CentraSite

CentraSite Application Framework

Version 9.5 SP1

November 2013

This document applies to CentraSite Version 9.5 SP1.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: IINM-DG-CSAF-95SP1-20140410

Table of Contents

Preface

The description of the CentraSite Application Framework consists of the following sections:

Introduction
Configuration
Mapping Beans to Registry Objects with Annotations
Querying the Registry
Event Mechanism
Asset Types
Association Types
Lifecycle Management
Revision Management
Multi-User Scenarios
Setting the Classpath
Examples
Javadocs for the Application Framework
Application Framework Error Messages

1 Introduction

The CentraSite Application Framework (CSAF) provides a programming model for developing custom extensions on top of CentraSite. It supports JAXR (Java API for XML Registries) and extends the CentraSite JAXR-based API and the Pluggable UI - the framework on which the CentraSite UI is built.

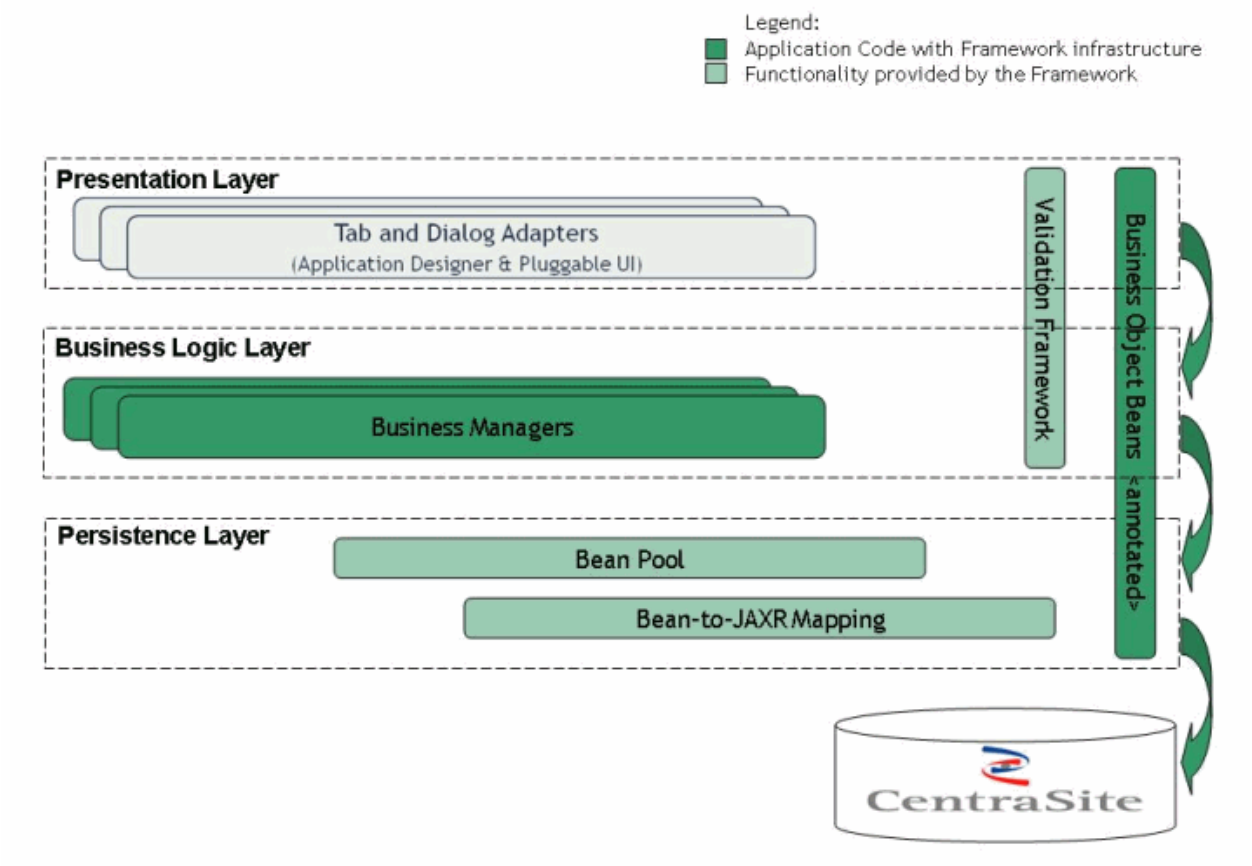
It contains two independent parts: the persistence framework and the validation framework.

The persistence framework provides the ability to operate on registry data using JavaBeans instead of the JAXR-based API. This is done in a fashion similar to object-relational mapping tools such as Hibernate or Java Persistence API. It this case, Java Beans are mapped to registry objects. All this is done declaratively using Java5 Annotations.

