2(7)	Indexed
serial_range_start	long	Indexed
serial_range_end	long	Indexed

SGTIN Allocation Details

Column Name	Column Type	Constraints
allocation_details_id	integer	Primary Key
allocation_id	integer	Indexed
destination_id	varchar2(255)	Indexed
serial_range_start	long	Indexed
serial_range_end	long	Indexed
allocation_date	date	

7.1.6.1 Pool Access Policy Configuration

Access Parameters

Column Name	Column Type	Constraints
parameter_name	varchar2(255)	Primary Key

Access Parameter Values

Column Name	Column Type	Constraints
row_id	integer	Primary Key
parameter_name	varchar2(255)	Indexed
parameter_value	varchar2(255)	

7.1.6.2 Pool Allocation Policy Configuration

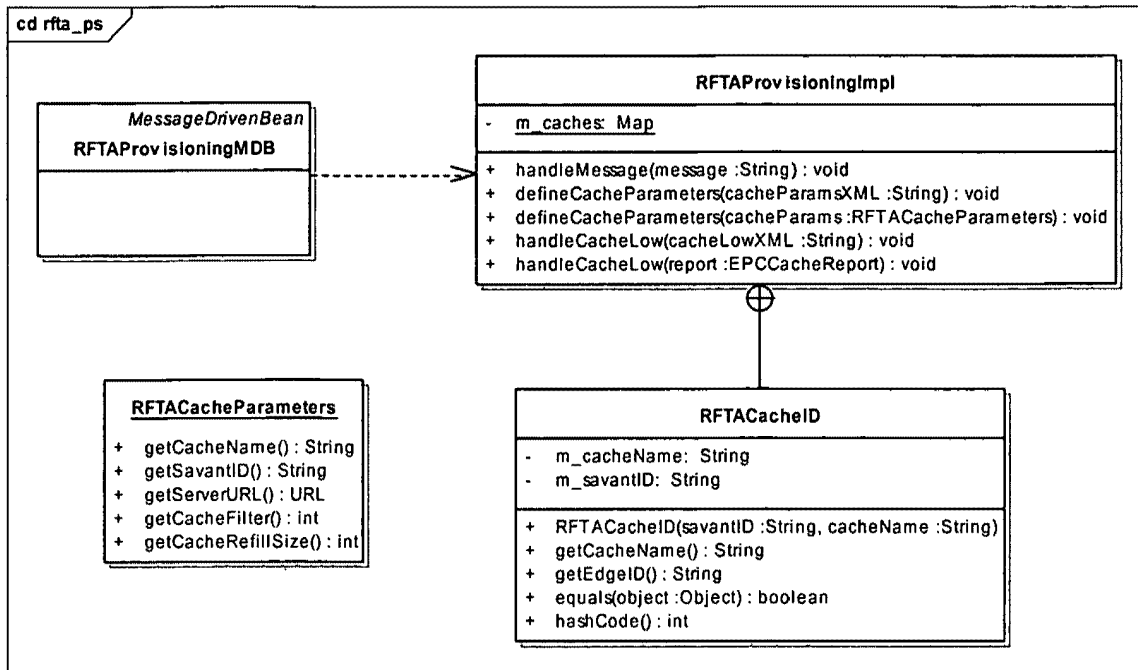
Pool Allocation Configuration

Column Name	Column Type	Constraints
config_row_id	integer	Primary Key
identity_type_name	varchar2(255)	Indexed
range_start	long	
range_end	long	
ordinal	int	

Pool Allocation Parameters

Column Name	Column Type	Constraints
row_id	integer	Primary Key
config_row_id	integer	Indexed
parameter_name	varchar2(255)	Indexed
parameter_value	varchar2(255)	Indexed

7.2 RFTA Integration Component



7.3 Admin Web Client

TBD

8 External Changes

8.1 Specific add/update to Java API

None.

8.2 Specific add/update to XML API

1. EPC-PS XSD
2. Any changes to ALEPC?

8.3 Specific add/update to Web Services API

1. WSDL for EPC Provisioning service
2. ALEPC changes?

8.4 Specific add/update to Configuration

Configuration parameters for EPC-PS:
Configuration parameters for RFTA-PS:

8.5 Specific add/update to Scripts

None.

8.6 Specific add/update to GUI

TBD.

8.7 Specific add/update to Command Line Interface

None.

8.8 Specific add/update to Protocol

None.

8.9 Specific add/update to Syntax

Any restriction or rule that customers must obey when choosing names or similar artifacts

8.10 Specific add/update to Database/Persistence

Refer to database schema section above. There will be no supported public view of the schema.

8.11 Specific add/update to Installer

TBD.

8.12 Specific add/update to Samples

TBD.

8.13 Specific add/update to jar files and other libraries

8.14 Security considerations

We may need authentication from the admin client – TBD.

Other things to consider:

- Lower level of authorization for unauthenticated clients.
- Limit the # of checkouts from an edge
- Allowing access to only a certain list of known edges.

8.15 Compatibility Considerations

8.16 Performance Considerations

8.17 Testing considerations

1. All checkouts and checkins should be logged.

8.18 Deployment considerations



EPC Provisioning Service Administrative Web Interface Design

Version of 2005-06-06 at 14:23

This document is Company Confidential

Table of Contents

Connecterra Feature xyz Design	1
1 Introduction	3
2 Key Terminology	3
3 Feature Overview	3
4 Use Cases	3
5 Architectural Overview	3
6 Design Overview	3
7 Design Detail	3
7.1 VCC Component 1 – key classes UML	8
7.1.1 com.connecterra.a.b.c Class	8
7.1.1.1 Dependencies on existing components	8
7.1.1.2 Modifications to existing components	8
7.1.2 com.connecterra.h.i.j Class	9
7.1.2.1 Dependencies on existing components	9
7.1.2.2 Modifications to existing components	9
7.2 VCC Component 2 – key classes UML	9
7.2.1 com.connecterra.k.l.m Class	9
7.2.1.1 Dependencies on existing components	9
7.2.1.2 Modifications to existing components	9
7.2.2 com.connecterra.n.o.p Class	9
7.2.2.1 Dependencies on existing components	9
7.2.2.2 Modifications to existing components	9
8 External Changes	9
8.1 Specific add/update to Java API	9

8.2	Specific add/update to XML API	9
8.3	Specific add/update to Web Services API	9
8.4	Specific add/update to Configuration	9
8.5	Specific add/update to Scripts	9
8.6	Specific add/update to GUI	9
8.7	Specific add/update to Command Line Interface	10
8.8	Specific add/update to Protocol.....	10
8.9	Specific add/update to Syntax	10
8.10	Specific add/update to Database/Persistence.....	10
8.11	Specific add/update to Installer	10
8.12	Specific add/update to Samples.....	10
8.13	Specific add/update to jar files and other libraries	10
8.14	Security considerations.....	10
8.15	Compatibility Considerations	10
8.16	Performance Considerations.....	10
8.17	Testing considerations	10
8.18	Deployment considerations	10

Revision history: latest at the top please			
Version	Author	History	Date(yyyy-mm-dd)
001	Marina Popova	Initial version	2005-06-03

1 Introduction

This document outlines the design for the Administrative Web Interface of the EPC Provisioning Service.

2 Key Terminology

This section lists key terms and definitions as utilized in the design:

3 Feature Overview

Admin application will provide the following functionality:

- Defining EPC pool configurations – “Define Pool Configuration” screen
- Viewing and editing existing pool configurations – “View Pool Configurations” screen
- Viewing details of existing EPC pools – “View Pools” screen

4 Use Cases

Any special testing considerations. Including special environment setup for system tests or the presence of GUI's

5 Architectural Overview

Admin application will be implemented as a Web Application deployable into any Application Server (JBoss, WebLogic, etc.) as a WAR file. Main implementation technologies will be JSP and JSF (Java Server Faces).

6 Design Overview

7 Design Detail

Depending on the effort and resources available, there will be either an initial screen that would allow choosing specific functional areas or three functional areas will be located in separate tabs.

7.1 Initial screen (optional)

7.2 View Pool Configurations screen

This screen provides detailed information about existing pool configurations. Pool configurations are represented in a table format in the correct order. The rows of the table (individual pool configurations) can be:

- deleted by checking the check-box in the leftmost column and clicking the “Delete” button located underneath the table
- re-ordered by using “Up” and “Down” buttons associated with the rows
- edited by clicking “Edit” buttons associated with the rows. This action will bring up a pop-up window with the “Define Pool Configuration” screen defined in the next section populated with the data of the selected pool configuration

A new pool configuration can be created by clicking on the “Create New Configuration” button. It will open a new screen, “Define Pool Configuration”.

A schematic representation of the screen is provided below

	Identity Type	Parameters		Range		Actions
		Name	Value	Start	End	
□	GTIN	location_id	picking01	0	10000	
		GTIN_value				
□	SSCC					

7.3 Define Pool Configuration screen

This screen can be used by an administrator to define parameters of a new pool configuration. The same screen is used for editing parameters of an existing pool configuration. The following pool configuration options can be set here:

- Identity Type – a pull-down menu with pre-defined supported identity types (GTIN or SSCC, for example)
- Pool Parameters – Name/Value pairs of pool parameters (for example, location_id). Represented in a table format. Table rows can be added or deleted.
- EPC Pool range start and end values – range of EPCs allowed in this pool

A draft representation of the screen is provided below

Define Pool Configuration

Identity Type

Range start end

Pool Parameters

	Name	Value
☒	Location_id	picking01
☒		
☒		

7.4 View Pools screen

This screen contains information about existing EPC pools. The screen is divided into two parts. The upper half contains a table with the list of existing pools and some brief information about the pools.

Pool Name	Creation Date	Pool Attributes		Checked out EPCs count	Available EPCs count	Actions
		Name	Value			
Name_1	06-01-05 12:00 PM	location_id	picking01	24000000	45000000	
		GTIN_value				
Name_2						

Each row (pool) has an associated "View" button. Clicking on the view button brings up detailed information about selected pool into the table in the lower half of the screen.

View Pools

Pool "PoolName"

Available EPCs	
range	allocation time
urn:epc:idpat:sgtin:0037000.130804.[0-9999999]	06-05-05 10:35:06
urn:epc:idpat:sgtin:0037000.130804.[20000000-29999999]	06-03-05 12:04:05

Checked Out EPCs		
range	requested by	date
urn:epc:idpat:sgtin:003000.130804.[10000000-19999999]		06-05-05
urn:epc:idpat:sgtin:0037000.130804.[30000000-30500000]		06-07-05

Available EPCs can be deleted from the pool by checking the check boxes of the corresponding table rows and clicking on the “Delete EPCs” button. More EPCs can be added to the pool by clicking on the “Add EPCs” button. As a result, a pop-up window will appear where a range of EPCs to be added can be specified.

Checked out EPCs can be checked back into the pool by clicking the “Check In” button.

8 VCC Component 1 – key classes UML

8.1.1 com.connecterra.a.b.c Class

8.1.1.1 Dependencies on existing components

The following dependencies exist

8.1.1.2 Modifications to existing components

The following modifications will be made

8.1.2 com.connecterra.h.i.j Class

8.1.2.1 Dependencies on existing components

The following dependencies exist

8.1.2.2 Modifications to existing components

The following modifications will be made

8.2 VCC Component 2 – key classes UML

8.2.1 com.connecterra.k.l.m Class

8.2.1.1 Dependencies on existing components

The following dependencies exist

8.2.1.2 Modifications to existing components

The following modifications will be made

8.2.2 com.connecterra.n.o.p Class

8.2.2.1 Dependencies on existing components

The following dependencies exist

8.2.2.2 Modifications to existing components

The following modifications will be made

9 External Changes

9.1 Specific add/update to Java API

A set of Java classes and interfaces that customers call from their own code

9.2 Specific add/update to XML API

An XML document type exposed to customers via file or network

9.3 Specific add/update to Web Services API

A web services API that customers create clients to call

9.4 Specific add/update to Configuration

Properties or other configuration file that customers may view or edit

9.5 Specific add/update to Scripts

Top-level script or executable that customers invoke to start programs running

9.6 Specific add/update to GUI

A GUI that customers use (excluding the installer UI, which has its own column)

9.7 Specific add/update to Command Line Interface

An interactive command-and-response user interface that customers use

9.8 Specific add/update to Protocol

A network protocol documented so that customers may write code that speaks one or both sides

9.9 Specific add/update to Syntax

Any restriction or rule that customers must obey when choosing names or similar artifacts

9.10 Specific add/update to Database/Persistence

Any repository for customer-generated data that must survive upgrades to software, including persistence across failures and restarts

9.11 Specific add/update to Installer

Any changes to the information requested or displayed during installation

9.12 Specific add/update to Samples

Any changes to sample code

9.13 Specific add/update to jar files and other libraries

9.14 Security considerations

9.15 Compatibility Considerations

9.16 Performance Considerations

Any customer facing performance issues that need to be considered

9.17 Testing considerations

9.18 Deployment considerations

ECSpecsOverview Frame

ECSpecs Panel (Top Panel)

Buttons		
Name	Description	Corresponds to API Feature
New	Create a new ECSpec. See pdfparatexton page -9.	ALEFactory.createECSpec()
Edit	Edit a highlighted (existing) ECSpec.	ALE.get(String ecSpecName)
Delete	Deletes a highlighted ECSpec, and all its subscribers.	ALE.undefine(String ecSpecName)
Import	ECSpecs can be defined using XML, as described in the <i>RFTagAware Programmer Guide</i> . This button lets you import an XML formatted ECSpec into the Administration Console.	XMLDeserializer.deserializeECSpec(java.io.Reader)
Activate Once	If you have already deployed an ECSpec, you can use the Activate Once button. The Activate Once button tells the Edge Server to use the ECSpec one time to create a report.	ALE.poll(String ecSpecName)
Column Labels		
Name	Description	Corresponds to API Feature
Name	The name of an ECSpec. You assign this name when you create the ECSpec.	ALE.listECSpecNames()
Activation Count	An ECSpec is "activated" each time the Edge Server uses it to create a report. ECSpecs can be activated when their boundary conditions are met, or when you press the Activate Once button. The Activation Count column shows how many times this ECSpec has been activated, since its creation.	ecSpecInfo = ALE.getECSpecInfo(ecSpecName) ecSpecInfo.getActivationCount();
Last Activation	The last time this ECSpec was activated.	ecSpecInfo.getLastActivated()
Last Reported	reviewers -- what does this column indicate?	ecSpecInfo.getLastReported()
Subscriber Count	How many subscribers this ECSpec has. For information about subscribers, see pdfparatexton page -2.	ecSpecInfo.getSubscriberCount()

Subscribers Panel (Bottom Panel)

Buttons		
Name	Description	Corresponds to API Feature
New	Create a new subscriber. A subscriber is a URI where the Edge Server should send the report you define in an ECSpec.	ALE.subscribe(ecSpecName, uri)
Edit	Edit an existing subscriber's URI.	list = ALE.listSubscribers(ecSpecName) uri = list.get(i); newUri = // edit uri ALE.unsubscribe(ecSpecName, uri) ALE.subscribe(ecSpecName, newUri)
Delete	Delete a subscriber.	ALE.unsubscribe(ecSpecName)
Column Labels		
Name	Description	Corresponds to API Feature
URI	The URI for this subscriber.	ALE.listSubscribers(ecSpecName)(URI)list.get(i)
Last Succeeded	The last time the Edge Server succeeded in sending a report to this subscriber	ecSubInfo = ALE.getECSubscriptionInfo(ecSpecName, uri) ecSubInfo.getLastSuccessTime()
Consecutive Failures	The number of consecutive times the Edge Server failed in its attempts to send a report to this subscriber.	ecSubInfo.getConsecutiveFailureCount()

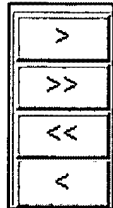
ECSpec Detail Frame

Menus

File Menu		
Name	Description	Corresponds to API Feature
View as Text	Lets you view the contents of this ECSpec in text format. For a sample, see pdfparatexton page -11.	N/A
View as XML	Displays the ECSpec in XML.	String str = XMLSerializer.serialize(ecSpec, true)
Revert	Returns to the settings this ECSpec had, before you made any modifications.	N/A
Export	ECSpecs can be defined using XML, as described in the <i>RFTagAware Programmer Guide</i> . This button lets you export the settings for this ECSpec into an XML file.	XMLSerializer.serialize(ecSpec, java.io.FileWriter)
Close	Closes the ECSpec detail frame, and returns you to the ECSpecs overview frame.	N/A
Tools Menu		
Name	Description	Corresponds to API Feature
Test	Activates the ECSpec you are working on (currently in memory), and shows you the report it would generate, in text format. For a sample, see pdfparatexton page - Error! Bookmark not defined.	ALE.immediate(ecSpec)
Redeploy	Redeploys the ECSpec (creates a new memory object for the ECSpec in the Edge Server). If you redeploy, you interrupt the subscribers who were getting reports from the ECSpec, before you redeployed.	ALE.redefine(ecSpecName, ecSpec)
Deploy As	Lets you deploy this ECSpec. Deploying means creating a memory object for the ECSpec in the Edge Server. When you click this button, the Administration Console opens a dialog box where you can type in a name for the ECSpec.	ALE.define(ecSpecName, ecSpec)
Help Menu		
Name	Description	Corresponds to API Feature
ECSpec Frame Help	Displays JavaHelp for the ECSpec detail frame.	n/a
About ALE Administration Console	Displays version and copyright information.	n/a

Logical Readers

<u>Name</u>	<u>Description</u>	<u>Corresponds to API Feature</u>
<u>Available</u>	Readers that are configured for this Edge Server.	ALE.listLogicalReaderNames()
<u>Selected</u>	Readers whose information will be used in the reports this ECSpec generates.	ECSpec.setLogicalReaderNames (List names)



Single arrow

Double arrow

Double arrow

Single arrow

Boundary

Start		
Start boundaries tell the ECSpec when to start listening to the readers, and storing the tag information it gets from the readers for use in the ECSpec's report. [lk: these are mutually exclusive options, but I don't see any way of resetting an ECSpec from within the java api... so I'm going to make guesses as to how to implement – kt should check this]		
Name	Description	Corresponds to API Feature
Continuous	reviewers -- what exactly does this mean? Once this ECSpec is deployed, start gathering information from the readers immediately, and continue to gather information continuously until you reach the stop boundary?	ECSpec.setRepeatPeriod(ALEFactory.createECInterval()) ECSpec.setStartTrigger(null)
Repeat Period	Start listening for tag information from specified readers each <i>n</i> milliseconds. For example, if you set Repeat Period to 5000 milliseconds (5 seconds), the ECSpec will start listening for tag information each 5 seconds.	ECSpec.setRepeatPeriod(ALEFactory.createECInterval(<i>n</i> , ECIntervalUnit.MS)) ECSpec.setStartTrigger(null)
Trigger URI	reviewers -- need info and an example here	ECSpec.setRepeatPeriod(ALEFactory.createECInterval()) ECSpec.setStartTrigger(uri)
Stop		
Stop boundaries tell the ECSpec when to stop listening to the readers. When the stop boundary is met, the ECSpec prepares its report with the tag information it has.		
Name	Description	Corresponds to API Feature
Duration	How long to accumulate the tag information that the ECSpec will include in its report. You can express this in either milliseconds or read cycles. Milliseconds: Tells the ECSpec to gather tag information for <i>n</i> milliseconds, and prepare its report based on the information it gathered during that time. Read Cycles: A read cycle is a single complete scan of all tags in a reader's field. This generally happens a few times a second, and is defined by individual reader manufacturers. If you click Read Cycles , you are telling the ECSpec to gather tag information for <i>n</i> read cycles, and prepare its report based on the information it gathered from those read cycles.	ECSpec.setDuration(ALEFactory.createECInterval(<i>n</i> , ECIntervalUnit.MS READ_CYCLES))
Stable Field Interval	When the reader(s) keep reporting the same tag information for a "stable field interval,"	ECSpec.setStableFieldInterval(ALEFactory.createECInterval(<i>n</i> ,

	then the ECSpec stops listening and creates its report. You can express this in either milliseconds or read cycles. Milliseconds: When the reader(s) keep reporting the same tag information for <i>n</i> milliseconds, the ECSpec stops listening and creates its report. Read Cycles: When the reader(s) keep reporting the same tag information for <i>n</i> read cycles, the ECSpec stops listening and creates its report.	ECIntervalUnit.MS READ_CYCLES))
Trigger URI	reviewers -- need info and an example here	ECSpec.setStopTrigger(uri)

Application Data

This is a free form text area that you can use to pass along comments to an application that is accessing this ECSpec programmatically.

Report Specifications

<u>Name</u>	<u>Description</u>	<u>Corresponds to API Feature</u>
New Report	Lets you create a new report. Opens the Report section. See pdfparatexton page -6.	ALEFactory.createECReportSpec()

Report

<u>Name</u>	<u>Description</u>	<u>Corresponds to API Feature</u>
Report field	Type a descriptive name for this report.	ECReportSpec.setName(String)
Report Type		
<u>Name</u>	<u>Description</u>	<u>Corresponds to API Feature</u>
Count	List the number of tags that match the conditions for this ECSpec.	ECReportSpec.setReportOutputSpec(ALEFactory.createECCountReportOutputSpec())
List	List the actual EPC values for each tag that matches the conditions for this ECSpec.	ECReportSpec.setReportOutputSpec(ALEFactory.createECListReportOutputSpec())
Report Set		
<u>Name</u>	<u>Description</u>	<u>Corresponds to API Feature</u>
Current	Include all tags.	ECReportSpec.setReportSetSpec(ECReportSetSpec.CURRENT))
Additions	Include only those tags that have been added, since the last time this ECSpec produced a report.	ECReportSpec.setReportSetSpec(ECReportSetSpec.ADDITIONS))
Deletions	Include only those tags that have been deleted, since the last time this ECSpec produced a report.	ECReportSpec.setReportSetSpec(ECReportSetSpec.DELETIONS))
Behaviour when Empty		
<u>Name</u>	<u>Description</u>	<u>Corresponds to API Feature</u>

Include in ECRReport	If there are no tags in this ECSpec, write a report anyway.	ECReportSpec.setIsEssential(false) ECReportSpec.setReportIfEmpty(true)
Omit from ECRReport	If there are no tags in this ECSpec, do not write a report.	ECReportSpec.setIsEssential(false) ECReportSpec.setReportIfEmpty(false)
Suppress ECRReport	Reviewers -- what does this mean?	ECReportSpec.setIsEssential(true) ECReportSpec.setReportIfEmpty(false)
Application Data		
Name	Description	Corresponds to API Feature
n/a	Free form data to include in this report	ECReportSpec.setApplicationData(String)
Include Filters		
Reviewers: this text is from a while ago -- need help on what parts may still be relevant. Indicates which tags to include in this report. If you specify anything in the Include section, you are saying that in order to be included in the report, a tag must (a) match at least one include filter; AND (b) match none of the exclude filters. Exception: if you specify NO include filters, then condition (a) does not apply. Specifying no include filters is equivalent to specifying an include filter that matches everything. For detailed information about the syntax to use here, see pdfparatexton page -Error! Bookmark not defined.		
Name	Description	Corresponds to API Feature
New Filter	Reviewers -- need info	all patterns are maintained in a list – clients can add individual patterns, but can't remove them individual unless they get/set on the entire pattern list as follows: <pre>List pats = ECReportSpec.getIncludePatterns() // mod pats ECReportSpec.setIncludePatterns(pats)</pre> This pattern is advised for all filter modification (new,delete) ops below.
Delete Filter	Reviewers -- need info	
Exclude Filters		
Name	Description	Corresponds to API Feature
New Filter	Creates a new exclude filter.	ECReportSpec.get/setExcludePattern

Delete Filter	Deletes an exclude filter	
Tag Format	Reviewers -- need info.	N/A (comment applies to this entry and the next 4): This is a function of the AdminConsole. Developers wishing to encode/decode their own URI patterns will find the detailed rules in the EPC Global Tag Data Standards document at (referencece here.
Value	Reviewers -- need info. Any: ... Specific Value: ... Range:	N/A
Company Prefix	Reviewers -- need info. Any: ... Specific Value: ...	N/A
Item Reference with Indicator	Reviewers -- need info. Any: ... Specific Value: ... Range:	N/A
Serial Number	Reviewers -- need info. Any: ... Specific Value: ... Range:	N/A

Subscriber Detail Frame

Action Menu		
Name	Description	Corresponds to API Feature
Subscribe	Create a subscriber with the information from subscriber detail frame.	ALE.subscribe(ecSpecName, uri)
Cancel	Do not create a subscriber.	N/A
Help Menu		
Name	Description	Corresponds to API Feature
ECSpec Frame Help	Displays JavaHelp for the ECSpec detail frame.	n/a
About ALE Administration Console	Displays version and copyright information.	n/a

Menus

Fields Present for all Subscriber Types

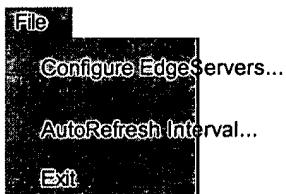
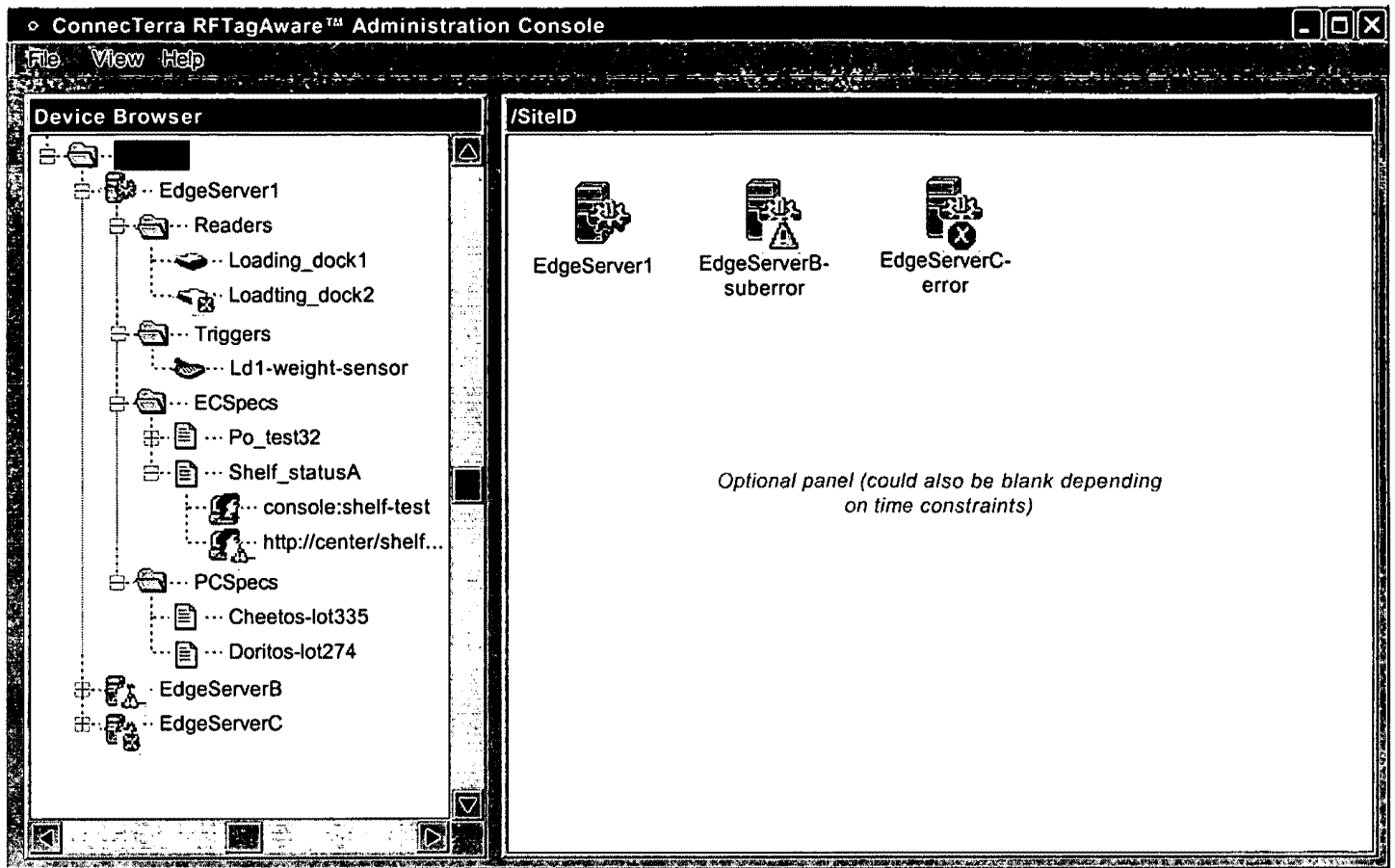
Name	Description	Corresponds to API Feature
<u>Subscriber Type</u>	Drop down menu. Can be: <u>XML on Edge Server Console</u> <u>XML via HTTP POST</u> <u>XML via JMS Message</u> <u>XML File on Edge Server</u> <u>Other</u>	N/A (comment applies to all remaining Subscriber Types entries): This is a function of the AdminConsole. Developers wishing to encode/decode their own URI subscribers will find the detailed rules in the Programmer's guide document at (reference here. – this is our doc, look for section on setting Notification URIs)
<u>Heading</u>	<u>Subscriber report heading.</u>	
<u>Failure Action</u>	<u>Use EdgeServer's defaults: Reviewers -- need info. From edge.props?</u> <u>Override EdgeServer's defaults:</u> <u>Unsubscribe after ... consecutive failures to deliver reports.</u> <u>Specify the number of consecutive failures need to unsubscribe.</u> <u>Unsubscribe after ... seconds of</u>	

	<u>consecutive failures to deliver reports.</u> <u>Specify the number of seconds of consecutive failures needed to unsubscribe.</u>	
--	---	--

Fields Specific to Different Subscriber Types

XML on EdgeServer Console		
Name	Description	Corresponds to API Feature
Heading	Subscriber report heading. Use the format: <code>console:report_name</code> For example: <code>console:MyReport</code>	
XML via HTTP POST		
Name	Description	Corresponds to API Feature
URL	The HTTP POST address. Use one of these formats: <code>http://host:port/remainder-of-URL</code> <code>https://host:port/remainder-of-URL</code> where host is the DNS name or IP address of the host where the receiver is listening for incoming HTTP connections. port is the TCP port on which the receiver is listening for incoming HTTP connections. The port and the preceding colon character may be omitted, in which case the port defaults to 80. remainder-of-URL is the URL to which the HTTP POST operation will be directed.	
XML via JMS Message		
Name	Description	Corresponds to API Feature
Host	Host name at which the JNDI Naming Service is available.	
Port	The port number at the JNDI host.	
Topic	Select either Topic or Queue. If you select Topic, specify the name of the topic that the <code>JMSNotificationDriver</code> will publish to.	
Queue	Select either Topic or Queue. If you select	

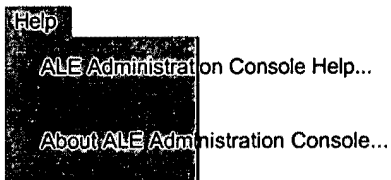
	Queue, specify the name of the queue that the <code>JMSNotificationDriver</code> will add to.	
Connection Factory	The JNDI name of the connection factory for obtaining a topic connection or a queue connection. Use a topic connection factory name when publishing to a topic. Use a queue connection factory name when adding to a queue.	
User	Optional. User name to create a topic or queue connection.	
Password	Optional. Password to create a topic or queue connection.	
Naming Context Factory	Optional. The initial context factory class name to use when performing JNDI lookups. If specified, it overrides the default specified in the <code>edge.props</code> file.	
Security Principal	Optional. The security principal associated with the JNDI Naming Service.	
Security Credential	Optional. The security credential associated with the security principal.	
Message Properties	Optional. These are added to the <code>javax.jms.TextMessage</code> as <code>String</code> properties. Name is the text message property name. Value is the string value.	
XML File on EdgeServer		
Name	Description	Corresponds to API Feature
Directory or File	Use the format: <code>file:///filename</code> where filename is either the name of a file, a directory, or a pattern as described in the <i>RFTagAware Programmer Guide</i>.	
Other		
Name	Description	Corresponds to API Feature
URI	The URI for the notification handler.	



Controls how often to automatically refresh the display.



Manually refreshes the currently visible information from the EdgeServer(s).



Launches a PDF document describing the current frame.

Configure EdgeServers

File Edit Delete

EdgeServerB	http://warehouse:6061
EdgeServerC	http://warehouse:6062

OK

Edit EdgeServer

Name:

Connection URL:

OK Cancel

Internally, this may be appended with 1 or more service specifiers, such as "/axis/service/ALEService", or "/axis/service/TelemetryService"

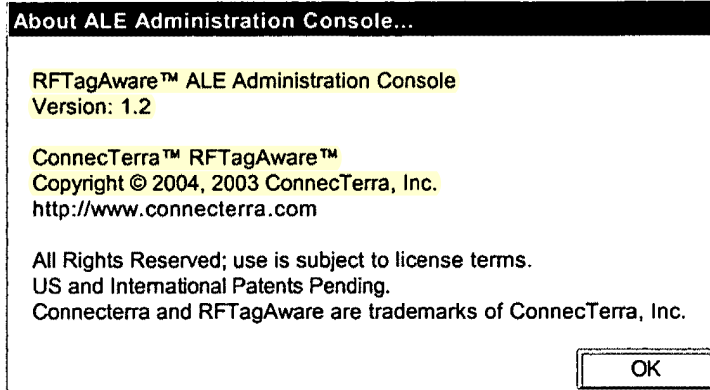
AutoRefresh Interval

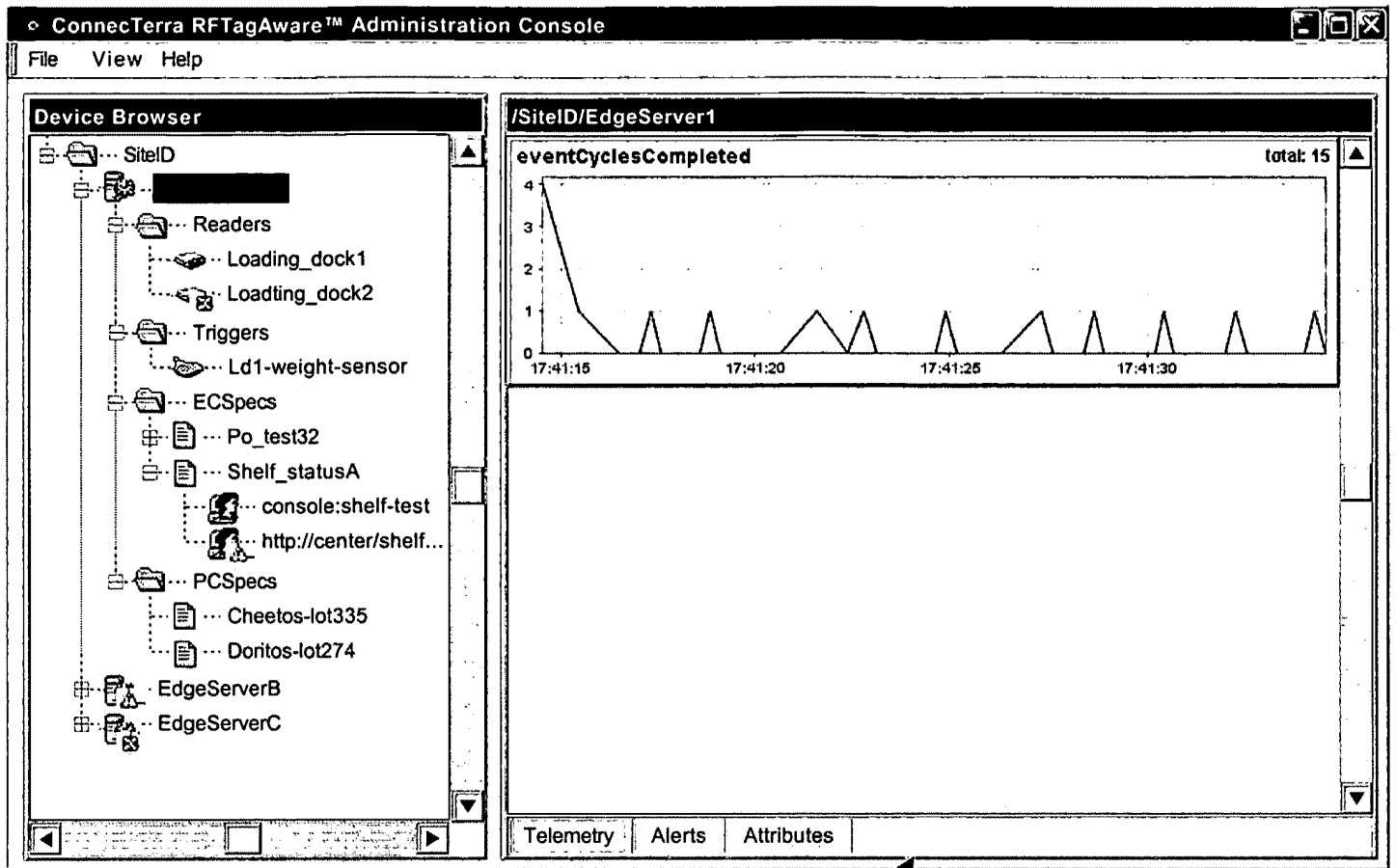
This setting controls how often certain status information is refreshed from configured EdgeServers. If the interval is greater than 0, then the application automatically refreshes the current view every interval. If the interval is 0, then the user must manually select View:Refresh in order to refresh the current view.

Automatic Refresh may be useful when monitoring RFTagAware deployments. See the User Guide for details.

Automatic Refresh Interval: ms

OK





Nav tabs may be either bottom or top of this display.
We should decide which looks better when we can
see the implementation.

ConnectTerra RFTagAware™ Administration Console

File View Help

Device Browser

- SiteID
 - Readers
 - Loading_dock1
 - Loading_dock2
 - Triggers
 - Ld1-weight-sensor
 - ECSpecs
 - Po_test32
 - Shelf_statusA
 - console:shelf-test
 - http://center/shelf...
 - PCSpecs
 - Cheetos-lot335
 - Doritos-lot274
 - EdgeServerB
 - EdgeServerC

/SiteID/EdgeServer1

Importance	Date	Message
SEVERE	2004-03-23 17:53:02 EST	[fake]reader offline!
WARNING	2004-03-23 17:53:02 EST	[fake]reader connection interrupted this is a really long ...
INFO	2004-03-23 17:53:02 EST	[fake]reader scan in progress
FINE	2004-03-23 17:53:02 EST	[fake]reader cycle #242 complete
FINER	2004-03-23 17:53:02 EST	[fake]reader 249 subsystem call returned -1
FINEST	2004-03-23 17:53:02 EST	[fake]reader mov # \$A9, \$3C

WARNING: 2004-03-23 17:53:02 EST

[fake]reader connection interrupted this is a really long status message with lots of details. Rather than show this extended text in a popup field (slightly ugly way of doing it), we're showing it in a preview pane now. The vertical separator above is draggable.

Telemetry Alerts Attributes

◦ ConnectTerra RFTagAware™ Administration Console

FileViewHelp

Device Browser

SiteID

Readers

Loading_dock1

Loading_dock2

Triggers

Ld1-weight-sensor

ECSpecs

Po_test32

Shelf_statusA

console:shelf-test

http://center/shelf...

PCSpecs

Cheetos-lot335

Doritos-lot274

EdgeServerB

EdgeServerC

/SiteID/EdgeServer1

Attribute Name	Value
Device ID	/SiteID/EdgeServer1
Device Type	Edge
manufacturer	ConnectTerra
version	0.1

TelemetryAlertsAttributes

ConnectTerra RFTagAware™ Administration Console

FileViewHelp

Device Browser

SitelD

EdgeServer1

Readers

Loading_dock1

Loading_dock2

Triggers

Ld1-weight-sensor

ECSpecs

Po_test32

Shelf_statusA

console:shelf-test

http://center/shelf...

Cheetos-lot335

Doritos-lot274

EdgeServerB

EdgeServerC

/SiteID/EdgeServer1 - PCSpecs

NewEditDeleteImportActivate Once

Name	Activation Count	Last Activation	Last Reported	Subscriber	Cache Size
Cheetos-lot335	2	2004-03-16 15:36:09 EST	2004-03-16 15:36:09 EST	0	300

PCSpec "Doritos-lot274" - Subscribers

NewEditDelete

URI	Last Succeeded	Consecutive Failures
console:shelf-test	2004-03-16 15:46:09 EST	0
http://center/shelf-check-app	2004-03-16 15:46:09 EST	2

New ECSpec

Edit

Delete

Import ECSpec...

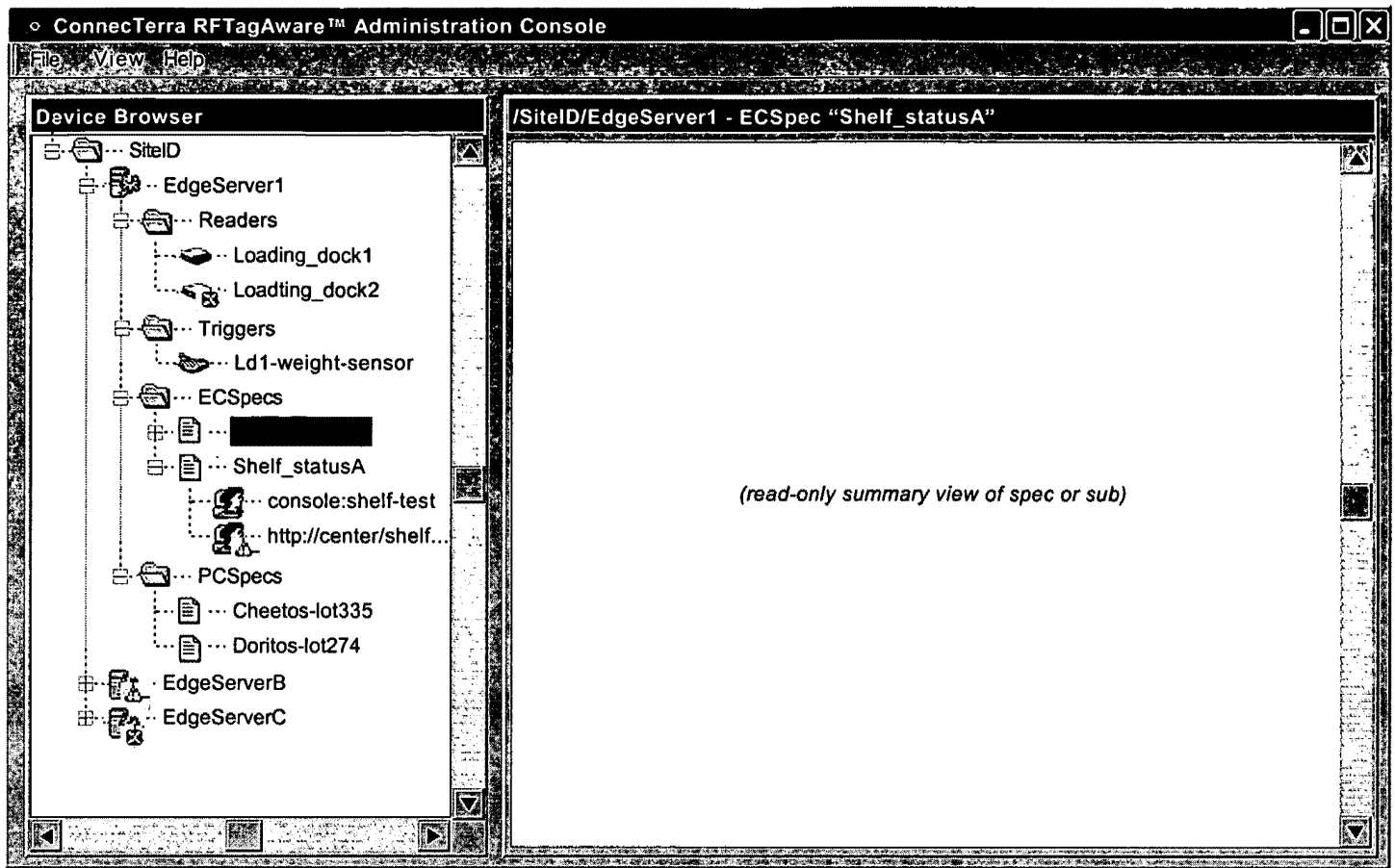
Activate Once

New Subscriber

Edit

Delete

Adasa Inc.
Exhibit 2011
IPR2024-01416
Page 82



/SiteID/EdgeServer1 - ECSpec "Untitled" (undeployed)

/SiteID/EdgeServer1 - ECSpec "Shelf_statusA" (undeployed)

o /SiteID/EdgeServer1 - ECSpec "Shelf_statusA"

File Tools Help

▼ Logical Readers

Available:

backgate
frontgate
Ld1-antenna1
Ld1-antenna2
Ld2-antenna1
Ld2-antenna2

Selected:

frontgate

▼ Boundary

▼ Start

☒ Continuous

☐ Repeat Period: Milliseconds

☐ Trigger URI:

▼ Stop

☒ Duration: ☒ Milliseconds ☐ Read Cycles

☐ Stable Field Interval: ☐ Milliseconds ☐ Read Cycles

☐ Trigger URI:

▼ Application Data

PRIORITY: HIGH

NOTE: This shipment should be routed through Atlanta.

File

View As Text
View As XML
Revert
Export...
Close

Tools

Test
Redeploy
Deploy As...

Help

ECSpec Frame Help...

About ALE Administration Console...

▼ Report Specifications (1)

 New Report

▼ Report: Doritos_infield

 Delete Report

▼ Report Type

☒ Count ☐ List

▼ Report Set

☒ Current ☐ Additions ☐ Deletions

▼ Behavior when Empty

☐ Include in ECRReport
☒ Omit from ECRReport
☐ Suppress ECRReport

▼ Application Data

PRIORITY: HIGH
NOTE: This shipment should be routed through Atlanta.

▼ Include Filters (1)

 New Filter

► Filter: SGTIN-64, ****

 Delete Filter

▼ Exclude Filters (0)

 New Filter

▼ Filter

 Delete Filter

Tag Format: SGTIN-64

Value

☒ Any
☐ Specific Value:
☐ Range: through

Company Prefix

☒ Any
☐ Specific Value:

Item Reference with Indicator

☒ Any
☐ Specific Value:
☐ Range: through

Serial Number

☒ Any
☐ Specific Value:
☐ Range: through

Collapsed version of Filter entry. (see exclude filters below for expanded version)

Other is also an option... presents URI input to user.

Depending on the format selected, 3 to 4 of these subsections may be visible. See EPC global spec for details.

Action
Subscribe
Cancel

Help
Subscriber Frame Help...
About ALE Administration Console...

This section is at the bottom of each variant below.

/SiteID/EdgeServer1/ECSpecs/Shelf_statusA - Subscriber

Action Help

Subscriber Type: XML on EdgeServer Console

Heading:

Failure Action

☒ Use EdgeServer's defaults.

☐ Override EdgeServer's defaults:

☐ Unsubscribe after consecutive failures to deliver reports.

☐ Unsubscribe after seconds of consecutive failures to deliver reports.

Subscriber Type: Other

URI:

Subscriber Type: XML via HTTP POST

http://

Subscriber Type: XML via JMS Message

Host: Port:

☒ Topic:

☐ Queue:

Connection Factory:

☐ User Password:

☐ Naming Context Factory:

☐ Security Principal: Security Credential:

Message Properties

New Delete

Subscriber Type: XML File on EdgeServer

Directory or File:

TextView Frame

(titlebar naming examples)

/SiteID/EdgeServer1/ECSpecs/Po_test32 - Text

ECSpec Frame:File:"View As Text..." is called.

/SiteID/EdgeServer1/ECSpecs/Po_test32 (undeployed) - Text

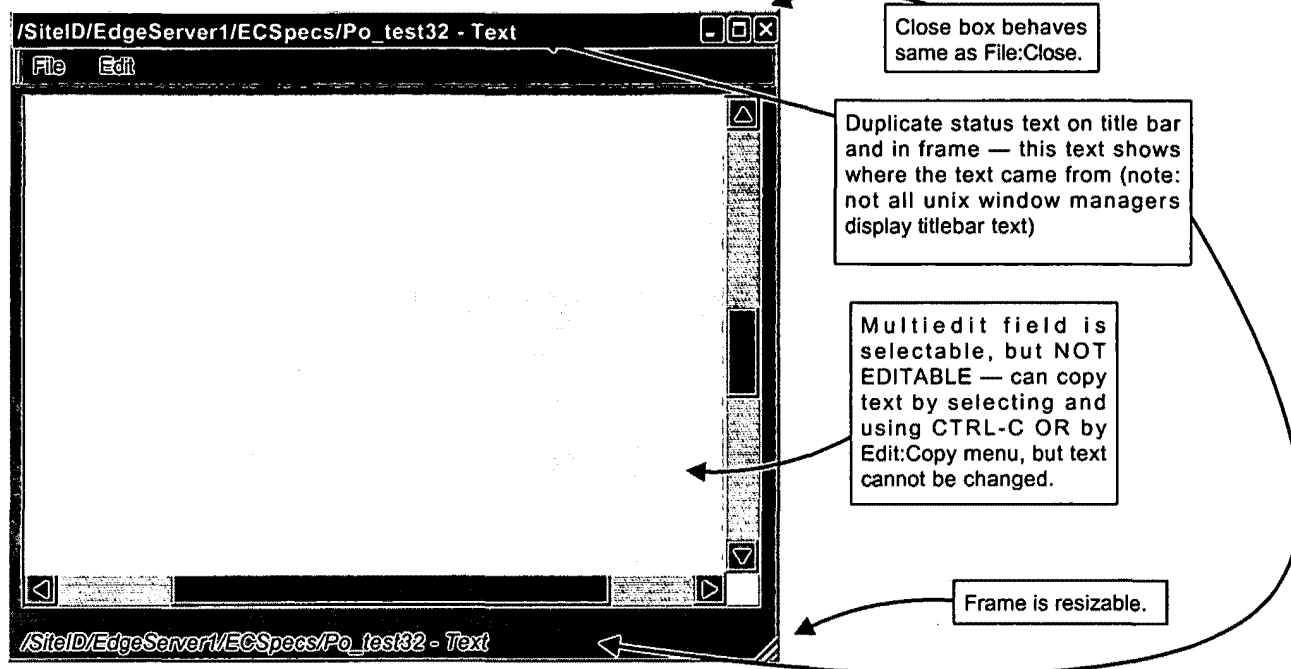
ECSpec Frame:File:"View As Text..." is called on a (undeployed) ecspec.