This framework was created with the intention of making it easier to work with registries that support the JAXR-based interface, such as CentraSite. Its usage does not require in general any specific or deep knowledge of this API.

A direct benefit of this is shortened application development time.

The validation framework provides an extensible mechanism for validating Java beans. Multiple numbers of constraints can be attached to each bean. The notion of scopes is also supported, i.e., constraints apply only when specific conditions about the bean are met.



This figure above shows the architecture of a common CentraSite application extension developed using CSAF.

There are two major points that have to be clear in order to understand how the persistence framework works, namely how the bean model is built based on the RegistryBean interface and the BeanPool.

The following topics are discussed in this chapter:

RegistryBean

The *RegistryBean* (*com.softwareag.centrasite.appl.framework.beans.RegistryBean*) interface has to stay on top of each bean model hierarchy. It contains the properties that a registry object would have, namely a key and a name. Implementing is the only restriction the framework on the application bean model. The user can use *DynamicRegistryBean* (*com.softwareag.centrasite.appl.framework.beans.DynamicRegistryBean*) for implementation of RegistryBeans. It implements *RegistryBean* and *RevisionBean* (*com.softwareag.centrasite.appl.framework.beans.RevisionBean*), which is the revision-aware extension of the RegistryBean interface.

There is one more option here. If the registry bean needs to be lifecycle-aware, then the user should use the *com.softwareag.centrasite.appl.framework.lcm.beans.LifeCycleAware* interface instead of *RegistryBean*. Its implementation is handled by *com.softwareag.centrasite.appl.framework.lcm.beans.LCAwareDynamicRegistryBean*.

BeanPool

The *BeanPool* (*com.softwareag.centrasite.appl.framework.persistence.BeanPool*) is the main interface with which the application interacts in order to use the persistence framework. All CRUD (create, read, update, delete) operations search via this interface, and registry queries are done via this interface. The user must be aware that the BeanPool instances are not thread safe. There can be only one beanPool per SessionContext. CSAF provides the functionality to create beanPool instances by using `SessionContext.createBeanPool();`. The beanPool can be accessed by `SessionContext.getCurrentBeanPool();`. This method returns the BeanPool instance that is associated with the given context. The *CurrentBeanPoolContext* interface defines the contract for implementations which knows how to scope the notion of a current bean pool. An implementation of this interface is provided as *ThreadLocalCurrentBeanPoolContext*, which maintains current bean pools for the given execution thread. This functionality is extensible, so users can create their own context by implementing *CurrentBeanPoolContext*.

More information about these interfaces can be found in the Javadoc of the framework.

StandaloneRegistryProvider

In order to obtain a connection to the repository, an instance of *StandAloneRegistryProvider* must be created. This registry provider has several important parameters for its creation that will affect the functionality of CSAF. CSAF supports several constructors which exclude some of the properties and use their default values instead. The constructor with full parameter list is:

```
StandaloneRegistryProvider(String registryUrl, String user, String ↩
password, boolean browserBehaviour){}
```

registryUrl	Specifies the URL to the CentraSite registry. Default value is http://localhost:53307/CentraSite/CentraSite
user	The username of the registry user
password	The password of the registry user
browserBehaviour	Sets the "com.centrasite.jaxr.BrowserBehaviour" property of the connection factory. To enable type management, this flag must be set to "true"; to enable RevisionManagement it must be "false". Default value is "false".

Example for creating a BeanPool instance by using SessionContext and StandaloneRegistryProvider

```
SessionContext context = null;
RegistryProvider provider = null;
try {
    provider = new StandaloneRegistryProvider(registryUsername,
                            registryPassword, true);

  Configuration conf = new Configuration();
                conf.setRegistryProvider(provider);

                conf.addBeanType(Item.class);
                conf.addBeanType(Action.class);
                conf.addBeanType(Entry.class);
                conf.addBeanType(ExternalLink.class);
                context = SessionContext.createInstance(conf);
          } catch (CSAppFrameworkException e) {
                // Do something with the exception
          }

BeanPool beanPool = context.getCurrentBeanPool();
```

2 Configuration

You can configure the CentraSite Application Framework via the *Configuration* object (*com.softwareag.centrasite.appl.framework.Configuration*). The following can be configured here:

- Bean types managed by CSAF
- Persistence mode
- Bean mode
- Maximum concept cache size
- Cache scope
- Re-reading of outdated objects

Additionally, the configuration object supports a generic property: key/name pair. It can used to configure any of the above mentioned properties generically.