/SiteID/EdgeServer1/ECSpecs/Po_test32 - ECRports @ 19:32:04 EST

Other variations.

/SiteID/EdgeServer1/ECSpecs/Po_test32 (undeployed) - ECRports @ 19:32:04 EST

/SiteID/EdgeServer1/ECSpecs/Untitled (undeployed) - ECRports @ 19:32:04 EST



File

Save As...

Print...

Close

Opens "Save As Text File..." dialog. User selects file and path. Text is saved to file at path location.

Opens "Print..." dialog. Prints text to printer.

Closes Text View Frame.

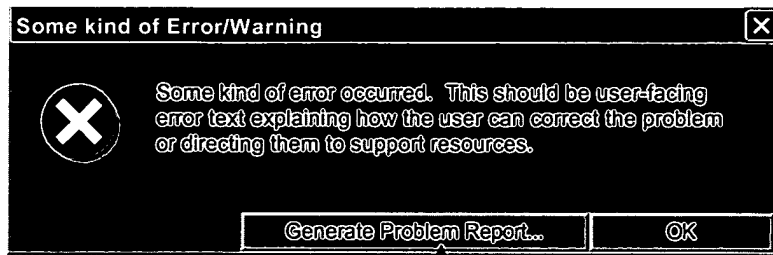
Edit

Copy

Select All

Copies selected text to clipboard for pasting into other applications.

Selects all text.



Selecting this button opens a file dialog that allows the user to specify where a problem report text file is written — this file may be sent to our support resources for debugging or solutions.

Attributes: --

☒ Include EC Spec in each EC Report

Stop:

☒ Duration (Milliseconds):

☒ Duration (Read Cycles):

☒ Stable Field Interval (Milliseconds):

☒ Stable Field Interval (Read Cycles):

☒ Trigger URI:

Group Patterns

Protocol: SGTIN-64

1 X [0-99] *

Add

[SGTIN-64] 2.2.*.*
[SSCC-96] 2.[3-99].*.*
[SSCC-96] [3-255].X.*.*

Remove

Edit

View as XML

View as Text

Report Type:

☒ Count ☒ List

List Items

☒ Include EPC ☒ Include Tag

☒ Include Raw Hex ☒ Include Raw Decimal

Omit Report

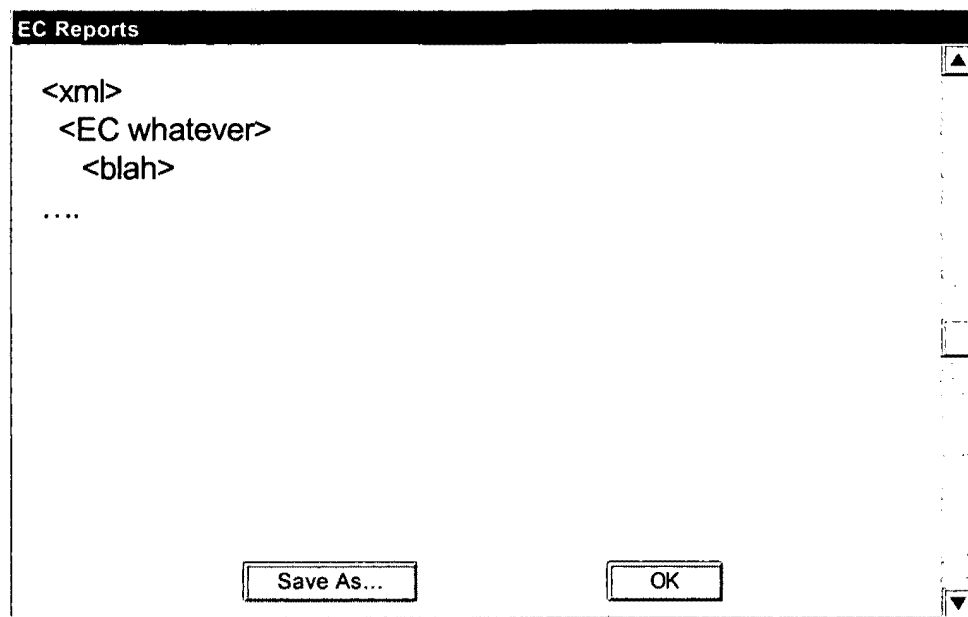
☒ Omit if empty

☒ Omit if unchanged from last cycle

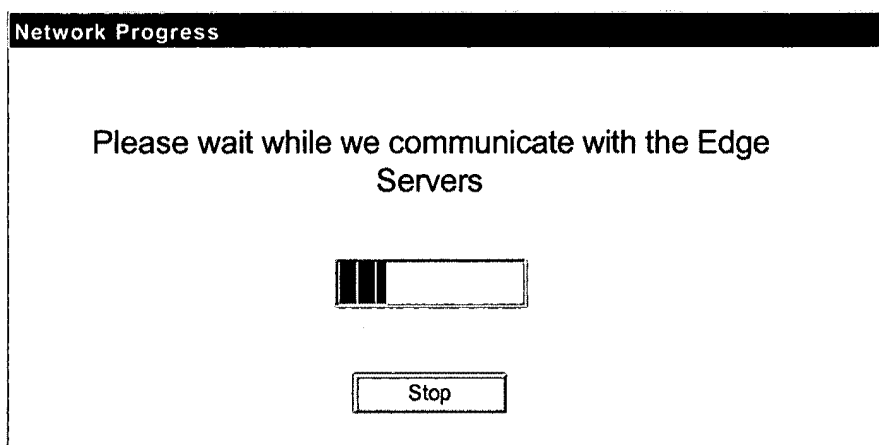
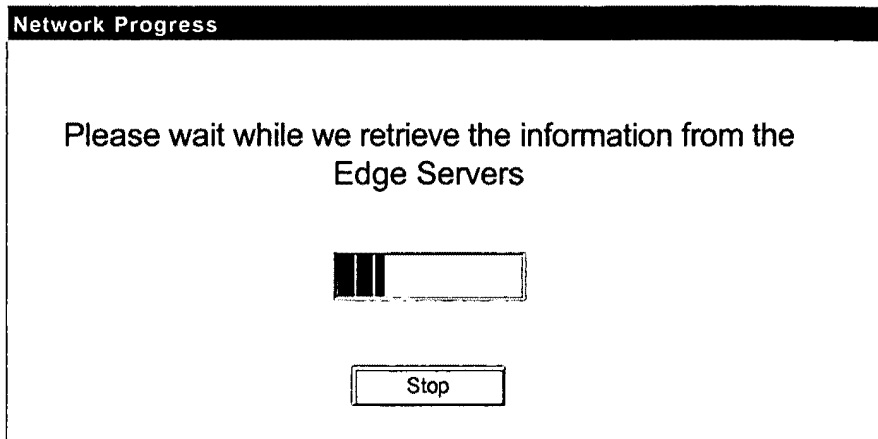
☒ If omitted, suppress all EC Reports this cycle

If List is not checked, the List Items section is grayed out. If the user attempts to Save the Report with neither "Count" or "List" checked, then they get an error message saying "You must select at least one of Report Type. If the user attempts to save with List checked, but none of the List Items check boxes checked, then they get an error message saying "You must select at least one Tag Format for the List Items". All four boxes are checked by default.

The "If omitted, suppress" check box grays out and appears unchecked if both of the other checkboxes are unchecked. If either are rechecked, then the check box is reenabled and it's previous state (checked or unchecked) is restored.



Currently, EC Reports appear in a small message window. In this release, the EC Report will come up in a dialog box instead. This dialog is non-modal, and the user can select the text to copy it into the copy buffer.





ALE GUI Controller and Model Design

Version of 2005-04-12 at 11:26

This document is Company Confidential

Table of Contents

ALE GUI Controller and Model Design	1
1 Introduction	3
2 Key Terminology	3
3 Feature Overview	4
4 Use Cases	4
5 Design Overview	4
6 Design Detail	6
6.1 com.connectterra.admin.endpoint package	6
6.1.1 Endpoint	6
6.1.2 ConsoleConfigurationManager	6
6.2 com.connectterra.admin.gui.consoleGUIModel package	6
6.2.1 ConsoleGUIModel	7
6.2.2 EdgeIDTreeModel	7
6.3 com.connectterra.admin.gui.edgeGUIModel	7
6.3.1 EdgeGUIModel	8
6.3.2 EdgePluginRole	8
6.3.3 EdgePlugin	9
6.3.4 Reader	9
6.4 com.connectterra.admin.gui.aleGUIModel	9
6.4.1 ALEGUIModel	9
6.4.2 ECSpecsModel	9
6.5 com.connectterra.admin.aleModel	10
6.5.1 Overview	10
6.5.2 ALEController	11
6.5.3 ALEConnectionFactory	11

6.5.4	ALEModel.....	12
6.5.5	ECSpecsModel	12
6.5.6	ALEListener	12
6.6	com.connecterra.admin.gui.telemetryGUIModel	13
6.6.1	TelemetryGUIModel	13
6.6.2	DeviceTelemetryModel	13
6.6.3	Alert and DeviceAttribute	14
6.6.4	DeviceTelemetryListener	14
6.7	com.connecterra.admin.telemetryModel	14
6.7.1	TelemetryController	15
6.7.2	TelemetryModelUnsubscriptionHandler	15
6.7.3	TelemetryHTTPServer	16
6.8	com.connecterra.admin.gui.configGUIModel	16
6.8.1	ConfigGUIModel	16
6.8.2	PhysicalReader	17
6.8.3	CompositeReader	17
6.9	com.connecterra.admin.gui.configModel	17
6.9.1	ConfigModel	17
6.9.2	ConfigController	17
6.9.3	ConfigListener.....	17
7	External Changes.....	17
7.1	Specific add/update to Java API.....	17
7.2	Specific add/update to XML API	17
7.3	Specific add/update to Web Services API.....	17
7.4	Specific add/update to Configuration	18
7.5	Specific add/update to Scripts	18
7.6	Specific add/update to GUI	18
7.7	Specific add/update to Command Line Interface	18
7.8	Specific add/update to Protocol.....	18
7.9	Specific add/update to Syntax	18
7.10	Specific add/update to Database/Persistence.....	18
7.11	Specific add/update to Installer	18
7.12	Specific add/update to Samples.....	18
7.13	Specific add/update to jar files and other libraries	18
7.14	Security considerations.....	18
7.15	Compatibility Considerations	18
7.16	Performance Considerations.....	18

7.17	Testing considerations	18
7.18	Deployment considerations	18

Revision history: latest at the top please			
Version	Author	History	Date(yyyy-mm-dd)
012	Chris Shabsin	add setECSpecInfos to ALEModel, more feedback on Config stuff	2005-04-12
011	Chris Shabsin	complete initial Config model/controller design	2005-04-12
010	Chris Shabsin	incorporate feedback from the design review	2005-04-11
009	Chris Shabsin	finish first draft of everything except ConfigModel	2005-04-05
008	Chris Shabsin	flesh out DeviceTree, ConsoleGUIModel, EdgeGUIModel, and ALEGUIModel	2005-03-29
007	Chris Shabsin	minor tweaks – review version	2005-03-10
006	Chris Shabsin	updated document to current design thoughts	2005-03-08
005	Chris Shabsin	remove WorkerQueue from this document (moved to WorkerQueue design.doc)	2005-03-07
004	Pete Gast	Revise ALEWrapper and SwingWorker subclasses	2005-02-17
003	Pete Gast	Add Requirements	2005-02-16
002	Marc Horowitz	Revise WorkerQueue design	2004-07-02
001	Marc Horowitz	Initial version	2004-06-22

1 Introduction

This document describes a part of the design for the ALE GUI. Specifically, it provides a controller and model design on which the ALE GUI views are to be built. These components may also be reused by other user interfaces which make use of the RFTagAware ALE and telemetry APIs.

2 Key Terminology

This section lists key terms and definitions as utilized in the design:

- **Model** – A data container that holds a view-independent version of system information. Often, models notify registered listeners when changes have occurred in the underlying data.

- **Controller** – An object that provides a mechanism for getting data into the Model in an application-specific way. The Controller is the link between the Model/View and the real world.
- **Telemetry** – The RFTagAware Edge Server has a Telemetry API that provides device status and time-series information on a subscription basis. The client subscribes to a telemetry stream, and the Edge Server begins pushing data to the client, until either the client unsubscribes, or there are enough consecutive delivery failures.
- **Time-series information** – Various devices on the Edge Server export time-series data streams via the Telemetry API. These data streams are either time/value pairs, or time/gauge pairs, and the names of these streams are device-specific.
- **Gauge** – a stock-ticker style data point with a high-water mark, low-water mark, and average value for any given time.

3 Feature Overview

The design encompasses separate controllers and models for the ALE API and the Telemetry API. They share a few classes which describe edge servers, since this is the common point of reference for both APIs.

Each controller provides methods which can be used to monitor changes to some edge server data. The interface hides details such as whether a publish/subscribe model or a polling model is used, so as protocols are enhanced, the GUI can work more efficiently without making changes to the view code.

Each controller also provides access to their underlying connection pools, to provide managed access to server connections for other tasks.

4 Use Cases

The Admin Console GUI has panels that monitor the state of the edge server and its constituent devices. The device browser maintains a graphical display of device status in its tree view; the display must update asynchronously, when the edge server pushes new telemetry information to the console application.

When certain device nodes are selected, the device information panel displays the device's associated time-series telemetry charts. When the user selects such a node, the application must send the edge server a telemetry subscription request, and when the user deselects the node, the application must unsubscribe.

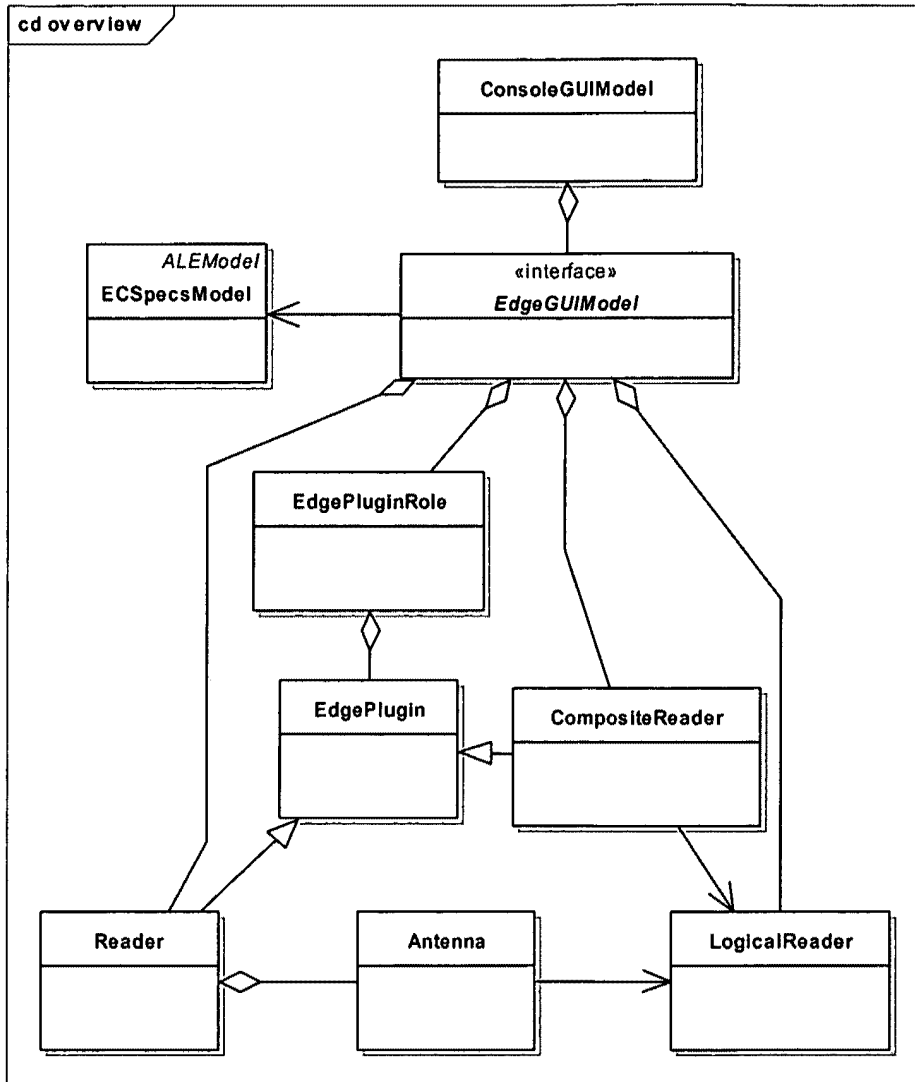
The ECSpec information panel displays an edge server's ECSpecs in a table, and updates that information periodically. The ALEController provides a subscription-like interface to this information so that the pattern of subscribing when the "ECSpecs" node is selected and unsubscribing when leaving the node parallels the device telemetry stream pattern.

5 Design Overview

The design of the ALE GUI must take into account a complex set of requirements.

- The GUI must be able to manage multiple edge servers. This management uses several different APIs (ALE and Telemetry, and possibly others in the future)
- For good user interaction, the GUI must never appear to block. Instead, any long-lived operation must be performed on a separate thread.
- For efficient use of resources, multiple calls to the same API on the same edge server should share a single connection.
- The user must be allowed to cancel operations, even though the underlying network protocol implementation does not always provide a mechanism to interrupt an operation in progress.
- Some of the information needed can be subscribed to, but other information must be polled.

To meet these requirements, this design provides an interface for a local caching model for the ALE API, one for the Telemetry API, and another for the Config API. The model also includes a listener/event mechanism so that the view can automatically updated when the model is changed. A controller provides methods which cause periodic updates to the local model to be made which can then be acted on by listeners.



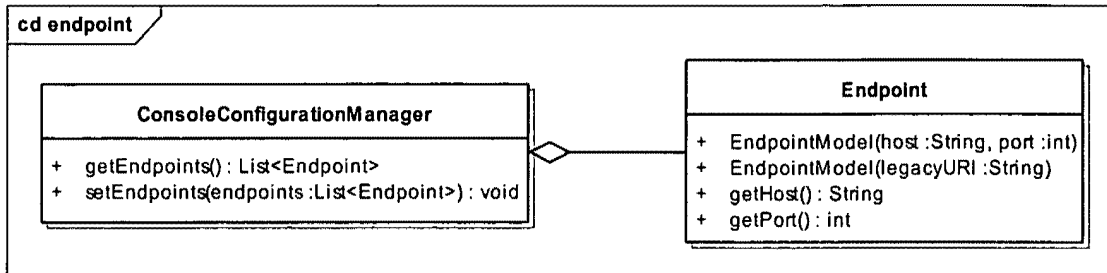
The model exposed by this architecture is a confluence of information and operational methods into one interface that hides the underlying mechanism used to implement the requests. Information from the various edge server interfaces will be merged into this one model; the GUI does not need to know that device status information comes from the telemetry. The model hides the fact that, while the association of logical reader name to physical reader comes from the Configuration API, the capability of that logical reader – whether it can read tags, write tags, or both – is gleaned from the ALE API.

While the ConfigAPIModel exposes Readers and CompositeReaders as plugins, the EdgeGUIModel treats these specially, hiding the reader and composite reader “roles” in favor of specific interfaces for accessing the models. Other plugin-type objects (filters and triggers) are handled through the plugin role interface, since the admin console should be able to handle new types of objects as they are introduced underneath.

6 Design Detail

This design will elaborate on the technical details and infrastructure for supporting the controller/model abstraction for the ALE and Telemetry APIs. The key components to support this subsystem will be considered.

6.1 com.connecterra.admin.endpoint package



6.1.1 Endpoint

The Endpoint model encapsulates the information necessary to connect to an edge server. This is simply a host and port, since the ALE, Config, and Telemetry API URI paths can all be generated from this information. The Endpoint can be constructed from a host/port combination or a string representation. The string representation can be a legacy URI (of the form “http://host:port” or “http://host:port/”), for backwards compatibility with older Admin Console configurations, or simply a host/port pair of the form “host:port”. When the admin console writes the configuration back out, though, legacy URLs will be converted to the new form, “host:port”. The URI-based constructor can throw an `InvalidURIException`, if the String is a malformed URI, or if it is not one of the expected legacy forms.

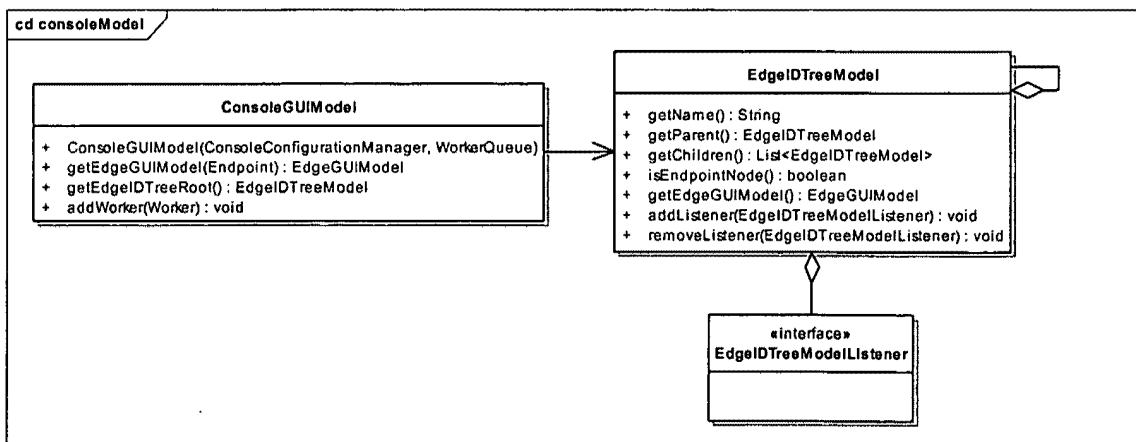
The Endpoint is immutable; to change an edge server endpoint, the endpoint must be removed from the `EndpointList` under the old value, and re-added with the new value.

This object is the key identifier for any edge server connection.

6.1.2 ConsoleConfigurationManager

The Admin Console initialization generates this configuration manager and passes it into `ConsoleGUIModel`. The `ConsoleGUIModel` gets its list of endpoints from this object. The config manager simply handles transfer of the list of endpoints to and from persistence.

6.2 com.connecterra.admin.gui.consoleGUIModel package



6.2.1 ConsoleGUIModel

The ConsoleGUIModel is the top-level Model object for the Admin Console application. It represents the tree structure of the device browser. It also holds the ConfigurationManager passed in from the application. It gets the list of connections to edge servers from that object, and uses it to persist changes to that list.

When an Endpoint is removed from the edge server list, the ConsoleGUIModel is responsible for making sure that the associated EdgeGUIModel disconnects from the server and that its associated node is removed from the device tree.

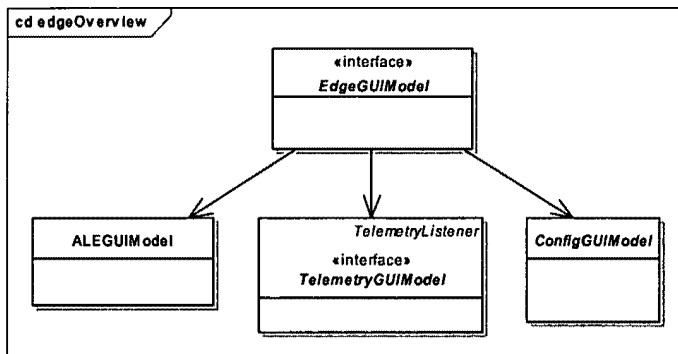
The Admin Console reads the edge list configuration into a ConsoleConfigurationManager and passes that object to the ConsoleGUIModel constructor. The constructor creates a preliminary EdgeIDTreeModel tree structure with each edge server in the “Unknown” path, and creates EdgeModels for each edge server. As the telemetry service returns device path information for each edge server, it updates the EdgeIDTreeModel tree structure to sort those servers into the reported device path.

6.2.2 EdgeIDTreeModel

The EdgeTreeModel is a tree structure maintained by the ConsoleGUIModel. The tree structure mirrors the structure present in the device ID paths returned each Edge Server’s telemetry.

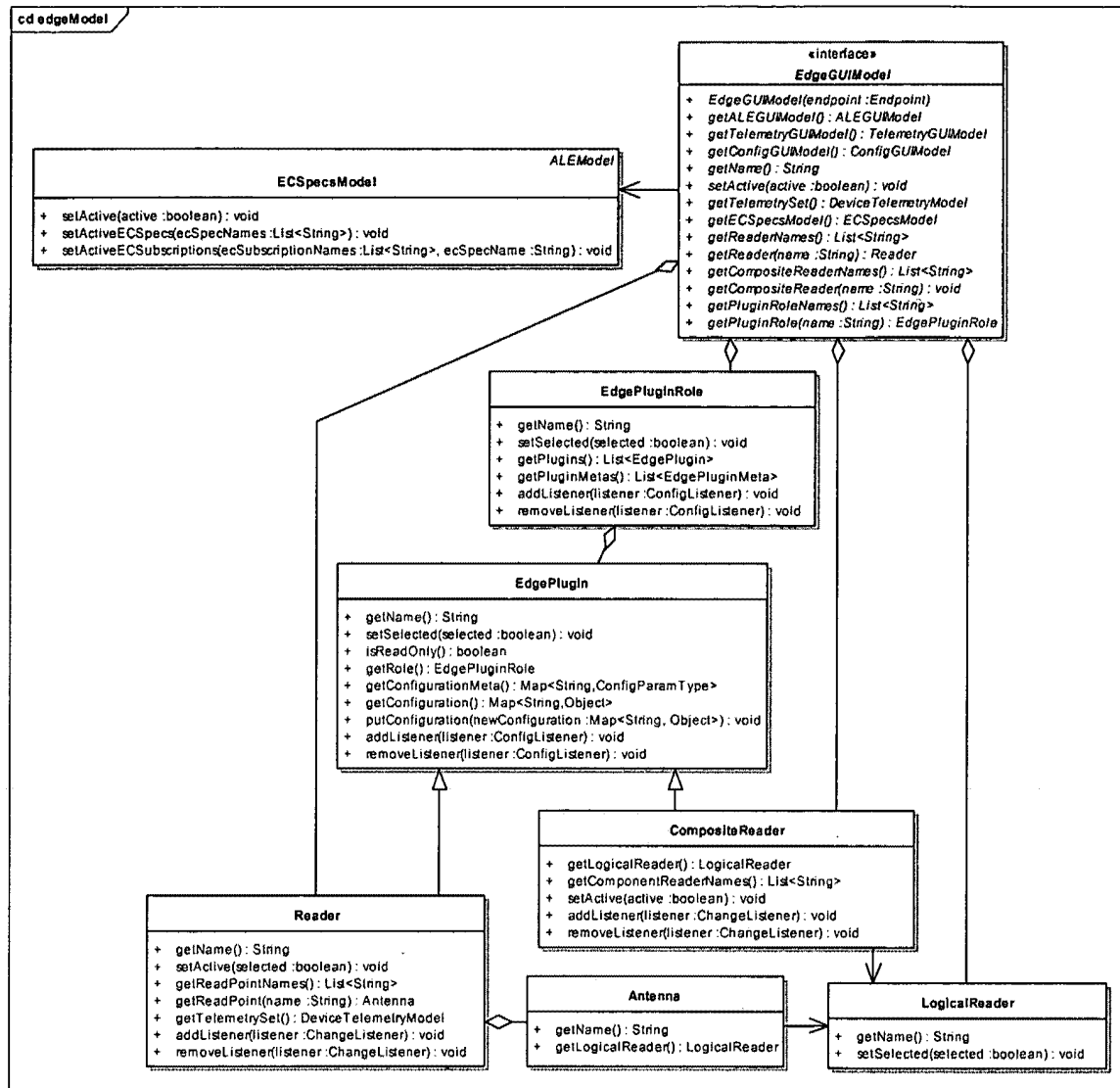
The following invariant applies to this tree – any node with children is not an endpoint node, and has no associated EdgeGUIModel; all leaves are endpoint nodes that contain a reference to an EdgeGUIModel object.

6.3 com.connecterra.admin.gui.edgeGUIModel



The EdgeGUIModel provides a subtree for the ConsoleGUIModel’s device tree to include. It also consolidates the information from the ALEGUIModel, TelemetryGUIModel, and ConfigGUIModel into one full representation.

6.3.1 EdgeGUIModel



An EdgeGUIModel is constructed to provide access to a single edge server specified by an Endpoint object. It creates an ALEGUIModel, a TelemetryGUIModel, and a ConfigGUIModel with that endpoint and holds those objects.

When the EdgeGUIModel is in “active” mode, it starts the TelemetryGUIModel monitoring the telemetry for the associated Edge Server. The EdgeGUIModel also exposes the TelemetrySetModel that holds the edge server’s “edge device” telemetry.

The EdgeGUIModel provides access to the edge server’s associated devices; readers and composite readers bypass the generic “role” mechanism provided to parallel the ConfigAPI, but other devices are referenced first by their role on the edge server.

Readers are instead accessed through the getReaderNames/getReader interface, since these devices have meaning even to an edge server that does not support the Config API. Composite Readers are the same.

6.3.2 EdgePluginRole

EdgePluginRole represents a type of configurable device the Edge Server exposes via the Configuration API. For Admin Console 1.3 this will simply provide read-only access to filters, per the limitations of the

Configuration API. Readers and composite readers bypass this mechanism, accessed through specific APIs on the EdgeGUIModel; the “reader” and “composite reader” roles are kept hidden at this layer.

6.3.3 EdgePlugin

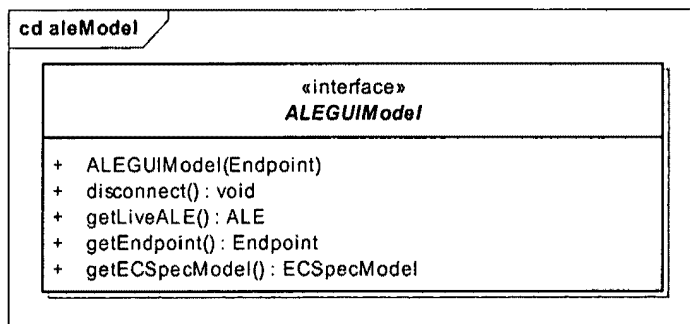
The EdgePlugin represents a configurable device exposed by the server’s Configuration API. The getConfigurationMeta() method returns the Config API’s native PluginMeta object for the associated PluginDevice’s Plugin. See the Dynamic Reader Configuration Design document for more information. This dependency on the Config API is useful because the parameter information structure is complex; it may be useful for these classes to be moved into a more shared component in the future.

6.3.4 Reader

A Reader represents a Physical Reader device on the edge server. The device has configuration information (with accessors inherited from the EdgePlugin object) and a TelemetrySet associated with it. It also has ReadPoints associated with it, each of which has a Logical Reader.

6.4 com.connecterra.admin.gui.aLEGUIModel

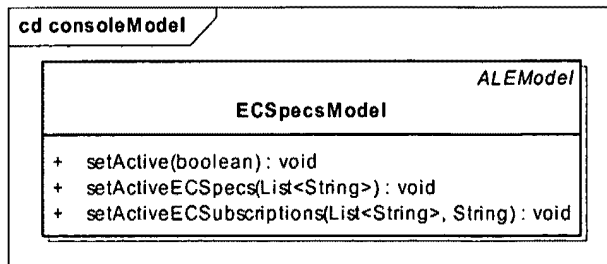
6.4.1 ALEGUIModel



The ALEGUIModel exposes a more UI-centric interface to the information in the ALEController and ALEModel. It provides direct access to the underlying caching ALEModel and the ECSpecModel.

The class also provides direct access to the ALEController’s “Live ALE” object – this object is a wrapper around the ALE interface that automatically checks out a connection, makes the requested call, and then checks the connection back in.

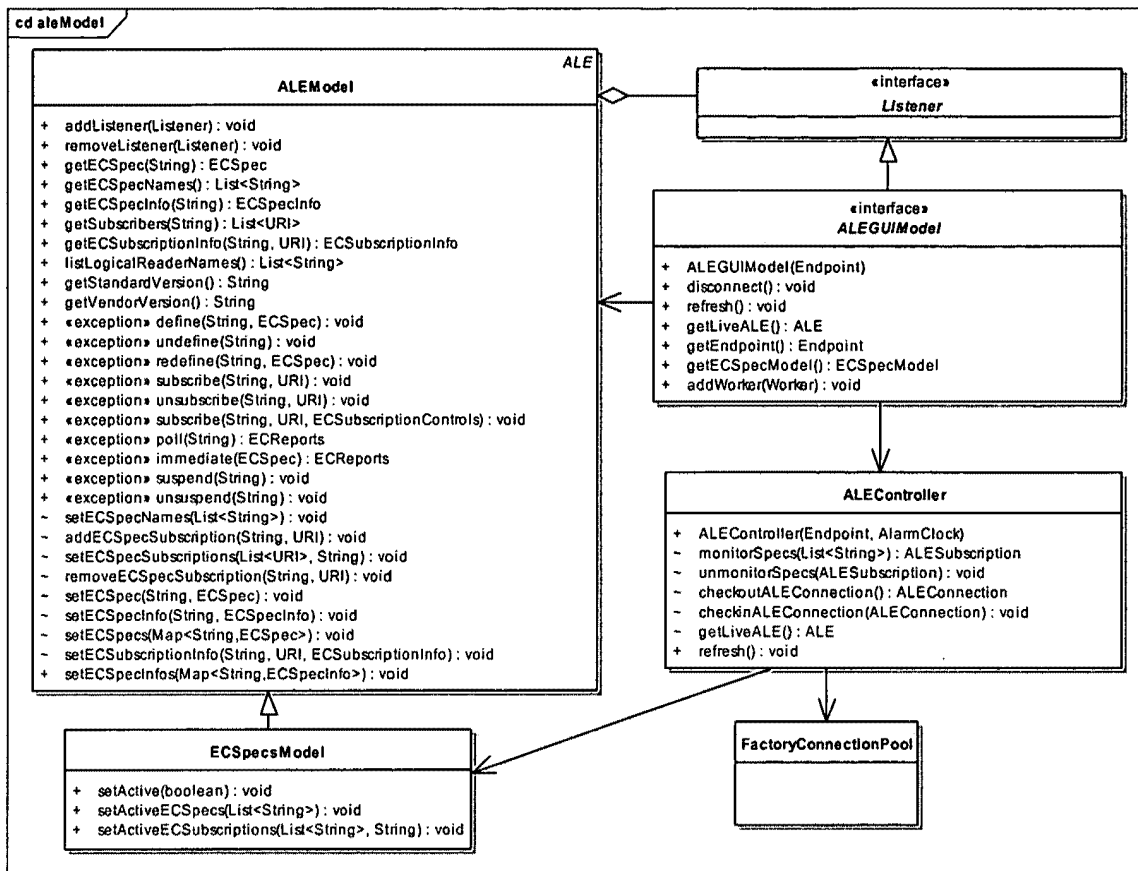
6.4.2 ECSpecsModel



The ECSpecsModel is a thin extension over the ALEModel (which itself implements the ALE interface to provide cached information). The Admin Console informs the ECSpecsModel which ECSpecs are “active” – active ECSpecs have their ECSpecInfo periodically polled and updated in the ECSpecsModel. Similarly, the ALEGUIModel maintains information on which ECSubscriptions are active.

6.5 com.connecterra.admin.aleModel

6.5.1 Overview



The model/controller abstraction over the ALE API is multi-layered, adding a distinct set of new function at each level.

The lowest level, the Connection Pool, enables server connection sharing. The `FactoryConnectionPool` uses an `AxisALEConnectionFactory` to create a simple Axis-based connection to the Edge Server's ALE service and wraps that ALE object in a layer that detects exceptions that are related to the connection to the server. If the connection goes down, that wrapper marks the associated connection object "suspect." Then when the connection is checked back into the connection pool, the pool knows to remove it.

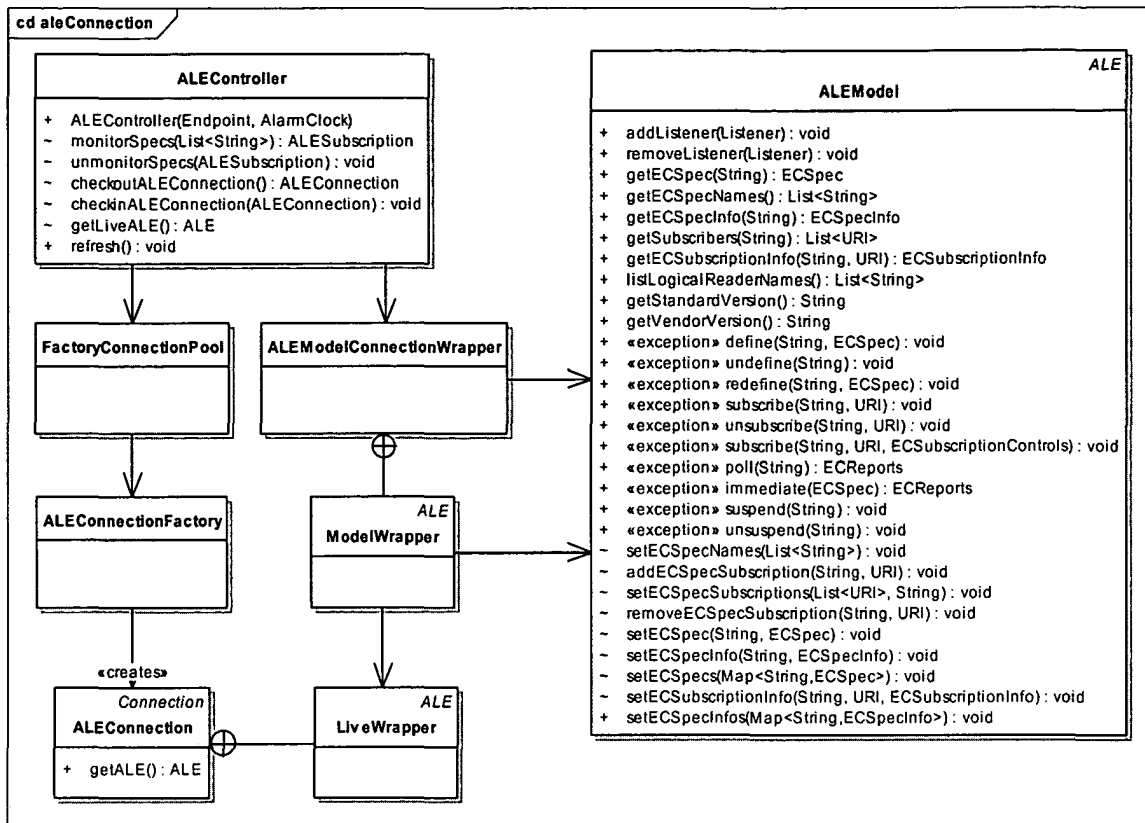
The next level consists of two classes, the `ALEController` and the `ALEModel`. The `ALEController` wraps the Connection's ALE object – the `ALEController`'s wrapper maintains the link between data seen over the ALE connection and the cached information in the `ALEModel`. The `ALEController` also has a polling thread that watches for changes in edge server's ECSpecs.

Operations on `ALEModel` always refer to the cache and never block on network traffic. The `ALEModel` implements certain package-private setter methods for use by the `ALEController`'s ALE wrapper. `ALEModel` calls to define, undefine, redefine, poll, or immediate result in an `IllegalOperationException`.

The `ECSpecModel` is a thin layer over `ALEModel` that provides a connection back to the `ALEConnection`'s polling mechanism.

The next level is the `ALEGUIModel`, which combines the `ALEController` and `ALEModel` into one interface. It exposes the `ALEController`'s live ALE connection, the Worker Queue, and the `ECSpecModel`.

6.5.2 ALEController



This class implements methods for requesting updates to ECSpecs or ECSpecSubscriptions on an edge server. In the first version of the implementation, this will be implemented with a polling thread. If the ALE API is enhanced or a ConnecTerra-private API is developed in the future to support a subscription mechanism for changes in this data, the implementation can be updated to take advantage of that.

The ALEController will use the AlarmClock mechanism from the ConnecTerra Util component to perform this periodic polling.

The ALEController also provides a “Live ALE” object which implements the ALE API by wrapping calls with checkout and checkin operations to the ALE Connection pool. This is useful for callers to perform specific ALE operations, such as creating or deleting ECSpecs, within the ALEController’s connection management framework.

6.5.3 ALEConnectionFactory

The ALEConnectionFactory instance does not return AxisALEClient instances directly. Instead, the AxisALEClient object will be wrapped in another class which implements the ALE interface. For each method in the ALE interface which can alter the model, this wrapper class will update the model, perhaps issuing additional ALE method calls if the arguments and return value are insufficient to understand the model changes. In this way, changes made as a result of user calls are automatically updated in the model, and if appropriate listeners are in place, the view as well.

Additionally, exceptions that indicate an error with the connection will label the associated ALEConnection object as suspect, causing the ConnectionPool to discard the connection and attempt to generate a new one.

The ALEController uses the FactoryConnectionPool to adapt this factory class into a connection pool.

6.5.4 ALEModel

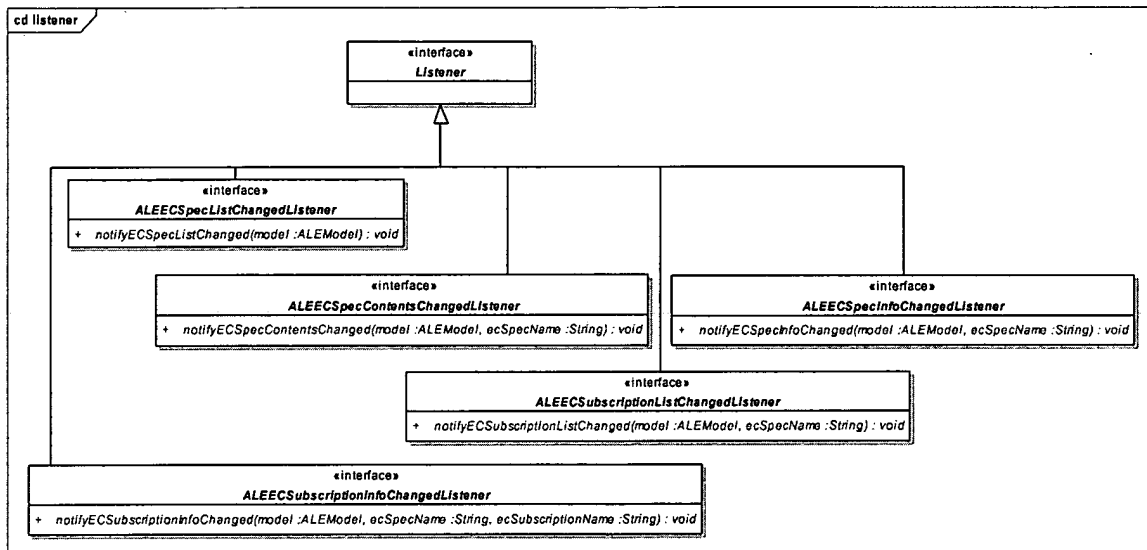
Operations on ALEModel always refer to the cache and never block on network traffic. The ALEModel implements certain package-private setter methods for use by the ALEController's ALE wrapper. ALEModel calls to define, undefine, redefine, poll, or immediate result in an `IllegalOperationException`.

The ALEModel object stores the local cached data for all the edge servers which the system is monitoring. There are methods for adding and removing listeners which receive events whenever the model changes. There are getter methods for the view to query the model in order to build UI objects. And, there are setter methods which are used by the controller to update the model, either as a result of a subscription or poll operation, or by the ALE instance wrapper.

6.5.5 ECSpecsModel

The ECSpecModel is a thin layer over ALEModel that provides a connection back to the ALEConnection's polling mechanism.

6.5.6 Listeners

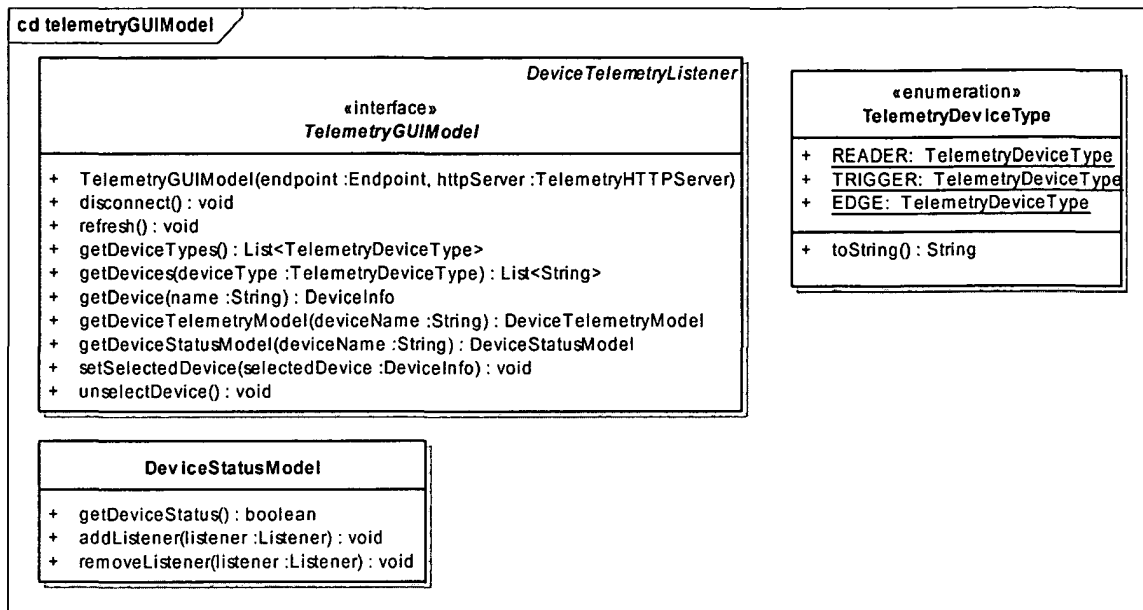


These listener interfaces monitor the ALEModel and ECSpecsModel for changes. Each leaf of the tree listens for a specific type of change.

When the listener is added to a model object, the model maintains lists of differentiated listener interfaces, and adds the listener to each list that matches that type. That way, an object can implement as many or as few of these interfaces as it wishes, but it only has to get passed into the model `addListener` call once to be added to all the relevant internal lists.

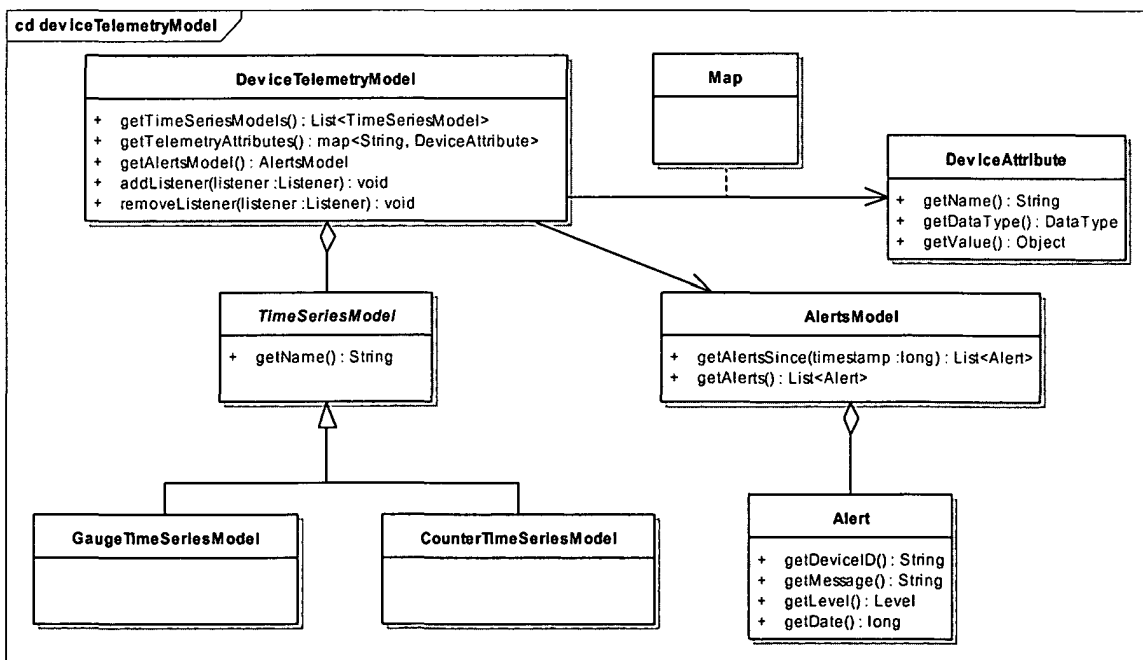
6.6 com.connecterra.admin.gui.telemetryGUIModel

6.6.1 TelemetryGUIModel



The **TelemetryGUIModel** exposes a GUI-specific representation of the telemetry for a given Edge Server. It lists the device types, and the devices in each type. It also provides a **DeviceTelemetryModel** for each device, combining all the telemetry data (alerts, time-series, and attributes) into one object. Only the selected device has a subscription to telemetry data. The **TelemetryGUIModel** subscribes to device status information for all devices, however, and the returned **DeviceStatusModel** for each device maintains its state based on that. When the **TelemetryGUIModel** is disconnected, associated **DeviceStatusModels** stop updating.