After the *Configuration* object has been initialized, it can be passed to the `com.softwareag.centrasite.appl.framework.SessionContext.createInstance()` method, which creates a *SessionContext* instance. This instance can then create *com.softwareag.centrasite.appl.framework.persistence.BeanPool* instances and can be used for the lifetime of the application.

The following topics are discussed in this chapter:

Bean Types Managed by CSAF

The framework keeps an internal data model for user-defined bean classes, i.e., bean classes that extend the *com.softwareag.centrasite.appl.framework.beans.RegistryBean* interface. After the bean interfaces have been defined as Java classes having the *@RegistryObject* annotation, they must be registered by calling the `Configuration.addBeanType(java.util.Class)` method.

In principle, calling `Configuration.addBeanType(java.util.Class)` for each bean class is not mandatory, since CSAF tries to process this information (configuration) at runtime when required. Nevertheless, it is still highly recommended because there are cases in which it is not possible to obtain the mapping information at runtime, e.g., when performing a search in the registry.

Bean Modes

The framework supports two bean modes: BACKED and SIMPLE (`com.softwareag.centrasite.appl.framework.persistence.BeanMode`). This mode specifies how the beans interact with the underlying implementation of the API supporting JAXR.

When using the SIMPLE mode, data from the bean is transferred to the registry object only when the user explicitly requests this by calling one of the BeanPool methods (`update()`, `flush()`,`delete()`).

When using the BACKED mode, data from the bean is transferred to the registry object immediately after it is set in the bean. The advantage of this is that extra features such as locking and caching can be used.

Note: SIMPLE mode is deprecated; BACKED mode should always be preferred.

Persistence Modes

The framework supports two persistence modes: FULL and MAP_ONLY. This mode specifies how and whether the data will be persisted in the registry.

When using the FULL mode, the data is entirely persisted in the registry. This is the default mode.

When using the MAP_ONLY mode, the data is not persisted in the registry at all; it is just mapped from the bean to the registry object. It is assumed that the persistence is done outside of the framework. This is used, for example, in a Pluggable UI environment, and applies to any custom extension of the CentraSite UI. In this, the Pluggable UI takes care of storing the object in the registry.

Cache Configuration

Two properties of the caching can be configured: the maximum concept cache size and the cache scope. Both parameters configure the concept mapping cache within the framework.

The default value for the cache size is 1000.

There are two available scopes for the cache:

APPLICATION	There is one cache for the whole application.
SESSION	Each session has its own cache.

Re-Reading Outdated Objects

The framework provides support to re-read outdated registry beans automatically. This is controlled by the `Configuration.PROP_AUTO_REREAD_OUTDATED_OBJECTS` property. Possible values are "true" and "false".

If the property value is set to "true" then when an outdated object is modified by the system it will automatically be re-read from the registry, i.e., reverted to the latest state in the database, before applying any changes. Otherwise the user receives a *com.softwareag.centrasite.appl.framework.persistence.ObjectOutdatedException* when performing the modification. Note that if this feature is turned on the current client of CSAF will override any changes made by another client.

3 Mapping Beans to Registry Objects with Annotations

The following topics are discussed in this chapter:

Introduction to Bean Mapping

The beans are mapped to registry objects using Java5 Annotations.

Each bean from the application bean model has to extend or implement the *RegistryBean* (*com.softwareag.centrasite.appl.framework.beans.RegistryBean*) interface. If an interface extends the RegistryBean interface, an implementation must be provided and specified using the @Bean annotation:

```
@RegistryObject(objectTypeName="{http://namespaces.CentraSite.com/csaf}Item")
@Bean(implementationClass = "...")
public interface Item extends RegistryBean{
...
}
```

The table below describes the annotations currently supported by the CentraSite Application framework.

Annotations	Scope	Properties	Description
@RegistryObject	Type	`objectTypeName` (optional) – the name of the object type of the registry object. `objectTypeKey` (optional) – the key of the object type. At least one of the properties must be specified.	Maps a bean to a registry object with a specific object type.
@Property	Method	`target` (optional) – the name of the target property in the registry object. The property must be a standard property of a predefined JAXR-based object type. If the target property is not specified, it is assumed that it matches the name of the bean property.	Maps a bean property to a registry object property. The properties should have the same type. The mapper does not provide type conversion, except for JAXR InternationalString to/from String.
@Slot	Method	`Name` (mandatory) – the name of the slot to which this property is to be mapped. The JAXR-based property being mapped can be custom defined, or the JAXR-based object type that	Maps a bean property to a registry object slot. Multivalue slots are supported. Also provided are type conversion slot values which are string to integer, Boolean, date, timestamp and Calendar.