6.6.2 DeviceTelemetryModel

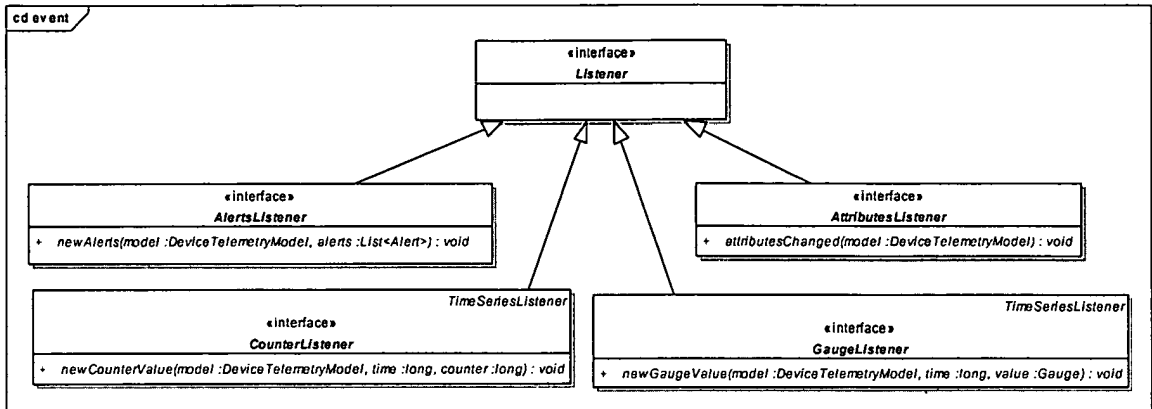


The Device TelemetryModel represents the information the Edge telemetry provides for any given device. The contents of the various models will be fleshed out as needed.

6.6.3 Alert and DeviceAttribute

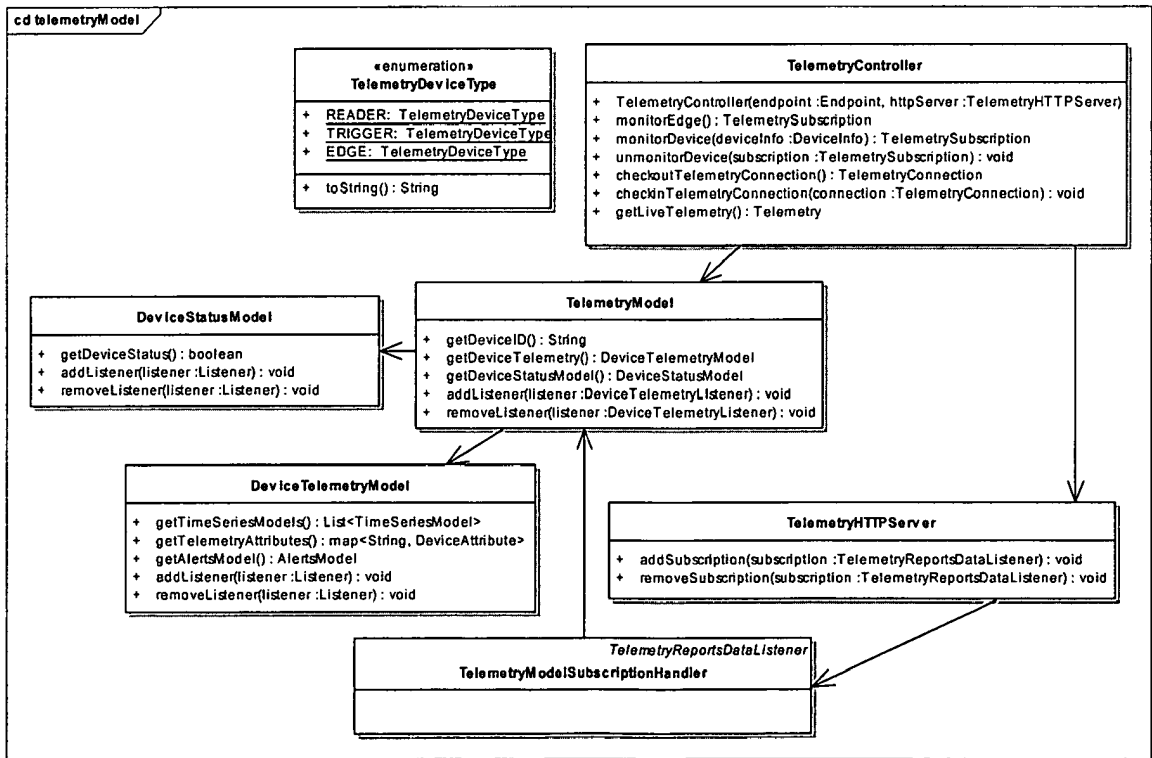
Alert and DeviceAttribute are objects that implement the interfaces from the Telemetry API.

6.6.4 DeviceTelemetryListener



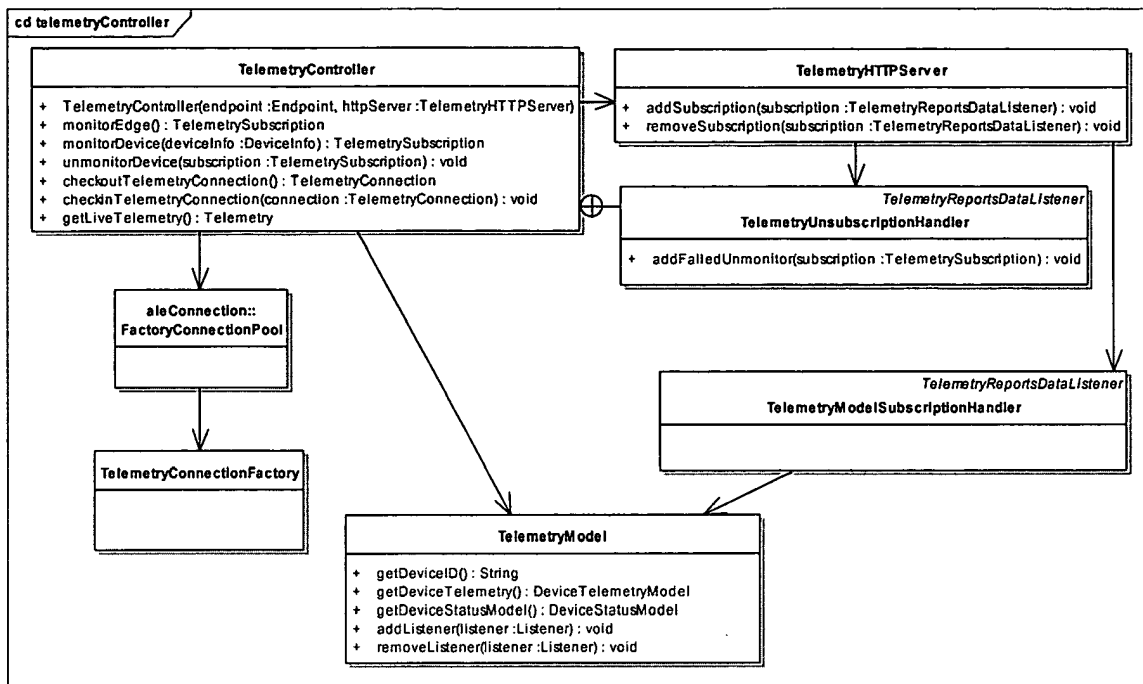
The TelemetrySetListener is a listener for the DeviceTelemetryModel. The DeviceTelemetryListener follows the ALEListener pattern with the specific list of interfaces the listener implements being used to dispatch the appropriate

6.7 com.connecterra.admin.telemetryModel



The Telemetry API is abstracted in a similar way as the ALE API, with succeeding layers adding new functionality to the underlying layer.

6.7.1 TelemetryController



This class is analogous to ALEController. It implements methods for monitoring device telemetry, which are implemented by subscribing and unsubscribing to TelemetrySpecs. This permits the admin console to subscribe just to the telemetry needed to update whatever UI objects are visible to the user. It also includes methods for checking a Telemetry connection object out of the connection pool, and checking it back in when done.

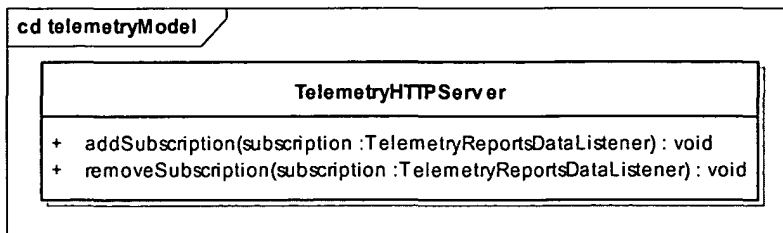
Additionally, the TelemetryController keeps track of failed unsubscription attempts; if telemetry associated with a terminated subscription arrives, it assumes the connection has been reestablished, and it attempts to resend the unsubscribe request.

6.7.2 TelemetryModelUnsubscriptionHandler

When the TelemetryController attempts to unsubscribe to a telemetry stream and fails for some transient reason, it does not expose that failure to its callers. Instead, it passes information about the failed subscription to its contained TelemetryModelUnsubscriptionHandler object. This object subscribes to telemetry information from the HTTP server, and when it recognizes data associated with a failed unsubscription, it attempts again to unsubscribe from that telemetry stream.

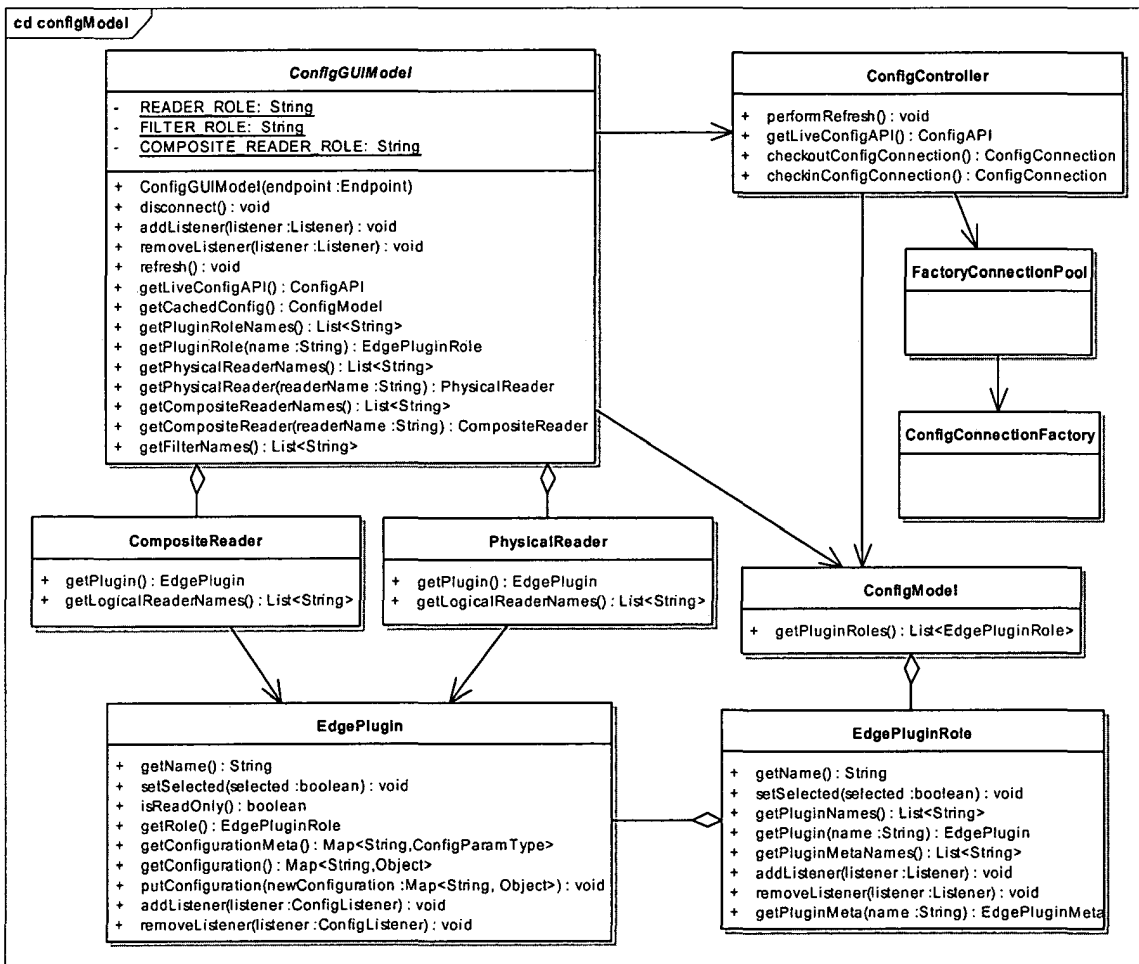
A race condition could arise if the UI attempts to subscribe to a stream that has, during a previous interaction, had a failed unsubscription attempt. In this situation the TelemetryUnsubscriptionListener could potentially unsubscribe shortly after the new subscription is established. To avoid this, the TelemetryController informs the TelemetryUnsubscriptionListener of new subscription requests before making the subscription request. The TelemetryUnsubscriptionListener detects a match in its list of failed unsubscriptions and removes matching entries from its list. The Worker Queue mechanism should be used to serialize these requests, making sure the removal from the Handler's list completes before the new subscription is attempted.

6.7.3 TelemetryHTTPServer



The TelemetryHTTPServer is an HTTP server thread that receives Telemetry streams from Edge Servers, converts the received XML data into TelemetryReports objects (from the TelemetryAPI component) and passes them on to listeners. The TelemetryModel and the TelemetryUnsubscriptionListener are the only two objects that will listen to the streams at this level of abstraction.

6.8 com.connecterra.admin.gui.configGUIModel



6.8.1 ConfigGUIModel

The ConfigGUIModel hides the Reader and Composite Reader roles from the Config Model and exposes the reader devices through a specific interface. This parallels the fact that the GUI will treat these devices specially, and the plugin interface exposed out of this layer can be used in a generic fashion to display the rest of the configurable objects.

6.8.2 PhysicalReader

PhysicalReader wraps a reader-type EdgePlugin.

It implements the getLogicalReaderNames call by going through the configuration for the reader and assuming all parameters of type “logical reader name” represent a logical reader defined by this physical reader. This assumption is true as of RFTA 1.3 for all reader drivers.

6.8.3 CompositeReader

PhysicalReader wraps a composite reader-type EdgePlugin. It understands the configuration specifics of the composite reader and exports a specific interface for that configuration task.

6.9 com.connecterra.admin.gui.configModel

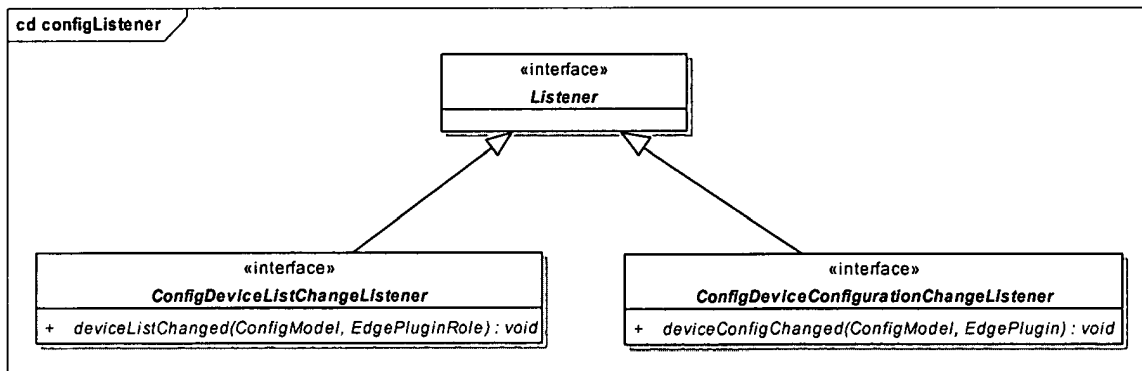
6.9.1 ConfigModel

The ConfigModel is a cache for all the configuration on an Edge Server.

6.9.2 ConfigController

The ConfigController is responsible for keeping the ConfigModel up to date. Currently, a refresh is triggered manually at the GUI level, though this may easily be extended to involve the AlarmClock with a polling interval, as in the ALEController. It all depends on the use case; the configuration is not expected to change often.

6.9.3 ConfigListener



ConfigListener follows the pattern established by ALEListener and TelemetryListener. Additional specific change listeners will be added as they are needed.

7 External Changes

7.1 Specific add/update to Java API

None.

7.2 Specific add/update to XML API

None.

7.3 Specific add/update to Web Services API

None.

7.4 Specific add/update to Configuration

Configuration of the Admin Console's list of edge servers now consists of a list of host/port pairs, not a list of URI bases.

7.5 Specific add/update to Scripts

None.

7.6 Specific add/update to GUI

This design enables usability, performance, and scalability changes to the UI, but does not actually make any of those changes.

7.7 Specific add/update to Command Line Interface

None.

7.8 Specific add/update to Protocol

None.

7.9 Specific add/update to Syntax

None.

7.10 Specific add/update to Database/Persistence

None.

7.11 Specific add/update to Installer

None.

7.12 Specific add/update to Samples

None.

7.13 Specific add/update to jar files and other libraries

It is likely that new jar files will be introduced, but the details are not yet known.

7.14 Security considerations

None.

7.15 Compatibility Considerations

The EdgeModel should manipulate the same configuration file format which is used by the current UALEGUI.

7.16 Performance Considerations

The design centers around providing an interface which appears responsive to the user and scales well to large installations.

7.17 Testing considerations

7.18 Deployment considerations



Unified ALEGUI – Code Design

Version of 2004-04-12 at 14:23

This document is Company Confidential

Table of Contents

Unified ALEGUI – Code Design	1
1 Introduction	1
2 Components	2
2.1 Visual Parts List	2
3 Code Design	3
3.1 TelemetryController	4
3.2 EdgeServerController	5
3.3 DeviceBrowserController	5
4 Synchronous vs. Asynchronous Task Handling	5

Revision history: latest at the top please			
Version	Author	History	Date(yyyy-mm-dd)
001	Larry Kyrala	created	2004-04-06

1 Introduction

This document describes the code design for the Unified ALEGUI (UALEGUI). The code shall also adopt some of the ideas presented in the following Connecterra Internationalization document:
\$/doc/design/Components/RFTagAware-Jun04/Connecterra i18n.doc.

2 Components

2.1 Visual Parts List

This section details the “parts” of the UALEGUI. These correspond to visual parts and the models that will need to be used to represent them. In addition, any relationships to existing data are indicated. This section corresponds to the estimates for work based on the visual design in the Information Architecture.

Visual Part	Swing Components	Models	Related Data
MainFrame	JMenuBar, JMenu, JMenuItem JSplitPane (JG ¹) DeviceBrowserPane		
DeviceBrowserPane	SimpleInternalFrame (JG) JScrollPane JTree DeviceTreeRenderer	EdgeServerNode FolderNode ² MonitorNode ³ ECSpecNode ECSpecSubscriberNode LegacyEdgeServerNode	Telemetry.getDeviceInfo() ALE.listECSpecNames() ALE.listSubscribers()
FolderPane	JScrollPane JPanel JLabel (with Icons)	EdgeServerNode MonitorNode FolderNode	
DevicePane	JTabbedPane DeviceTelemetryPane DeviceAlertsPane DeviceAttributesPane		
ECSpecsMonitorPane	JToolBar JTable ECSpecSubscribersMonitorPane	ECSpecsTableModel	ALE.getECSpecInfo()
ECSpecSubscribersMonitorPane	JToolBar JTable	ECSpecSubscribersTableModel	ALE.getECSpecSubscriptionInfo()
ReadOnlyPane	JPanel ECSpecPane ECSpecSubscriberPane		
ConfigureEdgeServersDialog	JToolBar JTable	ConfigEdgeServerTableModel	ualegui.props
EditEdgeServerDialog		ConfigEdgeServerTableModel	
AutoRefreshDialog			ualegui preferences, default = 2000 ms.
AboutDialog			properties bundle
DeviceTelemetryPane	JScrollPane CounterPanel GaugePanel		deviceID.counters deviceID.gauges
CounterPanel	JChartPanel	CounterDataModel	pushCounterData
GaugePanel	JChartPanel	GaugeDataModel	pushGaugeData
DeviceAlertsPane	JTable JScrollPane(s) JTextField	AlertDataModel ColorString AlertCellRenderer	pushEventLog
DeviceAttributesPane	JTable JScrollPane	AttributeDataModel	pushInfo?
ECSpecFrame	ECSpecPane		
ECSpecSubscriberFrame	ECSpecSubscriberPane		
ECSpecPane	JTree LogicalReaderPanel BoundaryPanel StartPanel StopPanel	FormNode	ECSpec ECReportOutputSpec ECReportSpec

¹ JG = JGoodies style

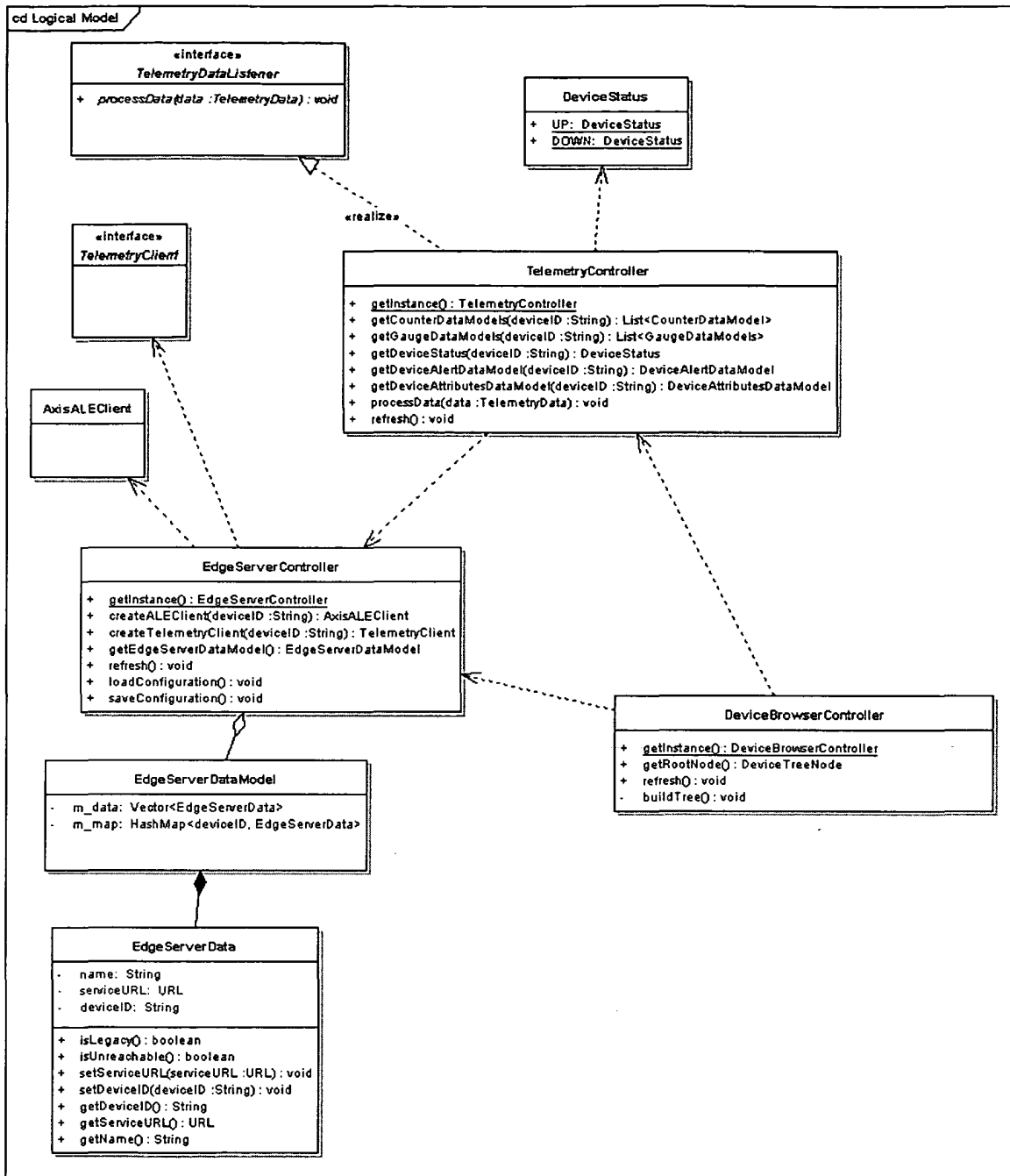
² These are no-op organizational nodes. Some will be generated by device id, others will be generated by predetermined rules (e.g. “EdgeServerNodes have “Readers” and “Triggers” FolderNodes).

³ MonitorNodes are containers that also monitor their contents. (e.g. “ECSpecs” is a MonitorNode).

	ApplicationDataPanel ReportSpecificationContainerPanel ReportSpecificationPanel ReportTypePanel ReportSetPanel BehaviorWhenEmptyPanel FilterContainerPanel FilterPanel		
ECSubscriberPane			ECSubscriptionControls URI
TextFrame	JTextField JScrollPane JMenuBar, etc.		Human-readable or XML serialized ECSpec, ECReport text.
ErrorDialog			resource bundle stack-trace (for problem report)

3 Code Design

This section details other components needed in the design that are not visual, such as controllers.



3.1 TelemetryController

The TelemetryController coordinates all monitoring data (counters, gauges, subscription status, last activated counts, etc.) coming from the edgserver. (Note that some of the data comes from traditional telemetry methods, and other parts of it come from ALE APIs). Whether the actual data is pulled or pushed doesn't matter, it will be stored and retrieved by this controller by mapping telemetry data items to deviceID. Get operations may be implemented differently – pull operations would be executed when gets are called, whereas push operations would retrieve stored data as it comes in from the edgserver. The end result however, is that the rest of the GUI application does not need to distinguish between push/pull modes, which will make it easy to change these modes for various data as desired in the future. Models

must be thread safe since they will be accessed/modified from various threads, including the main dispatch thread.

3.2 EdgeServerController

The EdgeServerController maintains the configuration data for edgeservers, as well as maintaining a mapping of deviceID to various EdgeServer client interfaces. Since the rest of the GUI application speaks of selectable devices in terms of their “deviceID”, they may ask the EdgeServerController for AxisALEClient interfaces or TelemetryClient interfaces by “deviceID” instead of having to ask for the service URL.

AxisALEClient is necessary rather than the more general ALEClient or ALE because access to the underlying factory methods is necessary for much of the construction work required of things like the ECSpec frame, etc.

3.3 DeviceBrowserController

The DeviceBrowserController takes care of building and maintaining the main tree control for the application. It uses the EdgeServerController to iterate all the configured Edgeservers and list devices/specs/subscribers. It also uses the TelemetryController to detect changes in device status.

4 Synchronous vs. Asynchronous Task Handling

Tasks that may take a long time are typically tasks using the EdgeServer in some way. These tasks may be initiated in the following contexts:

- 1) the task needs to execute synchronously – i.e. other user action is not allowed to occur, however wait_cursor is updated during this task over the affected frames. Impl note: use glassPane approach outlined in <http://java.sun.com/docs/books/tutorial/uiswing/components/rootpane.html> and set the cursor on start and end of the task.
- 2) the task may execute asynchronously – i.e. the user can browse other items while this task is completing. Visual feedback that an action is being processed should be given in the form of a progress dialog. Impl note: spawn new worker thread... this thread will either update the TelemetryController models async. or will post a ECSpec/Subscriber edit window invocation command using SwingUtils.invokeLater.

Examples of tasks:

- **populating the device browser tree** – (asynchronous trickle – user should be able to select while other parts of tree are loading rather than have to wait for the entire tree to load at once and on every update.) By and large this is handled in the implementation of TelemetryController, which will spawn a worker thread for interrogating pull data, and have separate listener threads for receiving push data.
- **populating monitor data (ecspec monitor, subscribers monitor, etc)** – again handled by telemetryController.

- **edit, deploy, activate once, test** – any of these operations should open a progress dialog. They all have well known end states: the progress dialog closes and
 - **edit** – the ECSpec or ECSubscriber edit frame opens. (on success only)
 - **commit the ECSubscriber** – ECSubscriber edit frame closes. (on success only)
 - **deploy** – the ECSpec edit frame closes. (on success only)
 - **activate once** – the TextView frame opens with the ECRReport (success only)
 - **test** - the TextView frame opens with the ECRReport (success only)



Unified ALEGUI – Information Architecture

Version of 2004-05-14 at 19:42

This document is Company Confidential

Table of Contents

Unified ALEGUI – Information Architecture	1
1 Introduction	3
2 Goals	3
2.1 Monitoring of real-time (status) and non-real-time (attributes/configuration) EdgeServer information	3
2.2 Authoring of ALE data objects.....	3
2.3 Managing the ALE service	3
2.4 Reporting of results	3
2.5 Cross-platform support (Windows, Linux, Solaris)	3
3 User Experience	4
3.1 Audience.....	4
3.1.1 Software Engineer	4
3.1.2 Developer	4
3.1.3 Support Professional.....	4
3.2 Scenario	4
3.3 Competitive Analysis – (TBD?? N/A?).....	5
4 Content and Functional Requirements	5
4.1 Content	5
4.1.1 ALE data objects	5
4.1.1.1 First-class objects:	5
4.1.1.2 Second-class objects:.....	6
4.1.2 Telemetry API Objects	8
4.1.2.1 First-class objects:	8
4.1.3 Tag Writing Data Objects.....	9
4.1.3.1 First-class objects:	9

4.1.3.2	First-class objects:	9
4.1.3.3	Second-class objects:	9
4.1.4	Other Content	11
4.2	Functional Requirements	11
4.2.1	Ability to monitor real-time (status) and non-real-time (attributes/configuration) EdgeServer information	11
4.2.1.1	Real-time data - (pushed from ES or polled by UALEGUI).	11
4.2.1.2	Non-real-time data – (manually refreshed or selected on demand).	12
4.2.2	Ability to author ALE data objects presented in section 4.1 Content	12
4.2.3	Ability to manage ALE service	13
4.2.4	Ability to manage the ALEGUI	13
4.2.5	Ability to report results of authoring and managing tasks	13
4.2.6	Cross-platform support for Windows, Linux and Solaris.	13
4.2.7	User Help	13
4.2.8	General	13
4.2.9	Performance and Scalability	14
5	UALEGUI Design	15
5.1	Motivating design metaphors	15
5.2	Visual Design	16
5.2.1	Main Frame – SiteID Selection	17
5.2.2	Configure-EdgeServers and Preferences	18
5.2.3	About Dialog	19
5.2.4	Device (EdgeServer) Telemetry Selection	19
5.2.5	Device (EdgeServer) Alerts Selection	20
5.2.6	Device (EdgeServer) Attributes Selection	20
5.2.7	ECSpecs Monitor Selection	21
5.2.8	PCSpecs Folder Selection (not for R2)	22
5.2.9	ECSpec selection (read only view of spec)	23
5.2.10	ECSpec Frame (part 1)	24
5.2.11	ECSpec Frame (part 2)	25
5.2.12	ECSubscriber Frame	36
5.2.13	TextView Frame	41
5.2.14	Error Dialogs	42

Revision history: latest at the top please			
Version	Author	History	Date(yyyy-mm-dd)

005	Larry Kyrala	post review comments added	2004-04-05
004	Larry Kyrala	added Visual Design	2004-03-29
003	Larry Kyrala	edits as result of review	2004-03-22
002	Larry Kyrala	further edits for review	2004-03-17
001	Larry Kyrala	created	2004-03-08

1 Introduction

The purpose of this document is to present the "Information Architecture" of the Unified ALEGUI (UALEGUI) for the R2 timeframe.

2 Goals

2.1 Monitoring of real-time (status) and non-real-time (attributes/configuration) EdgeServer information

Among these forms of information: telemetry, configuration, device and service attributes.

2.2 Authoring of ALE data objects

Creating working ALE data objects can quickly become very complex. Some kind of authoring capability that insulates the user from the raw complexity of composing such objects into valid constructs is desirable if not necessary.

2.3 Managing the ALE service

The user needs to have control over how the ALE service processes data: when and how specs are processed, and how report results are produced. Also controls the storage, retrieval and removal of data objects maintained by the ALE service instance(s).

2.4 Reporting of results

Once data has been processed, results may need to be displayed directly by the Unified ALEGUI. This is useful in developer testing, product demonstration and spot-checking deployed configurations. However, its use for an actual production reporting solution may be limited. – e.g. for enterprise reporting solutions, use of ALE notification subscribers will probably be more useful and flexible.

2.5 Cross-platform support (Windows, Linux, Solaris)

This motivates the adoption of Java/Swing/AWT as a client-side technology for implementation of the Unified ALEGUI.

3 User Experience

3.1 Audience

There are at least three different roles that might use the UALEGUI to manage a given RFTA deployment: software engineers, developers and support professionals. Workers in a given deployment may take on one or more of these roles depending on available resources.

3.1.1 Software Engineer

a software engineer will likely have installed our sample code that allows programming of the ALE API (Java or WSDL). They will require the ability to monitor ALE service processes and reader output (as exposed by ALE) in order to test and debug aspects of their applications using ALE.

3.1.2 Developer

an IT professional who will be responsible for authoring ECSpecs to produce the desired ECRports for their organization/application.

3.1.3 Support Professional

an IT professional who will be responsible for monitoring status of equipment and making sure that all devices are operating within acceptable limits. Must be "aware" of machine/service status so that they can respond quickly to problems in the field.

3.2 Scenario

One possible example of how the different audiences might interact in a RFTagAware deployment might be as follows:

- 1) **software engineers** are tasked to build an enterprise application or integration layer that uses the ALE API. For example, maybe they are working on an inventory management system that coordinates data from the field with back office data in their data center.
 - a. SW engineers build the application
 - b. they test functionality of the application against local instances of the EdgeServer and use the UALEGUI to setup and debug related ECSpec use cases for their application.
 - c. later, when rolling out the solution to a larger-scale prototype, the SWs will use the UALEGUI to troubleshoot/debug interactions with the application that arise in a larger context.
- 2) **developers** meet with engineers to determine use cases and what kinds of ECSpecs they will need to drive the application. Developers must also consider the proposed deployment scenarios and may need to fine-tune ECSpecs for optimal results.
 - a. Developers use UALEGUI to author ECSpecs for their application.
 - b. they can then test these specs in the prototype. They use feedback from ALEGUI to determine how well their authored ECSpecs work in prototyping.
 - c. Later, when the solution is being rolled out, developers may need to adjust some of their models as problems arise or assumptions are tested in a larger context.

- 3) **Support professionals** monitor the status of the deployed readers, edgeservers, (and perhaps their application as well). They may try to fix certain types of configuration problems, or forward more complex problems to developers and/or engineers as necessary.
 - a. **Support** staff monitors deployment with UALEGUI monitoring features
 - b. When problems are spotted they can read log status, check antenna signals, etc.
 - c. They may also provide detailed feedback on real-world deployment performance and problems back to developer and SW engineering teams.

3.3 Competitive Analysis – (TBD?? N/A?)

It's unknown what other monitoring/management applications are available for RFID at this time.

It would be interesting if the UALEGUI could be used with **any** ALE-standard implementation (not just ours) but it's too early to explore this as a requirement.

Tivoli, HP OpenView may be alternate management interfaces, but the EdgeServer might integrate with these tools as well, so they aren't necessarily competitors.

4 Content and Functional Requirements

4.1 Content

4.1.1 ALE data objects

Supporting goal 2.2, all ALE interface defined objects will be part of the content managed by the unified ALEGUI.

These are the data objects defined by the ALE interface (Java and WSDL APIs). They fall into two categories (defined for purposes of interaction design):

- 1) "first-class" ALE data objects, which are directly consumed or returned by the ALE interface methods, and
- 2) "second-class" ALE data objects, which are parts of first-class ALE objects. Although these "second-class" objects are defined by the ALE interface, they do not appear directly in any ALE interface methods. However, they do appear in ALEFactory creation methods (Java API only) or as members of returned first-class objects.

Here are tables of the ALE data objects followed by a relationship diagram.

4.1.1.1 First-class objects:

Input	Output
ECSpec	ECSpec
<i>ECSubscriptionControls (also 2nd class output)</i>	ECReports
String : ECSpec name	ECSpecInfo
URI : Subscription	ECSubscriptionInfo

	List<String> : all ECSpec names on a given server List<URI> : all Subscription URIs for a given ECSpec List<String> : all logical reader names on a given server
--	--

4.1.1.2 Second-class objects:

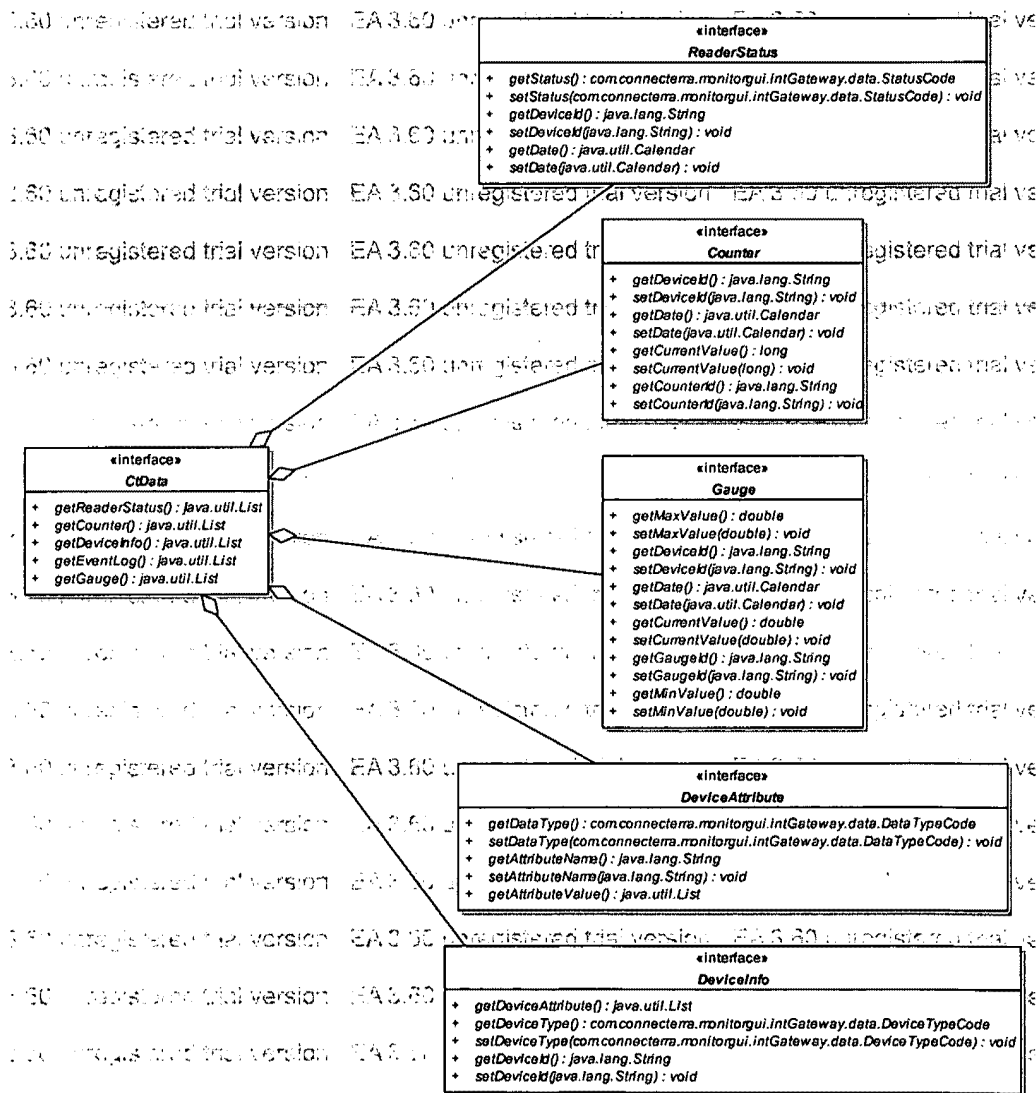
Input	Output
ECCountReportOutputSpec	<i>ECSubscriptionControls (also 1st class input)</i>
ECListReportOutputSpec	ECListReport
ECReportSpec	ECCountReport
ECReportSetSpec	ECTerminationCondition
ECInterval	
ECIntervalUnit	

4.1.2 Telemetry API Objects

The Telemetry API contains five types of data objects. (this is based on existing function, design for TelemetryAPI coming soon – see Sharath [put link to spec here later])

4.1.2.1 First-class objects:

Input	Output
	ReaderStatus
	Counter
	Gauge
	DeviceAttribute
	DeviceInfo



4.1.3 Tag Writing Data Objects

Please see the spec (in particular section 4 Java API) in

`$/doc/design/Components/RFTagAware-Jun04/ALEWritingArch.doc`, which is currently under development. See `/x/sw/vcc/images/1224-smr-01/ALEAPI/src` for latest design. (where this information was last updated from.)

4.1.3.1 First-class objects:

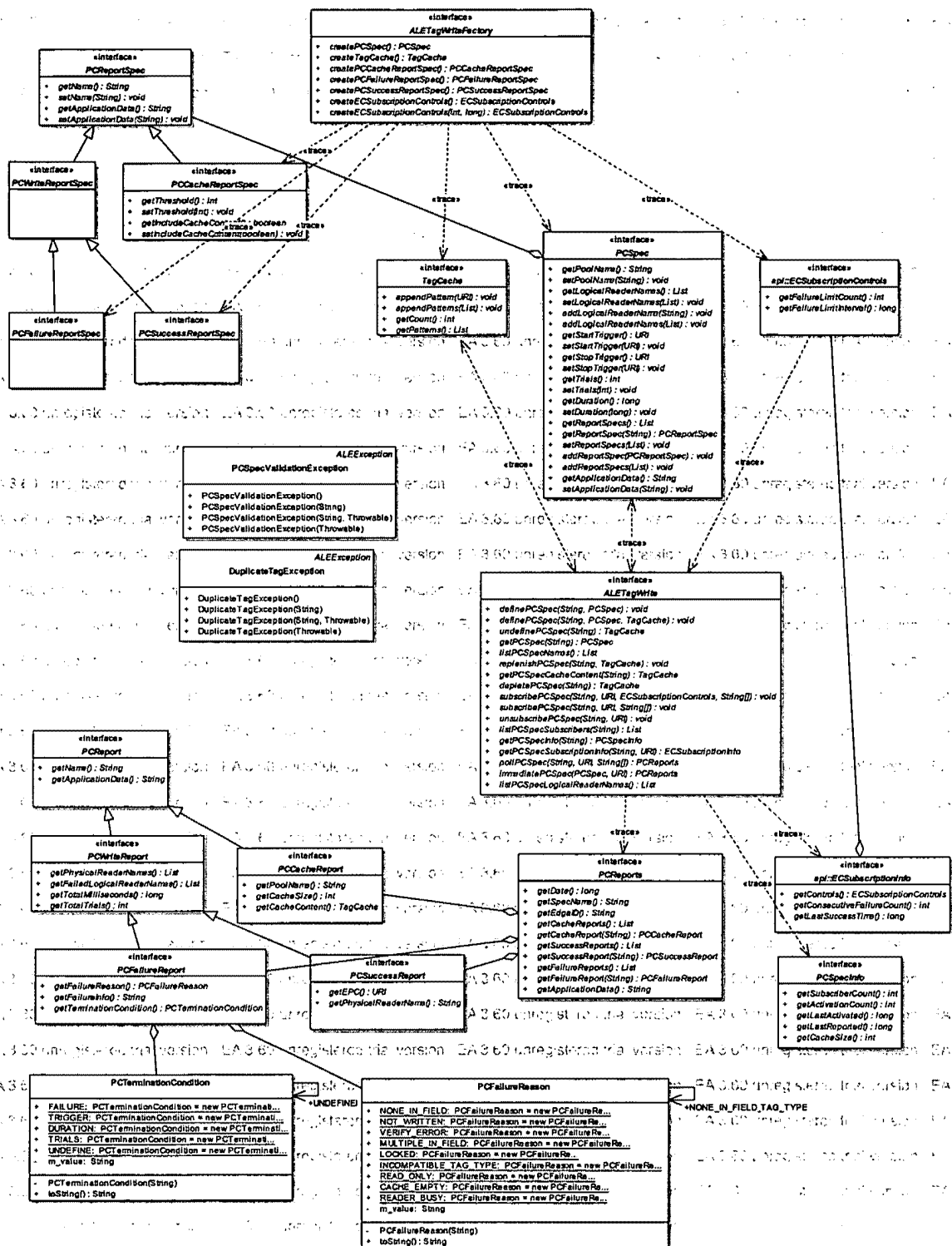
Input	Output
WCSpec	WCSpec
String : WCSpec name	TagCache
TagCache	WCReports
EPC	

4.1.3.2 First-class objects:

Input	Output
PCSpec	PCSpec
<i>ECSubscriptionControls (also 2nd class output)</i>	PCReports
String : PCSpec name	PCSpecInfo
URI : Subscription	ECSubscriptionInfo
TagCache	TagCache
	List<String> : all PCSpec names on a given server
	List<URI> : all Subscription URIs for a given PCSpec
	List<String> : all logical reader names on a given server

4.1.3.3 Second-class objects:

Input	Output
PCCacheReportSpec	<i>ECSubscriptionControls (also 1st class input)</i>
PCFailureReportSpec	PCCacheReport
PCSuccessReportSpec	PCFailureReport
	PCSuccessReport
	PCTerminationCondition
	PCFailureReason



4.1.4 Other Content

EdgeServers – (display name, service URL) used by UALEGUI to contact an EdgeServer instance.
[UALEGUI configuration]

Text representations (views and files) of ECSpec and ECRReport objects.

XML representations (files) of ECSpec and ECRReport objects. (handled by ALE Java API serializers)

4.2 Functional Requirements

4.2.1 Ability to monitor real-time (status) and non-real-time (attributes/configuration) EdgeServer information

4.2.1.1 Real-time data - (pushed from ES or polled by UALEGUI).

- 1) time-series data does not need to be persisted on the client.
- 2) time-series data “history” is limited to some number n samples
 - a. (not in R2) history length determined by time interval
 - b. (not in R2) configurable history length - number of samples or time interval
- 3) an initial “burst-mode” transfer from the EdgeServer may send history for the past n seconds (or other aggregate interval).
- 4) **counters** for a given *device*¹ – monotonically increasing time-based series (of type long) to be visualized as rate of change over time. Current counter value displayed in corner of chart.
 - a. counters shall have “overwrite” semantics. This means that if counters get two values for the same timestamp, the most recent value is the one used. counters may not contain more than one value per timestamp.
- 5) **gauges** – value range (of type double) sampled over the telemetry period (min, max, current (or last)) to be visualized as a “stock market”-type chart. “current” value displayed in corner of chart.
- 6) device status – **bad | good**
 - a. the status shall not be solely communicated via color.
 - b. the meaning of “bad” and “good” depends on the device being monitored. Documentation shall describe what status means for each device type.
- 7) event logs – {**Importance** (java.util.logging.Level), **Date**², **Message** (String)}
 - a. (not in R2) filter (search) and sort columns.
- 8) **display new devices**³ opportunistically when they send device status telemetry.

¹ Examples of devices in R2 are EdgeServers, Physical Readers and Triggers. More specifically, a *device* is any EdgeServer object that implements and uses the Telemetered interface.

² All user-visible date-time stamps (referred to as “Date” in this document) shall conform to the following format (in the context of java.text.SimpleDateFormat) “yyyy-MM-dd HH:mm:ss z”. See <http://lists.connecterra.net/archive/auto-id/0530.html>.

³ devices have a hierarchal relationship: Savants (EdgeServers) contain Readers and Triggers.

- a. for EdgeServers (ES) -- because push and poll models conflict sometimes (bug #857) for R2: UALEGUI maintains a list of ES. For each ES, telemetry API is called to initiate a "legacy-mode" push (similar to V1.0).
- b. fall-back position in case we can't do this – support ES (edge.props) configuration of monitor, in which case UALEGUI would have to add ES opportunistically.
- c. options 8a and 8b are mutually exclusive. One or the other must be implemented before R2 is shipped.

4.2.1.2 Non-real-time data – (manually refreshed or selected on demand).

- 1) device info (List of attribute-value pairs)
- 2) ECSpecInfo – **activation count** (int), **last activated** (Date), **last reported** (Date), **subscriber count** (int)
 - a. (not in R2) instead of manually refreshing, this info would become real-time data (polled) and move to the previous section.
- 3) ECSubscriptionInfo – **consecutive failure count** (int), **last success** (Date).
 - a. (not in R2) instead of manually refreshing, this info would become real-time data (polled) and move to the previous section.
- 4) **display the list of known (configured) EdgeServers** along with their status. A periodic refresh interval (set via "Settings" menu and synced with properties) will refresh the status of the entire list of EdgeServers currently being managed (from startup of the UALEGUI to shutdown).
- 5) **display the ECSpec names** currently on a given EdgeServer
 - a. (not in R2) filter (search) and sort ECSpecs.
- 6) **display the logical readers** currently on a given EdgeServer
- 7) **display the subscribers** currently for a given ECSpec on a given EdgeServer.

4.2.2 Ability to author ALE data objects presented in section 4.1 Content

- 1) create **new** objects: ECSpecs, WCSpecs
- 2) **import/export** objects from XML: ECSpecs, WCSpecs
- 3) **edit** existing objects: ECSpecs, WCSpecs
- 4) **delete** existing objects ECSpecs, WCSpecs
- 5) when editing objects, the ability to **revert** to previous "snapshot" (creation, last imported or last deployed, depending on editing context)
- 6) **deploy** as newly named ECSpecs, WCSpecs and **redploy** existing ECSpecs, WCSpecs that have been edited.
- 7) **close** objects being edited (with warning if changes are undeployed)
- 8) when editing objects, the ability to **test** revisions before deploying them.
 - a. when testing ECSpecs that use ECReportOutputSpecs containing the ADDITIONS or DELETIONS settings, the UALEGUI shall warn the user that such settings cannot be tested. (must deploy and activate once in order to test these specs – doc explanation).
- 9) while editing objects, the GUI should *validate* that the objects don't contain errors.
 - a. fields should be type-checked and range-checked for safety

- b. illegal constructs (combinations of options, etc.) should not be allowed – UI guided constraint of the user (UI makes mistakes difficult or impossible) is preferable to post-entry notification.