Annotations	Scope	Properties	Description
		this property comes from can be custom. `targetType` (optional) – specifies the type of the bean property. It is used when the property is a collection and thus the mapping cannot guess the underlying property. `type` (optional) – the type of the bean property. Supported types are BOOLEAN, DATE, CALENDAR, TIMESTAMP, INTEGER and AUTO. The latter allows the mapper to guess the property type.	
@Slots	Method	`targetType` (mandatory) - the type of bean that is to be mapped to a single slot.	Maps all slots of a registry object to a bean property (Collection).
@SlotProperty	Method	`target` (mandatory) – can be one value from the enum SlotPropertyName – NAME, SLOT_TYPE, VALUES.	Used in conjunction with the @Slots property. Maps the properties of the bean of the type specified as target type with the @Slots annotation. A slot has a name, slot type and values. All these properties can be mapped using this annotation.
@TelephoneNumbers	Method	`type` (optional) – the type of the telephone numbers.	Maps a bean property to the TelephoneNumber object from the JAXR-based infomodel. Such objects are used in the User JAXR Object.
@ExternalLink	Method	`slotName` (optional) – the name of a slot inside the ExternalLink registry object to be mapped that is checked for having a specified value. This is used to pick the proper ExternalLink if the registry object has more than one. `slotValue` (optional) – the value of the slot to be checked. `type` (optional) – type of the bean used for the mapping	Maps a bean property to a ExternalLink JAXR-based object or a collection of them.
@Association	Method	`key` (optional) – the key of the association type to be used. Either	Maps a property to an association. It can be either the association

Annotations	Scope	Properties	Description
		type (see below) or key must be present. `type` (optional) – the association type to be used. Either type or key must be present. `targetType` (optional) – the type of the bean to be mapped. It is used when the bean property is a collection and the type cannot be guessed. `mappedTo` (optional) – the property can be mapped to either the association registry object or the target of the association. `cascadeStyle` (optional) – Supported cascade styles are ALL (Cascade on all operations), UPDATE (Cascade on update operations), DELETE (Cascade on delete operations), NONE (no cascading).	object itself or the target of the association.
@AssociationTarget	Method	None	Used in conjunction with the @Association annotation. Maps a bean to a target of an association. It is used when a bean is mapped to an association object using the @Association annotation. Then inside that bean a property must be mapped to the target.
@Classification	Method	`classificationScheme` (optional) – the name of the ClassificationScheme to be used. `parentConcept` (optional) – the path of the parent concept. Used when mapping enumeration classifications. `parentConceptKey` (optional) – the key of the parent concept. Either the path or the key can be used. `conceptPath` (optional) – the path of the concept for this classification. `conceptKey` (optional) – the key of the concept for this classification.	Maps a bean property to a classification. Both the classification object and its concept can be used. The mapping can be simple – Bean property <-> Classification(Concept) or enumeration – Bean property <-> Classification (Concept) which concept is under a specified parent concept. The latter provides a set of predefined possible concepts, thus is similar to the notion of enumeration.

Annotations	Scope	Properties	Description
		Either the path or the key can be used. `targetType` (optional) – the type of the bean used for the mapping. Required when the property is a collection and the type cannot be guessed. `mappedTo` (optional) – the bean can be mapped either to the classification object or to its concept. `cascadeStyle` (optional) – The supported cascade styles are ALL, UPDATE, DELETE and NONE.	
@ClassificationConcept	Method	None	Used in conjunction with the @Classification annotation. Maps a bean to the Concept of the Classification specified in the @Classification annotation.
@ClassifiedInstances	Type	`instances` (mandatory) – the array of the instances that this mapping will address.	Maps class hierarchy to registry objects. Classifications are used to achieve this. Each registry object that corresponds to a bean from the hierarchy is classified with a concept. The latter belongs to a taxonomy mirroring the class hierarchy.
@ClassifiedInstance	Type	`classificationScheme` (mandatory) – the classification scheme to which the concept belongs. Either the scheme name or the key must be specified. `classificationSchemeKey` (mandatory) – the key of the classification scheme. Either the scheme name or the key must be specified. `conceptKey` (mandatory) – the key of the concept used to classify this instance. `conceptPath` (mandatory) – the path of the concept used to classify this instance.	Sets the information for a specific mapping between a bean from the hierarchy and a registry object.