4.2.3 Ability to manage ALE service

- 1) **activate** selected ECSpec(s) once.
- 2) *suspend* selected ECSpec(s).
- 3) create **new** Subscribers⁴
- 4) **edit** existing Subscribers (really unsubscribe/subscribe with similar info)
- 5) **delete** existing Subscribers

4.2.4 Ability to manage the ALEGUI

- 1) **configure** the EdgeServers to monitor in the UALEGUI.
- 2) **exit** the application (with warnings about undeployed state)

4.2.5 Ability to report results of authoring and managing tasks

- 1) when the user executes “activate once” or “test”, a view of the returned ECRReport object shall be presented.
- 2) **export** objects to XML: ECSpecs, WCSpecs, ECRreports⁵
- 3) **export** objects to Text files: ECSpecs, WCSpecs, ECRreports
- 4) (nice to have R2) **print** objects (XML or Text).

4.2.6 Cross-platform support for Windows, Linux and Solaris.

4.2.7 User Help

- 1) Help shall be *context sensitive* so that if a given panel or frame is in focus when help is selected (Help menu, app shortcut?) that the relevant page of help will appear.
 - a. (rank work effort for R2) cross-platform pdf launch or HTML (in rich text control)
 - b. (not in R2) JavaHelp

4.2.8 General

- 1) Properties files shall guard against illegal values and warn appropriately about problems.
- 2) User-facing dialogs shall be used in error conditions when the application is unable to correct the problem. In this case, the dialog with the user shall be one of the following:
 - a. instructions or actions that the user may make to remedy the situation.
 - b. user-centered description of what the problem was – they may not be able to fix it, but let them know a general reason.

⁴ creating a new Subscriber is really a composite of an ECSpecName + notification URI + ECSubscriptionControls. Subscribers are categorized under “Management” because once created they immediately start an actual process on the ALE service, whereas creating ECSpecs only loads data on the service.

⁵ returned as a result of “activate once” or “test”.

c. java class names never exposed.

- 3) User-facing error dialogs shall have an alternate mode, such as a “create problem report” button that would allow a user to save a text file with detailed stack trace information to send via email to our support center. Clicking this button would close the error dialog and then notify the user where the report was written and its filename with another message dialog before returning to the application.
- 4) All actions that are longer than 2 seconds shall open a task-progress dialog allowing the user to cancel the task if desired. The application shall handle the cancellation without generating errors. (Caution: implementation considerations for interrupting ALEClient calls has not been tested/considered)
- 5) Application shall not be MDI (Multi Document Interface) – instead a navigation area shall support a selection (single-click) → browse (in a single info pane), and activation (double-click) → edit (open writable version of content in a new window) metaphor.
- 6) Application frames shall be clearly labeled (titlebar text and/or status bars) so that users can easily identify the state, origin, and relationship of such frames. e.g. an ECSpec edit frame might have the following status: “*ECSpec ‘foo’ on EdgeServer ‘es1’ (undeployed)*” This identifies what is being edited, where it comes from (belongs to), and whether or not the state has changed since it was last deployed. Perhaps titlebar contains “‘foo’ (undeployed)” while the status bar contains the full description.
- 7) User Actions that may cause potential disruptions to processes on a EdgeServer should display a warning confirmation before execution. If the user cancels, the action should not occur. e.g. user is about to delete a subscriber – this may affect other applications listening for the subscription data – the user should be warned.
- 8) Application shall show a Help:About dialog indicating the name of the app, version, etc.
- 9) Objects shall be edited in a single scrollable frame – subforms (e.g. ECReportOutputSpec) shall have two modes: summary and edit. summary is a collapsed version of the form summarizing the selections in a compact space suitable for quick browsing. edit is an expanded version of the form allowing the user to select various controls, buttons and fields in order to edit the contents of the form. “browsing” an object always displays a readonly pane that shows only summary modes. Editing an object opens it in a new window – the user may collapse or expand subforms as desired as they edit the object.
- 10) File dialogs should be sticky – remember where the user last saved/opened.
- 11) File dialogs should default to the user home when first used.

4.2.9 Performance and Scalability

- 1) the GUI shall remain responsive at all times. any tasks that result in prolonged work should be carried out in a background thread. Either a cancelable progress dialog (see 4.2.8 #4) or a status-bar “background” process dialog should be employed. (Caution: implementation considerations for interrupting ALEClient calls has not been tested/considered)
 - a. e.g. populating / refreshing ECSpecInfo for “thousands” of ECSpecs on an EdgeServer, might display a statusbar progress bar indicator. While work continues in background, display is updated opportunistically, and full user control on partial display is still possible. (for example: refreshing an email list doesn’t stop the user from opening an item even before the refresh is complete.)
- 2) UALEGUI should perform reasonably with the following monitoring environment as defined for R2:
 - a. up to 6 EdgeServers

- b. For each EdgeServer, up to 6 physical readers (3x6 = 18 graphs, probably 6 visible at one time).
 - c. For each physical reader up to 4 logical readers (1 per antenna) plus up to 6 composite readers for a total of up to 10 logical readers per physical reader.
 - d. For each EdgeServer up to 100 ECSpecs
 - e. For each ECSpec, up to 10 subscribers
- 3) Screen resolution: support a minimum of 800x600 (i.e. common laptop displays)

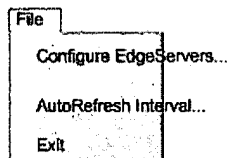
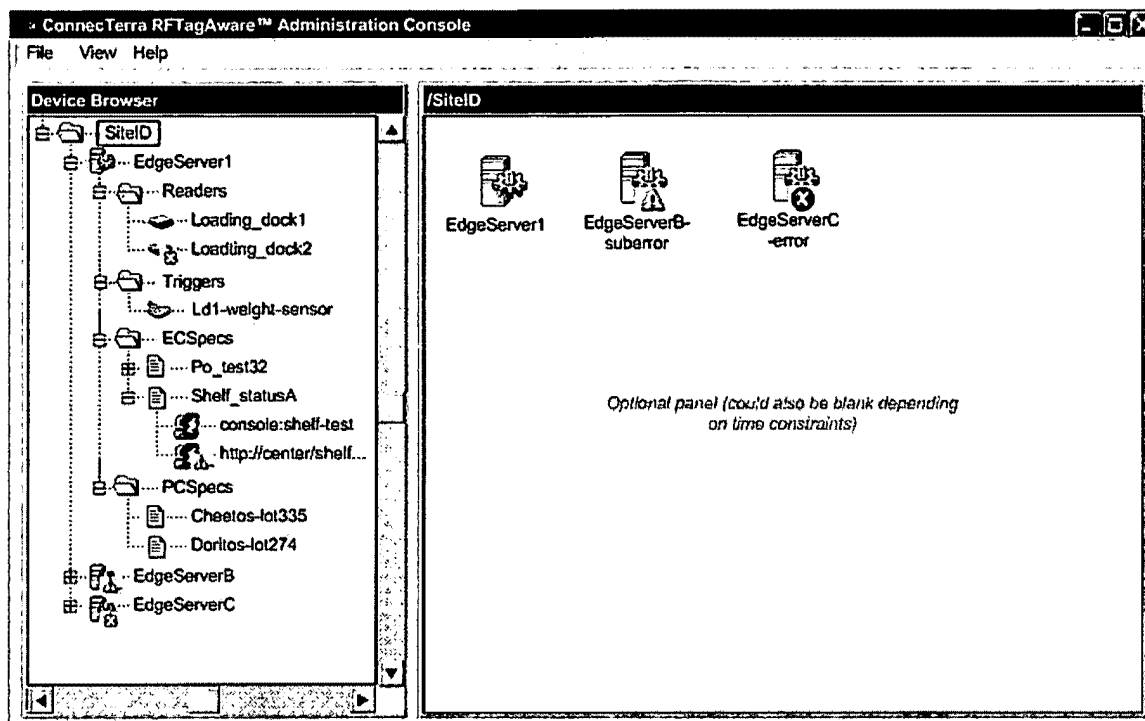
5 UALEGUI Design

5.1 Motivating design metaphors

In dealing with the content described in section 4.1 and the overall workflow of the application, we can draw from several existing visual design metaphors from leading applications. Here are some applications that may provide examples of design metaphors we can use:

- 1) Windows MMC Management Console
- 2) Explorer-type applications
- 3) IBM's Eclipse "perspectives" (authoring, monitoring)
- 4) Altova's XMLSpy (<http://www.altova.com/>)
- 5) Microsoft Visual Studio .NET for web "solutions" authoring and deployment management.

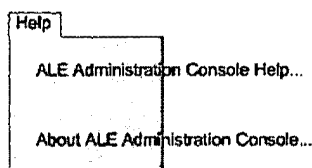
5.2.1 Main Frame – SiteID Selection



Controls how often to automatically refresh the display.

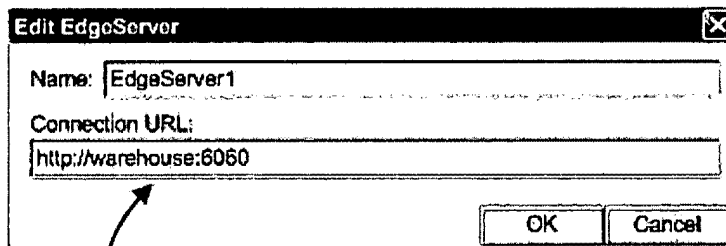
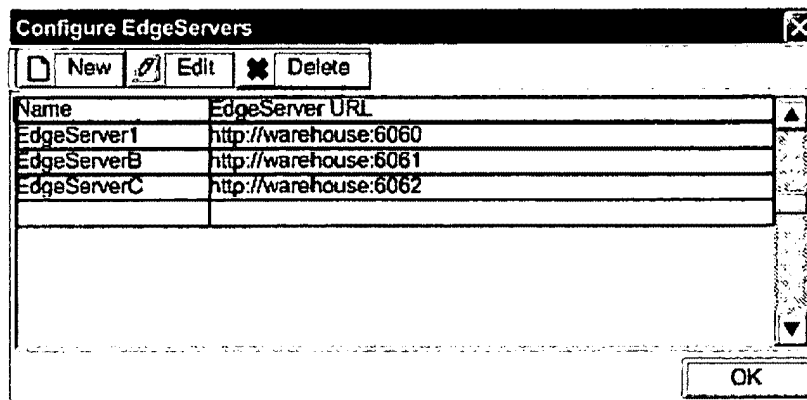


Manually refreshes the currently visible information from the EdgeServer(s).

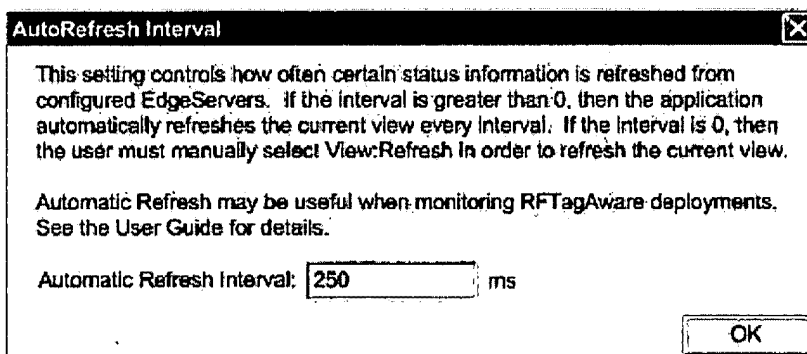


Launches a PDF document describing the current frame.

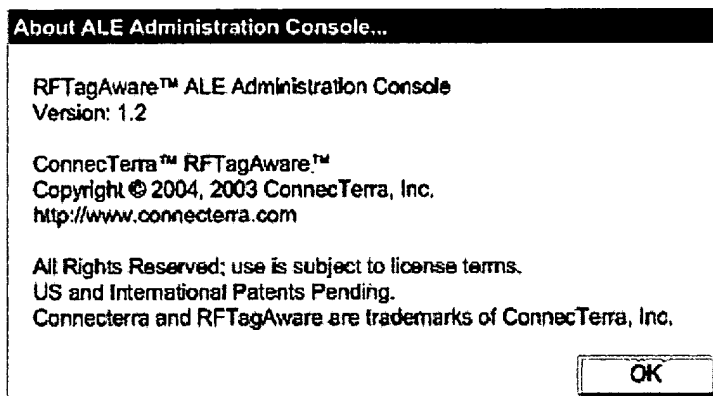
5.2.2 Configure-EdgeServers and Preferences



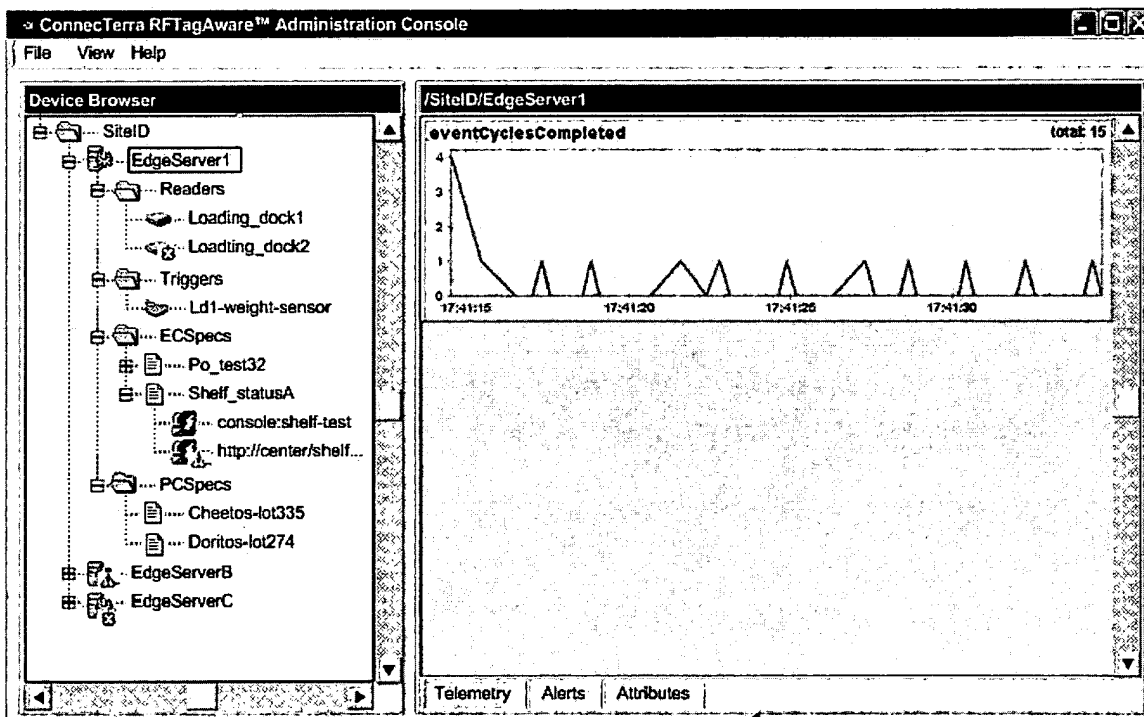
Internally, this may be appended with 1 or more service specifiers, such as "/axis/service/ALEService", or "/axis/service/TelemetryService"



5.2.3 About Dialog



5.2.4 Device (EdgeServer) Telemetry Selection



Nav tabs may be either bottom or top of this display.
We should decide which looks better when we can see
the implementation.

5.2.5 Device (EdgeServer) Alerts Selection

The screenshot shows the 'ConnectTerra RFTagAware™ Administration Console' with the 'Alerts' tab selected. The 'Device Browser' on the left shows a tree structure with 'SiteID' expanded, containing 'EdgeServer1', 'Readers', 'Triggers', 'ECSpecs', and 'PCSspecs'. The main pane displays a table of alerts for '/SiteID/EdgeServer1'.

Importance	Date	Message
SEVERE	2004-03-23 17:53:02 EST	[fake]reader offline
WARNING	2004-03-23 17:53:02 EST	[fake]reader connection interrupted this is a really long ...
INFO	2004-03-23 17:53:02 EST	[fake]reader scan in progress
FINE	2004-03-23 17:53:02 EST	[fake]reader cycle #242 complete
FINER	2004-03-23 17:53:02 EST	[fake]reader 249 subsystem call returned -1
FINEST	2004-03-23 17:53:02 EST	[fake]reader mov #3A9,\$3C

Below the table, a 'WARNING: 2004-03-23 17:53:02 EST' message is displayed in a preview pane:

[fake]reader connection interrupted this is a really long status message with lots of details. Rather than show this extended text in a popup field (slightly ugly way of doing it), we're showing it in a preview pane now. The vertical separator above is draggable.

At the bottom, there are tabs for 'Telemetry', 'Alerts', and 'Attributes'.

If no alerts are selected, the preview pane disappears. The alerts pane (most likely empty) fills the vertical space in this case.

5.2.6 Device (EdgeServer) Attributes Selection

The screenshot shows the 'ConnectTerra RFTagAware™ Administration Console' with the 'Attributes' tab selected. The 'Device Browser' on the left is the same as in the previous screenshot. The main pane displays a table of attributes for '/SiteID/EdgeServer1'.

Attribute Name	Value
Device ID	/SiteID/EdgeServer1
Device Type	Edge
manufacturer	ConnectTerra
version	0.1

At the bottom, there are tabs for 'Telemetry', 'Alerts', and 'Attributes'.

5.2.7 ECSpecs Monitor Selection

The screenshot shows the ConnectTerra RFTagAware™ Administration Console. On the left is the **Device Browser** showing a tree structure: SiteID > EdgeServer1 > Readers > Loading_dock1 > Loading_dock2 > Triggers > Ld1-weight-sensor > ECSpecs. The ECSpecs node is selected. The main panel displays the **/SiteID/EdgeServer1 - ECSpecs** view with a toolbar (New, Edit, Delete, Import, Activate Once) and a table of ECSpecs.

Name	Activation Count	Last Activation	Last Reported	Subscriber Count
Po_test32	2	2004-03-16 15:36:09 EST	2004-03-16 15:38:09 EST	0
Shelf_statusA	445	2004-03-16 15:46:09 EST	2004-03-16 15:46:09 EST	2

Below the table is a context menu with options: New ECSpec, Edit, Delete, Import ECSpec..., and Activate Once. Below this is the **ECSpec "Shelf_statusA" - Subscribers** view with a toolbar (New, Edit, Delete) and a table of subscribers.

URI	Last Succeeded	Consecutive Failures
console:shelf-test	2004-03-16 15:46:09 EST	0
http://center/shelf-check-app	2004-03-16 15:46:09 EST	2

Below the table is a context menu with options: New Subscriber, Edit, and Delete.

This panel disappears if no ECSpec is selected (or exists) above. In this case the (most likely blank) ECSpec panel extends to fill the entire space.

These context menus are nice to have, but not must-have. They duplicate the functionality of the toolbars. They grey out appropriately.

The ECSpecs icon should look more like a "monitor" of some kind – denoting that this node is more than just a container, it allows monitoring of the contained ECSpecs and subscriptions within.

5.2.8 PCSpecs Folder Selection (not for R2)

Connecterra RFTagAware™ Administration Console

File View Help

Device Browser

- SiteID
 - EdgeServer1
 - Readers
 - Loading_dock1
 - Loading_dock2
 - Triggers
 - Ld1-weight-sensor
 - ECSpecs
 - Po_test32
 - Shelf_statusA
 - console:shelf-test
 - http://center/shelf...
 - PCSpecs
 - Cheetos-lot335
 - Doritos-lot274
 - EdgeServerB
 - EdgeServerC

/SiteID/EdgeServer1 - PCSpecs

New Edit Delete Import Activate Once

Name	Activation Count	Last Activation	Last Reported	Subscriber	Cache Size
Cheetos-lot335	2	2004-03-16 15:36:09 EST	2004-03-16 15:36:09 EST	0	300
Doritos-lot274	445	2004-03-16 15:46:09 EST	2004-03-16 15:46:09 EST	0	15

PCSpec "Doritos-lot274" - Subscribers

New Edit Delete

URI	Last Succeeded	Consecutive Fails
console:shelf-test	2004-03-16 15:46:09 EST	0
http://center/shelf-check-app	2004-03-16 15:46:09 EST	2

New ECTSpec

Edit

Delete

Import ECTSpec..

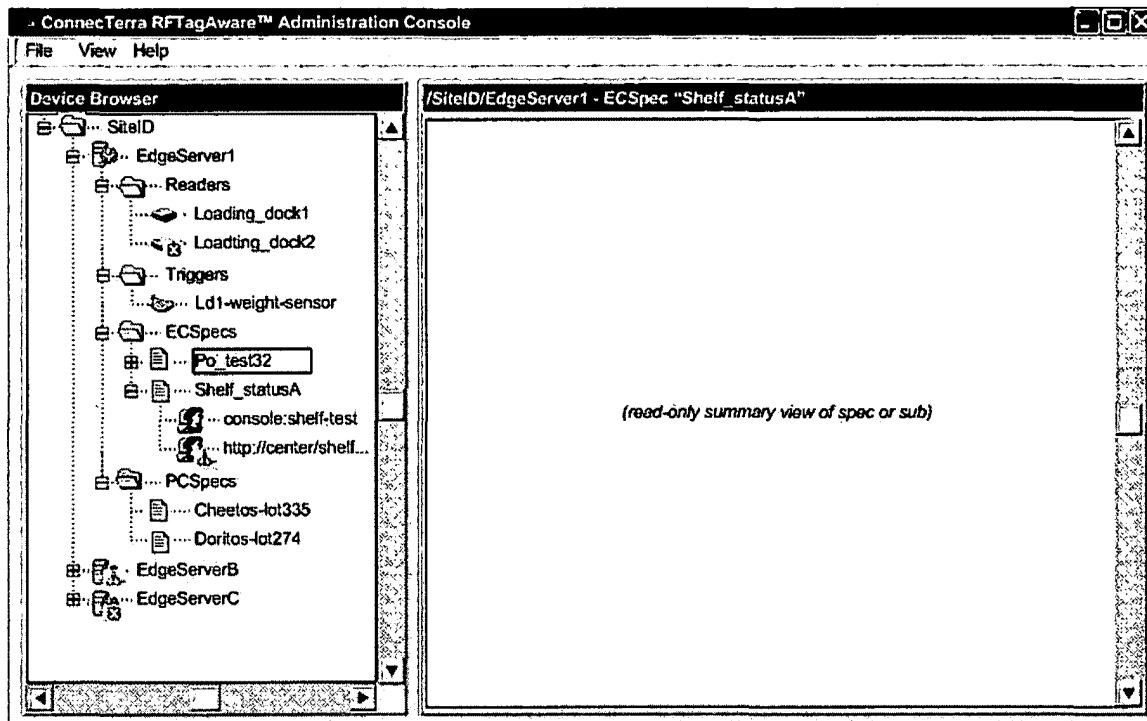
Activate Once

New Subscriber

Edit

Delete

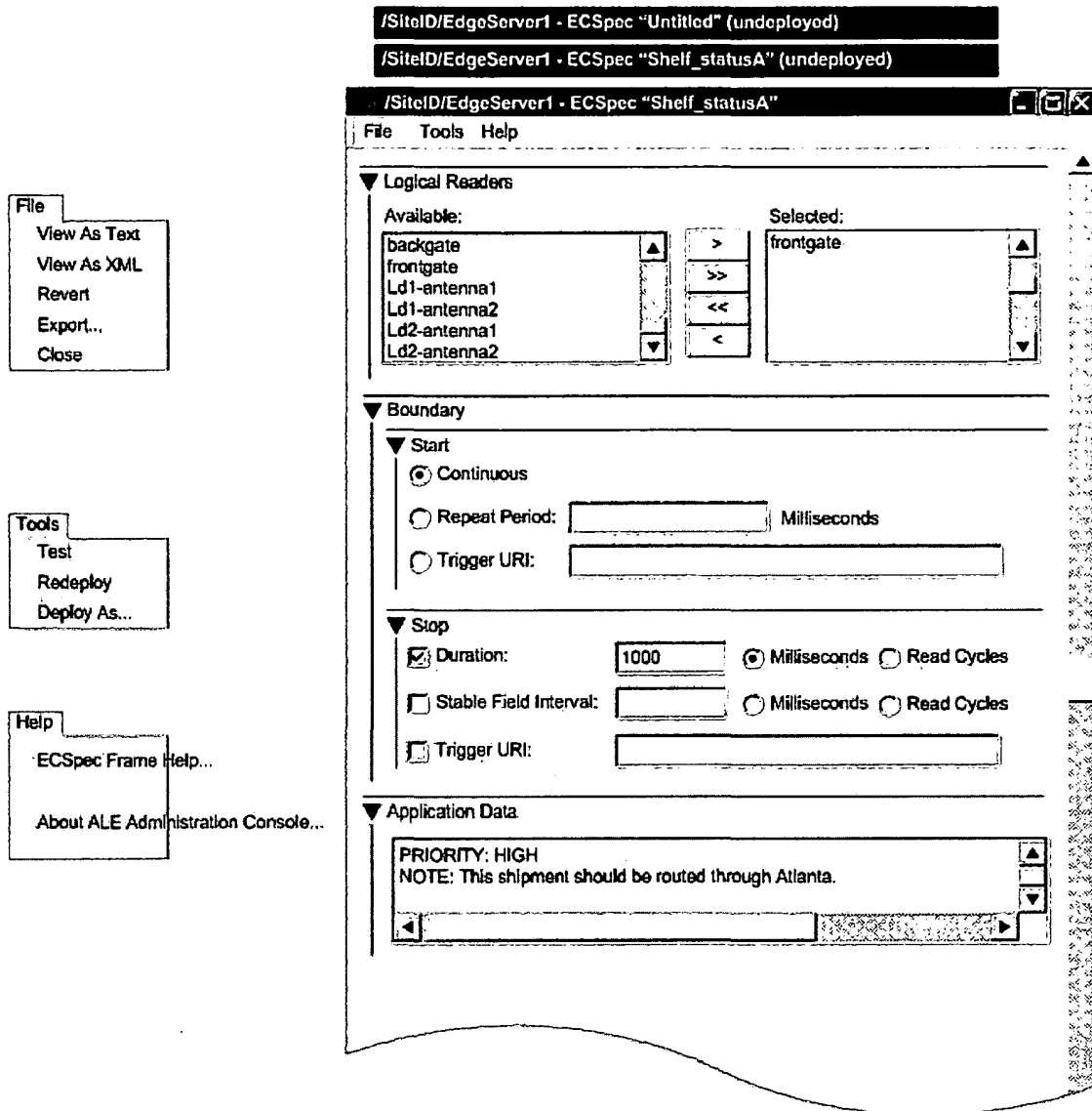
5.2.9 ECSpec selection (read only view of spec)



The read-only view of a ecspec shall display the collapsed view for every leaf node, and shall expand every non-leaf node. collapsed leaf nodes shall display the full content of information summarized. no nodes can be collapsed or expanded in read-only mode – they are a fixed presentation. Any editable fields are locked during read-only mode.