Annotations	Scope	Properties	Description
		`beanType` (mandatory) – the type of the bean that corresponds to this instance.	
@ClassificationAttribute	Method	`attributeName` (mandatory) – The name of the attribute represented by this annotation. `cascadeStype` (optional) – The cascading style for this mapping. `targetType` (optional) – The type of the mapped bean. The bean itself must be of type Concept.	Annotation for mapping the return value of a (getter) method to the classification attribute specified at type level. The attribute name is mandatory and is used to identify the attribute. This annotation is very similar to the {@link Classification} annotation in terms of supported attributes and underlying representation. The difference is that the taxonomy is obtained from the attribute description. In order to use this annotation, a classification attribute must be defined at type level (the registry object type must have a classification attribute with the same attribute name as specified in the annotation).
@FileAttribute	Method	`attributeName` (mandatory) – The name of the attribute represented by this annotation. `cascadeStype` (optional) – The cascading style for this mapping. `targetType` (optional) – The type of the mapped bean. The bean itself must be of type ExternalLink.	Annotation for mapping the return value of a (getter) method to the file attribute specified at type level. The attribute name is mandatory and is used to identify the attribute. This annotation is very similar to the {@link ExternalLink} annotation in terms of supported attributes and underlying representation. In order to use this annotation, a file attribute must be defined at type level (the registry object type must have a file attribute with the same attribute name as specified in the annotation).
@Relationship	Method	`attributeName` (mandatory) – The name of the attribute represented by this annotation. `cascadeStype` (optional) – The cascading style for this mapping.	Annotation for mapping the return value of a (getter) method to the attribute specified at type level. The attribute name is mandatory and is used to identify the attribute. This annotation is very

Annotations	Scope	Properties	Description
		`targetType` (optional) – The type of the mapped bean. The bean itself must be of type Concept.	similar to the {@link Association} annotation in terms of supported attributes and underlying representation. The difference is that the association and target types are not specified but are obtained from the attribute description. In order to use this annotation, a relationship attribute must be defined at type level (the registry object type must have a relationship attribute with the same attribute name as specified in the annotation).

Example:

```
/**
 * Java bean interface representing JAXR-based registry objects of type  ↩
ServiceInterfaceVersion.
 */
@RegistryObject(objectTypeName = ↩
"{http://namespaces.CentraSite.com/csaf}ServiceInterfaceVersion")
@Bean(implementationClass = ↩
"com.softwareag.centrasite.appl.framework.persistence.beanmodel.impl.ServiceInterfaceVersionImpl")
public interface ServiceInterfaceVersion extends RegistryBean{

    @Property(target = "name")
    public String getName();
    public void setName(String name);

    /**
     * Returns the description
     */
    @Property(target = "description")
    public String getDescription();

    /**
     * Sets the description
     */
    public void setDescription(String description);

    /**
     * Returns the attachments
     */
    @ExternalLink(type = ↩
com.softwareag.centrasite.appl.framework.persistence.beanmodel.ExternalLink.class)
   ↩
```

```
 public List<com.softwareag.centrasite.appl.framework.persistence.beanmodel.ExternalLink> ↩
getAttachments();

    /**
     * Sets the attachments
     */
    public void setAttachments(
   List<com.softwareag.centrasite.appl.framework.persistence.beanmodel.ExternalLink> ↩
attachments);

    /**
     * Returns the short name of the interface version. Maps to ↩
{http://namespaces.CentraSite.com/csaf}shortName slot.
     */
     @Slot(name = "{http://namespaces.CentraSite.com/csaf}shortName")
     String getShortName();

    /**
     * Sets the short name property of the interface version.
     */
    void setShortName(String shortName);

    /**
     * Returns.
     */
   @Association(type = "HasReviewRequest", targetType = ReviewRequestOutcome.class, ↩
cascadeStype = CascadeStyle.DELETE)
    List<ReviewRequestOutcome> getReviewRequestOutcomes();

    /**
     * @param list
     */
    public void setReviewRequestOutcomes(List<ReviewRequestOutcome> list);

    /**
     * Returns the findings, which are attached to the bean.
     */
    @Classification(classificationScheme = "CSAF -Taxonomy", conceptPath = ↩
"/ClassificationInstances/Finding", targetType = Finding.class)
    List<Finding> getFindings();

    /**
     *
     * @param pFindings
     */
    public void setFindings(List<Finding> pFindings);


    @Slots(targetType = SlotBean.class)
    public Collection<SlotBean> getSlots();

    public void setSlots(Collection<SlotBean> slots);
```

```
}

/**
 * Implementation of the {@link ServiceInterfaceVersion} bean interface.
 */
public class ServiceInterfaceVersionImpl extends DynamicRegistryBean implements
ServiceInterfaceVersion {

    private String _shortName;
    private List<ReviewRequestOutcome> _reviewRequestOutcomes;
    private Collection<SlotBean> slots;
    private String _instanceSlotName;
    private List<Finding> findings;
    private List<ExternalLink> externalLinks;

    /**
     * {@inheritDoc}
     */
    public String getShortName() {
        return _shortName;
    }

    /**
     * {@inheritDoc}
     *
     * The setter is annotated that modifies the object and it needs to be
     * updated in the JAXR-based registry.
     */
    public void setShortName(String shortName) {
        _shortName = shortName;
    }

    public List<ReviewRequestOutcome> getReviewRequestOutcomes() {
        return _reviewRequestOutcomes;
    }

    public void setReviewRequestOutcomes(List<ReviewRequestOutcome> list) {
        _reviewRequestOutcomes = list;
    }

    public Collection<SlotBean> getSlots() {
        return slots;
    }

    public void setSlots(Collection<SlotBean> slots) {
        this.slots = slots;
    }

    public String getInstanceSlotName() {
        return _instanceSlotName;
    }
```

```
    public void setInstanceSlotName(String slotName) {
        _instanceSlotName = slotName;
    }

    public List<Finding> getFindings() {
        return findings;
    }

    public void setFindings(List<Finding> findings) {
        this.findings = findings;
    }

    public List<ExternalLink> getAttachments() {
        return externalLinks;
    }

    public void setAttachments(List<ExternalLink> attachments) {
        externalLinks = attachments;
    }
}
```