A fall-back alternative to this kind of display is to use a text-view representation of the ECSpec and just make this a text control. (this may be easier).

5.2.10 ECSpec Frame (part 1)



The central idea to the components in this section are expandable and collapsible views. Each leaf node shall have a collapsible view, all non-leaf nodes shall be collapsible in edit view, but not in read-only view as described in the preceding section.

Selecting "Deploy" or "Redeploy" closes the ECSpec being edited after successfully deploying.

5.2.11 ECSpec Frame (part 2)

Report Specifications (1) New Report

Report: Doritos_infield Delete Report

Report Type

☒ Count ☐ List

Report Set

☒ Current ☐ Additions ☐ Deletions

Behavior when Empty

☐ Include in ECRReport
☒ Omit from ECRReport
☐ Suppress ECRReport

Application Data

PRIORITY: HIGH
 NOTE: This shipment should be routed through Atlanta.

Include Filters (1) New Filter

Filter: SGTIN-64, *.*.* Delete Filter

Exclude Filters (0) New Filter

Filter Delete Filter

Tag Format: SGTIN-64

Value

☒ Any
☐ Specific Value:
☐ Range: through

Company Prefix

☒ Any
☐ Specific Value:

Item Reference with Indicator

☒ Any
☐ Specific Value:
☐ Range: through

Serial Number

☒ Any
☐ Specific Value:
☐ Range: through

Callouts:

- Collapsed version of Filter entry. (see exclude filters below for expanded version)
- Other is also an option... presents URI input to user.
- Depending on the format selected, 3 to 4 of these subsections may be visible. See EPC global spec for details.

For validation of the Filter tag format, see the EPC Tag Data Specification 1.1, which gives detailed rules for the production and parsing of such tags.

Repeat Period:

- Integer ≥ 0

Trigger URI:

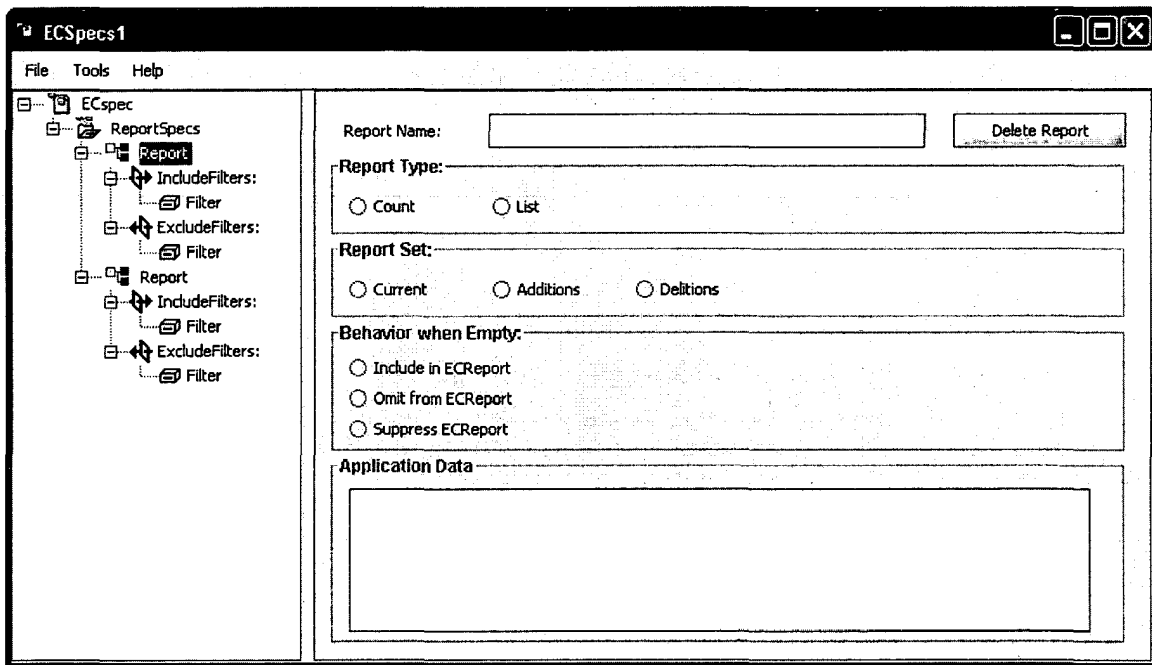
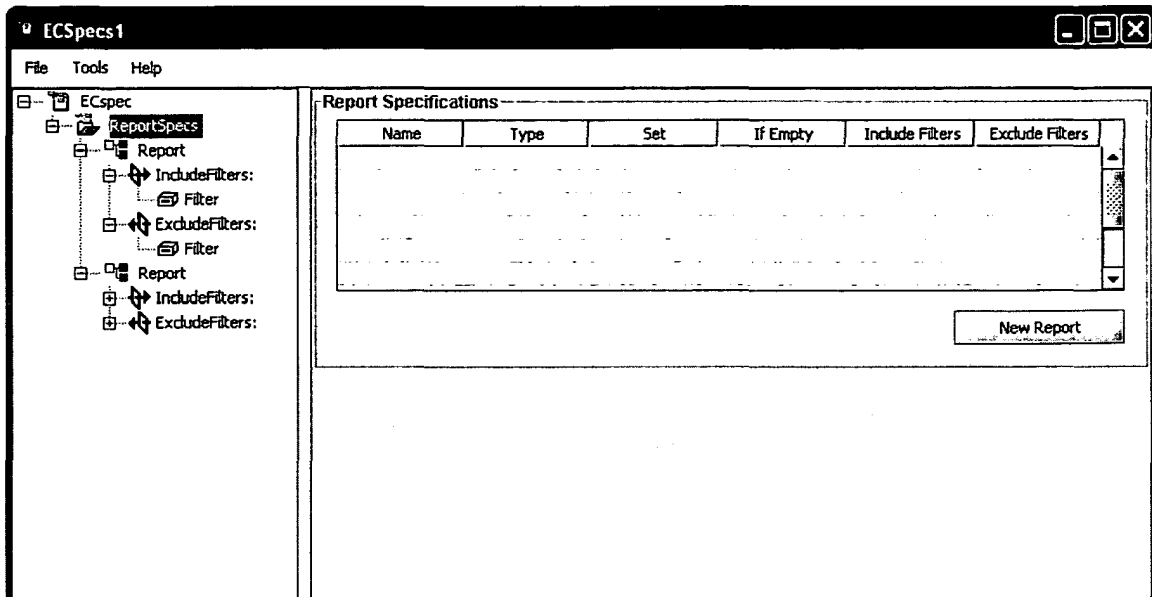
- Valid URI (long string)

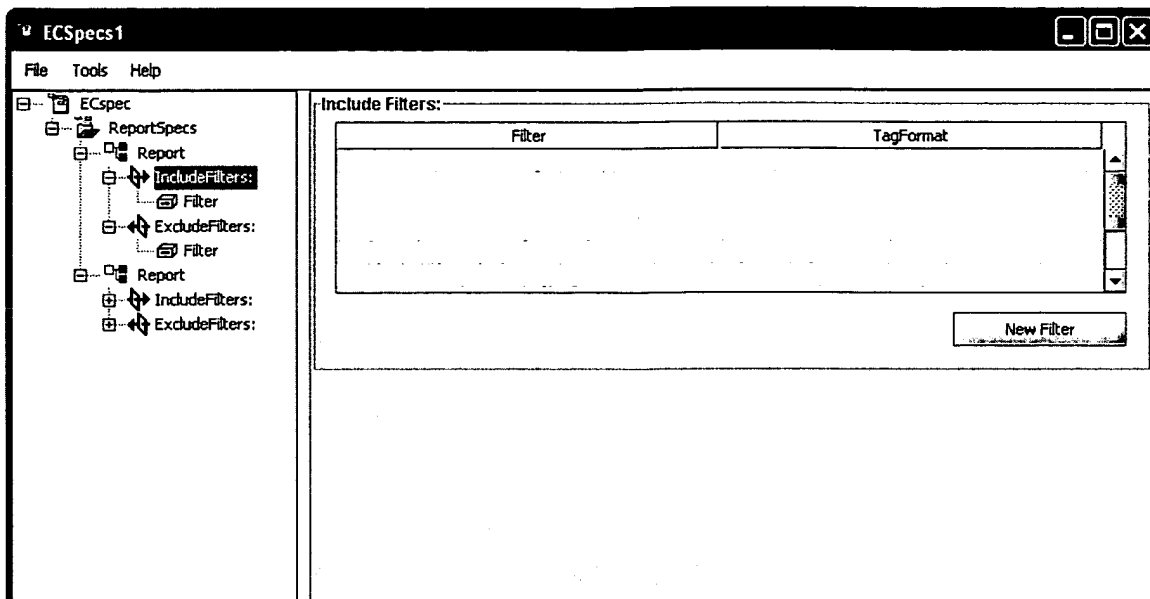
Duration:

- Integer ≥ 0

Application Data:

- text





Filter Tag Format: SGTIN-64, SGTIN-96

The screenshot shows the ECspecs1 application window. On the left is a tree view with the following structure: ECspec > ReportSpecs > Report > IncludeFilters: > Filter (highlighted) > ExcludeFilters: > Report. The main panel on the right contains the following sections:

- Tag Format:** A dropdown menu set to "SGTIN-64" and a "Delete Filter" button.
- Filter Value:** Radio buttons for "Any" (selected), "Specific" (with an input field), and "Range" (with two input fields separated by "through").
- Company Prefix:** Radio buttons for "Any" (selected) and "Specific" (with an input field).
- Item Reference with Indicator:** Radio buttons for "Any" (selected), "Specific" (with an input field), and "Range" (with two input fields separated by "through").
- Serial Number:** This section is highlighted with a red border. It contains radio buttons for "Any" (selected), "Specific" (with an input field), and "Range" (with two input fields separated by "through").

Filter Value:

- all values are numeric from 0 to 7

Company Prefix:

- numeric value, leading 0 s matter

Item Reference with Indicator:

- numeric

Serial Number:

- SGTIN-64 : numeric values less than 2^{25} (33554432)
- SGTIN-96: numeric values less than 2^{38}

If Company Prefix is "Any", Item Reference with Indicator should be "Any".

Filter Tag Format: CGLN- 64, CGLN-96

The screenshot shows the ECSpecs1 application window. On the left is a tree view with the following structure: ECSpec (expanded) -> ReportSpecs (expanded) -> Report (expanded) -> IncludeFilters: (expanded) -> Filter (selected). Below this is ExcludeFilters: (expanded) -> Report. The main panel on the right contains the following controls:

- Tag Format:** A dropdown menu set to "SGLN-64" and a "Delete Filter" button.
- Filter Value:** A section with three radio buttons: "Any" (selected), "Specific" (with an empty text box), and "Range" (with two empty text boxes separated by the word "through").
- Company Prefix:** A section with two radio buttons: "Any" (selected) and "Specific" (with an empty text box).
- Location Reference:** A section with three radio buttons: "Any" (selected), "Specific" (with an empty text box), and "Range" (with two empty text boxes separated by the word "through").
- Serial Number:** A section with three radio buttons: "Any" (selected), "Specific" (with an empty text box), and "Range" (with two empty text boxes separated by the word "through").

Filter Value:

- all values are numeric from 0 to 7

Company Prefix:

- numeric value, leading 0 s matter

Location Reference:

- numeric

Serial Number:

- SGLN-64: numeric values less than 2 power 19 (524288)
- SGLN-96: numeric values less than 2 power 41

If **Company Prefix** is "Any", **Location Reference** should be "Any".

Filter Tag Format: SSCC-64, SSCC-96

ECSpecs1

File Tools Help

ECspec

- ReportSpecs
 - Report
 - IncludeFilters:
 - Filter**
 - ExcludeFilters:
 - Report

Tag Format: SSCC-64 Delete Filter

Filter Value

- ☒ Any
- ☐ Specific
- ☐ Range

Company Prefix

- ☒ Any
- ☐ Specific

Serial Reference

- ☒ Any
- ☐ Specific
- ☐ Range

Filter Value:

- all values are numeric from 0 to 7

Company Prefix:

- numeric value, leading 0 s matter

Serial Reference:

- numeric

If Company Prefix is "Any", Serial Reference should be "Any".

Filter Tag Format: GRAI-64, GRAI-96

ECspecs1

File Tools Help

ECspec

- ReportSpecs
 - Report
 - IncludeFilters:
 - Filter
 - ExcludeFilters:
 - Report

Tag Format: GRAI-64 Delete Filter

Filter Value

- ☒ Any
- ☐ Specific
- ☐ Range

Company Prefix

- ☒ Any
- ☐ Specific

Asset Type

- ☒ Any
- ☐ Specific
- ☐ Range

Serial Number

- ☒ Any
- ☐ Specific
- ☐ Range

Filter Value:

- all values are numeric from 0 to 7

Company Prefix:

- numeric value, leading 0 s matter

Asset Type:

- numeric

Serial Number:

- GRAI-64: numeric values less than 2 power 19 (524288)
- GRAI-96: numeric values less than 2 power 31

If Company Prefix is "Any", Asset Type should be "Any".

Filter Tag Format: GIAI-64, GIAI-96

The screenshot shows the ECSpecs1 application window. On the left is a tree view with the following structure: ECspec (expanded) -> ReportSpecs (expanded) -> Report (expanded) -> IncludeFilters: (expanded) -> Filter (selected). Below this is ExcludeFilters: (expanded) -> Report. The main area on the right contains the following controls:

- Tag Format:** A dropdown menu currently showing "GIAI-64" and a "Delete Filter" button.
- Filter Value:** A section with three radio buttons: "Any" (selected), "Specific" (with an adjacent text input field), and "Range" (with two adjacent text input fields separated by the word "through").
- Company Prefix:** A section with two radio buttons: "Any" (selected) and "Specific" (with an adjacent text input field).
- Individual Asset Reference:** A section with three radio buttons: "Any" (selected), "Specific" (with an adjacent text input field), and "Range" (with two adjacent text input fields separated by the word "through").

Filter Value:

- all values are numeric from 0 to 7

Company Prefix:

- numeric value, leading 0 s matter

Individual Asset Reference:

- GIAI-64: numeric values less than 2 power 39
- GIAI-96: numeric

If Company Prefix is "Any", Individual Asset Reference should be "Any".

Filter Tag Format: GIO-64-I, GIO-64-III, GIO-96

The screenshot shows the ECSpecs1 application window. On the left is a tree view with 'ECSpec' expanded, showing 'ReportSpecs', 'Report', 'IncludeFilters:', 'Filter', 'ExcludeFilters:', and 'Report'. The main area has a 'Tag Format:' dropdown set to 'GIO-64-I' and a 'Delete Filter' button. Below are three sections: 'General Manager Number', 'Object Class', and 'Serial Number'. Each section has radio buttons for 'Any', 'Specific', and 'Range', followed by input fields. The 'Serial Number' section is highlighted with a red box.

Tag Format:

General Manager Number

☒ Any

☐ Specific

☐ Range through

Object Class

☒ Any

☐ Specific

☐ Range through

Serial Number

☒ Any

☐ Specific

☐ Range through

General Manager Number:

- GIO-96: numeric values less than 2 power 28
- GIO-64-I: numeric values less than 2 power 21
- GIO-64-III: numeric values less than 2 power 26

Object Class:

- GIO-96: numeric values less than 2 power 24
- GIO-64-I: numeric values less than 2 power 17
- GIO-64-III: numeric values less than 2 power 13

Serial Number:

- GIO-96: numeric values less than 2 power 36
- GIO-64-I: numeric values less than 2 power 24
- GIO-64-III: numeric values less than 2 power 23

5.2.12 ECSubscriber Frame

This section is at the bottom of each variant below.

Action
 Subscribe
 Cancel

Help
 Subscriber Frame Help...
 About ALE Administration Console...

./Site/D/EdgeServer1/ECSPecs/Shelf_statusA - Subscriber
[-] [X]

Action
Help

Subscriber Type: XML on EdgeServer Console

Heading:

Failure Action

☒ Use EdgeServer's defaults.
☐ Override EdgeServer's defaults:

☐ Unsubscribe after
to deliver reports

consecutive failures

☐ Unsubscribe after
failures to deliver reports

seconds of consecutive

Subscriber Type: Other

URI:

Subscriber Type: XML via HTTP POST

http://

Subscriber Type: XML via JMS Message

Host:

Port:

☒ Topic:

☐ Queue:

Connection Factory:

☐ User

Password:

☐ Naming Context Factory:

☐ Security Principal:

Security Credential:

Message Properties

Name	Value

Subscriber Type: XML File on EdgeServer

Directory or File:

ECSubscriber1

Action Help

Subscriber

Type: XML on EdgeServer Console

Heading:

Failure Action

☐ Use EdgeServer's defaults

☐ Override EdgeServer's defaults:

☐ Unsubscribe after consecutive failures to deliver report

☐ Unsubscribe after seconds of consecutive failures to deliver report

Heading:

- Any character string

ECSubscriber1

Action Help

Subscriber

Type: XML via HTTP POST

URI:

Failure Action

☐ Use EdgeServer's defaults

☐ Override EdgeServer's defaults:

☐ Unsubscribe after consecutive failures to deliver report

☐ Unsubscribe after seconds of consecutive failures to deliver report

URI:

- Any character, valid URI

ECSubscriber1

Action Help

Subscriber

Type: XML via JMS Message

☐ Topic:

☐ Queue:

Connection Factory:

☐ Provider URL:

☐ User: Password:

☐ Naming Context Factory:

☐ Security Principal:

Security Credential:

Naming Service Message Properties:

New Delete

Name	Value

JMS Message Properties:

New Delete

Name	Value

Either "Topic" or "Queue", and "Connection Factory" are required fields. All other fields are optional (hence the checkboxes).

All other fields are strings containing any character. (this is not strictly true, but the validation rules if any need to be specified by Sharath in the Deployment Guide – we will need this for online help as well – the URI wrap/unwrap implementation is being done by Manju) It would be very helpful to have examples of possible entry data.

The screenshot shows the 'ECSubscriber1' application window. It has a menu bar with 'Action' and 'Help'. The 'Subscriber' section contains a 'Type' dropdown menu set to 'XML File on Edge Sever' and an empty 'Directory or File' text box. The 'Failure Action' section has two radio buttons: 'Use EdgeServer's defaults' (selected) and 'Override EdgeServer's defaults:'. Below the radio buttons are two checkboxes, both labeled 'Unsubscribe after', each followed by an empty text box and a label: 'consecutive failures to deliver report' and 'seconds of consecutive failures to deliver report'.

Directory or File:

- Any character

The screenshot shows the 'ECSubscriber1' application window. It has a menu bar with 'Action' and 'Help'. The 'Subscriber' section contains a 'Type' dropdown menu set to 'Other' and an empty 'http://' text box. The 'Failure Action' section has two radio buttons: 'Use EdgeServer's defaults' (selected) and 'Override EdgeServer's defaults:'. Below the radio buttons are two checkboxes, both labeled 'Unsubscribe after', each followed by an empty text box and a label: 'consecutive failures to deliver report' and 'seconds of consecutive failures to deliver report'.

http:// :

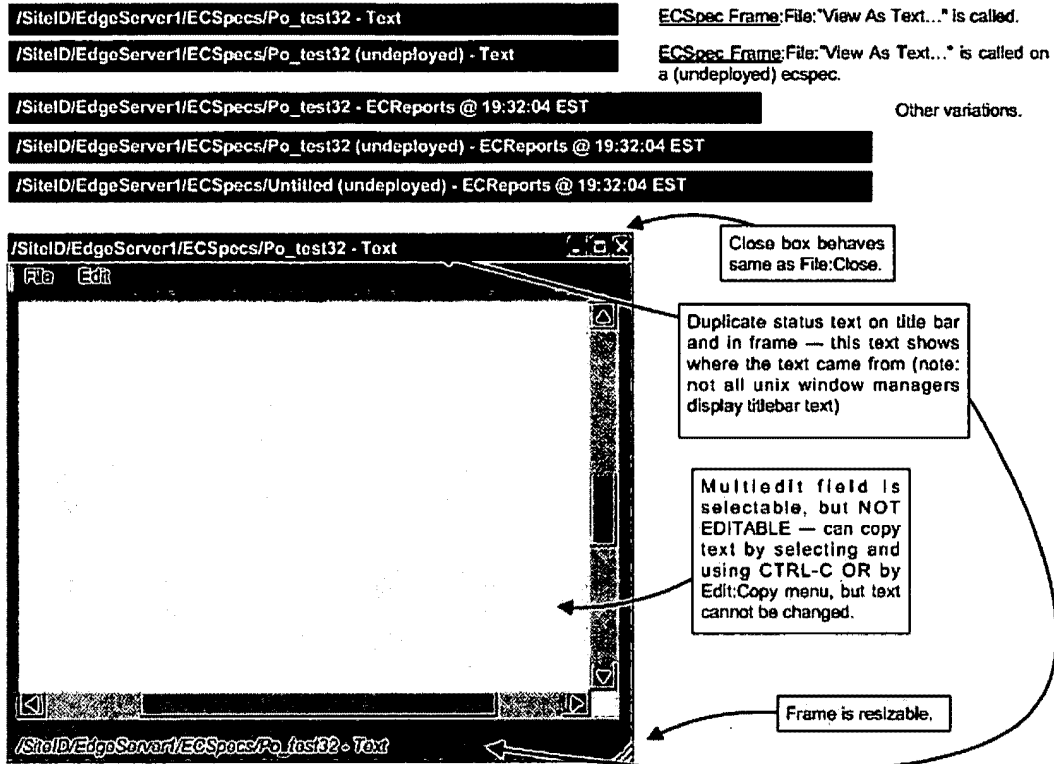
- Any character, valid URL

For "XML via HTTP POST" the URL should probably not hard code http:// because some users may want to use an https:// URL instead.

5.2.13 TextView Frame

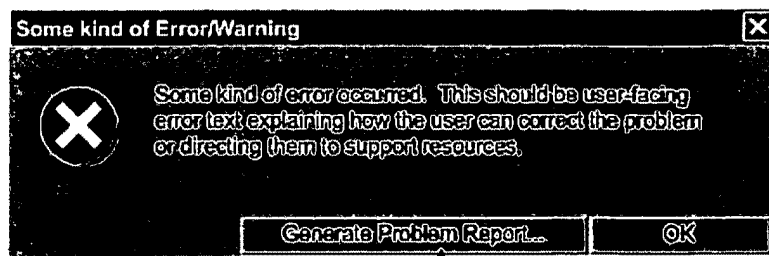
TextView Frame

(titlebar naming examples)



A nice to have feature would be displaying HTML-formatted text (to clarify presentation), but when saved as text, have it write out the plain text file. (requires two serializations).

5.2.14 Error Dialogs



Selecting this button opens a file dialog that allows the user to specify where a problem report text file is written — this file may be sent to our support resources for debugging or solutions.

The contents of the problem report are likely a stacktrace from the underlying exception along with any necessary execution context plus some standard CT boilerplate text describing what the file is, when it was written out, perhaps the amount of free memory in the JVM, etc.

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☒ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☒ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☒ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.