Standard Mappings

The Standard Mappings (*com.softwareag.centrasite.appl.framework.beans.standard*) are RegistryBeans that represent all supported JAXR-based Registry Objects under the package *com.centrasite.jaxr.infomodel*. They provide the functionality to operate and manage JAXR-based RegistryObjects through the Application Framework with ease.

There are other kinds of objects that are included in this package although they are not RegistryObjects (*EmailAddress*, *PostalAddress*, *Slot* … etc.). The Application Framework provides a mapping for them as well. Standard Mapping instances are created by the BeanPool's `create(beanClass)`; standard non-registry object mappings (*EmailAddress*, *PostalAddress*, *Slot* … etc.) are managed using the *com.softwareag.centrasite.appl.framework.beans.standard.StandardMappingManager*.

For more information about the methods and functionality supported by Standard Mappings API, please refer to the Javadoc of the framework.

Standard Mappings Usage Sample

```
//Create a com.softwareag.centrasite.appl.framework.beans.standard.Organization

com.softwareag.centrasite.appl.framework.beans.standard.Organization organization =
    ↩
beanPool.create(com.softwareag.centrasite.appl.framework.beans.standard.Organization.class);
organization.setName("MyOrganization");

// Create StandardMappingManager for managing Standard non RegistryObjects //mappings

StandardMappingManager smm = new StandardMappingManager(registryProvider);

//Create a postal address

com.softwareag.centrasite.appl.framework.beans.standard.PostalAddress pa =
    smm.createPostalAddress("streetNumber", "street", "city",
                            "stateOrProvince", "country", "postalCode","type");
organization.setPostalAddress(pa);

// Get existing user and add it to the organization

com.softwareag.centrasite.appl.framework.beans.standard.User user =
    ↩
beanPool.read(com.softwareag.centrasite.appl.framework.beans.standard.User.class, USER_KEY);
Collection<User> users = new ArrayList<User>();
users.add(user);
organization.setUsers(users);

// save the changes

beanPool.flush();
```

Generating Beans with the Command Line

You can use a command line utility to generate registry beans.

- Under Windows
- Under Linux

- Input Parameters

Under Windows

Use the following procedure to generate registry beans under Windows:

1. Open *GenerateCSAFBeans.cmd* in a text editor.
2. Add the following property statement:

```
<CentraSiteInstallDir>\utilities\ GenerateCSAFBeans.cmd -user <USERNAME> -pass ↩
<PASSWORD> -url <CENTRASITE-URL> -typename <TYPENAME> -interfacepackage ↩
<INTERFACEPACKAGE> -implpackage <IMPLPACKAGE> -destination <DESTINATION>
```

where, *<CentraSiteInstallDir>* is the CentraSite installation directory. By default, this is the *CentraSite* folder under *<SuiteInstallDir>*.

Under Linux

Use the following procedure to generate registry beans under Linux:

1. Open *GenerateCSAFBeans.sh* in a text editor.
2. Add the following property statement:

```
<CentraSiteInstallDir>/utilities/ GenerateCSAFBeans.sh -user USERNAME -pass ↩
PASSWORD -url CENTRASITE-URL -typename TYPENAME -interfacepackage INTERFACEPACKAGE ↩
-implpackage IMPLPACKAGE -destination DESTINATION
```

where, *<CentraSiteInstallDir>* is the CentraSite installation directory. By default, this is the *CentraSite* folder under *<SuiteInstallDir>*.

Input Parameters

The following table describes the complete set of input parameters that you can use to generate the registry beans:

Parameter	Description
USERNAME	*Required*. Your CentraSite user ID.
PASSWORD	*Required*. The password for your CentraSite user account.
CENTRASITE-URL	*Required*. The fully qualified URL for the CentraSite registry/repository. If you omit this parameter, the importer assumes that the registry/repository resides at `http://localhost:53307/CentraSite/CentraSite`. **Note:** If the registry/repository is running on a different machine and port number, you can use this parameter to specify its location instead of using the individual -h

Parameter	Description
	and -p parameters. (If you specify the -dburl parameter with the -h and/or -p parameters, the -h and -p parameters will be ignored.)
TYPENAME	*Required*. The namespace and name of the type to be generated (mandatory). Example: {http://test}TestService. Or, the name of the virtual type to be generated. Example: "Virtual service" Note: The quotation marks are necessary, in order that "Virtual service" is parsed as a single token.
INTERFACEPACKAGE	*Required*. The name of the package in which the interfaces should be generated. For example: `com.sag.generated`
IMPLPACKAGE	*Required*. The name of the package in which the implementation should be generated. For example: `com.sag.generated.impl`
DESTINATION	*Required*. The location where the generated bean will be stored.

4 Querying the Registry

The Application Framework provides two search functionalities:

- The ***Application Framework Simple Search*** uses only framework-specific data, so it is simpler to use and supports all needed query operations. This search interface is also the recommended one to use.
- The ***Application Framework JAXR-based Search*** combines framework and JAXR-based data. The advantage of this search is that it can use the whole JAXR-based functionality to query the registry. The disadvantage is that in order to use it, the user must have considerable knowledge of JAXR.

Application Framework Simple Search

The Application Framework Simple Search uses framework-specific data only and its usage has been made as simple as possible.

1. **Creating a search object**
   In order to search the registry, the user must create a search object using a BeanPool instance. The BeanPool offers several methods for creating search objects:

   - **Without arguments:**

     ```
     BeanPool.createSearch();
     ```

     This creates a search object which, when executed, searches for objects in the registry from all registered bean types (see the section **Bean Types Managed by CSAF**).

   - **When a List of items is passed:**

     ```
     BeanPool.createSearch(List<Class<? extends RegistryBean>> beanClasses)
     ```

     The created search object searches through all objects in the registry, from the specified list of types.

   - **When a single type is passed:**

     ```
     BeanPool.createSearch(Class<? extends RegistryBean bean> beanClass)
     ```

     The search object searches the registry only for items from the specified type.

   The search object has a `result()` method which searches the registry and returns a list of all RegistryBean objects that satisfy the search criteria.

   Example:

```
BeanPool beanPool = sessionContext.getCurrentBeanPool();

Search search = beanPool.createSearch();
```

2. **Adding search predicates**

   Predicate explanation

   The predicate is an object representation of a query criterion used to restrict the search results. Predicates can be created from a factory-like class called Predicates (`com.softwareag.centrasite.appl.framework.persistence.search.Predicates`).

   It provides two static methods for creating each specific predicate:

   - **Without specifying Bean Type:**

     ```
     Predicates.eq(String propertyName, Object value)
     ```

   - **By specifying a Bean Type:**

     ```
     Predicates.eq(String propertyName, Object value,Class<? extends RegistryBean> ↩
     beanType)
     ```

     where

     - *method name* is the comparison operator:

and	logical conjunction
eq	equal
ge	greater than or equal to
gt	greater than
le	less than or equal to
like	matches a string that can include wildcards
lt	less than
ne	not equal
or	logical disjunction

     - *property name* is the name of the property to be compared. This property name is a string value representing the name of the Java property (*getName()* corresponds to “name”). The search functionality supports adding a sequence of properties. This is accomplished by knowing the searched RegistryBean property hierarchy and by separating following properties with a dot ".".

       Example:

Predicates.*eq*("externalLink.uri ", value)	The predicate is created for the URI property of the externalLinks of the searched RegistryBean, which should be equal to the given value.

- *value* is the value to compare against. Most methods expect an Object value because the search can handle a variety of objects including String, Number, Date, Calendar, Key, RegistryBean and others. There are also methods that expect a specific value type. An example is like(`String propertyName`, `String value`), which supports wildcards and therefore the expected value type is String. Other object types that are worth mentioning are the so-called support types (TelephoneNumbers, InternationalString, LocalizedString, EmailAddress, PostalAddress, etc.). They can be used for search criteria but not as a searched object because they are not registry beans. For example, the following search is valid:

```
Search search = beanPool.createSearch(User.class);
Predicates.eq("telephoneNumbers.countryCode", "someCountryCode");
```

  But the following search is not valid:

```
Search search = beanPool.createSearch(EmailAddress.class);
```

- *beanType* is the bean type for which the predicate will be applied.

  Important: If no beanType is specified then the predicate is applied to the first bean type in the Search object's list of bean types. Note that the first item of that list must support the property passed to the predicate, otherwise the search will fail. In cases where the search object is created for all supported bean types, the list is filled randomly so the user must be aware of all common properties supported by these RegistryBean types.

Each predicate can be added to the search object by invoking the search method:

```
addPredicate(Predicate predicate);
```

A search object can add multiple predicates, which can be treated as predicates joined by an "and" operator. For example:

```
Search search =  beanPool.createSearch();
search.addPredicate(predicate1);
search.addPredicate(predicate2);
search.addPredicate(predicate3);
```

is equal to "predicate1 and predicate2 and predicate3" in the query to be executed.

There are two more methods in the Predicates class: `and(Predicate p1, Predicate p2)` and `or(Predicate p1, Predicate p2)`. These methods create a so-called combine predicate.

They join two predicates by logical conjunction or logical disjunction respectively. This predicate can be added to the search object in the same way as the common predicates explained above.

Supported predicates description

All supported predicates are created from methods in the Predicates class (`com.softwareag.centrasite.appl.framework.persistence.search.Predicates`).

Like Predicate

A predicate that supports usage of wildcards. The value field of the creating methods:

```
like(String propertyName, String value)
like(String propertyName, String value, Class<? extends RegistryBean> beanType)
```

is of Type String, so the user may add strings (possibly including wildcards).

Example:

```
like("name","%partOfExpectedName");
```

Wildcards

The like predicate supports wildcards in the manner of SQL and UDDI. For more details, refer to the section *About wildcards* in the UDDI specification: **http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908082**. The wildcard characters are as follows:

Wildcard character	Indicates
%	Any value for any number of characters
_	Any value for a single character

The following special cases are supported:

To represent...	use the character string...
%	\%
_	_
\	\\

Greater Than Predicate

A predicate that compares Number, Date or Calendar, returning true if the compared object value is greater than the value given in the predicate's creating method “value” field:

```
gt(String propertyName, Object value)
gt(String propertyName, Object value, Class<? extends RegistryBean> beanType)
```

The value must be one of the following types: Number, Date, Calendar.

Example:

```
Calendar calendar = Calendar.getInstance();
Predicate predicate = Predicates.gt("requestDate",calendar);
```

Less Than Predicate

A predicate that compares Number, Date or Calendar, returning true if the compared object value is less than the value given in the predicate's creating method “value” field:

```
lt(String propertyName, Object value)
lt(String propertyName, Object value, Class<? extends RegistryBean> beanType)
```

The value must be one of the following types: Number, Date, Calendar.

Example:

```
Predicate predicate = Predicates.lt("copyNumber",203);
```

Greater or Equal Predicate

A predicate that compares Number, Date or Calendar, returning true if the compared object value is greater than or equal to the value given in the predicate's creating method “value” field:

```
ge(String propertyName, Object value)
ge(String propertyName, Object value, Class<? extends RegistryBean> beanType)
```

The value must be one of the following types: Number, Date, Calendar.

Example:

```
Predicate predicate = Predicates.ge("copyNumber",203);
```

Less or Equal Predicate

A predicate that compares Number, Date or Calendar, returning true if the compared object value is less than or equal to the value given in the predicate's creating method “value” field:

```
le(String propertyName, Object value)
le(String propertyName, Object value, Class<? extends RegistryBean> beanType)
```

The value must be one of the following types: Number, Date, Calendar.

Example:

```
Predicate predicate = Predicates.le("copyNumber",203);
```

Equal Predicate

A predicate that returns true if the compared object value is equal to the value given in the predicate's creating method “value” field:

```
eq(String propertyName, Object value)
eq (String propertyName, Object value, Class<? extends RegistryBean> beanType)
```

The value must be one of the following types: Number, Date, Calendar, String, Key, RegistryBean.

If the value is of type RegistryBean then the comparison is made by the RegistryBean's key.

Example:

```
Predicate predicate = Predicates.eq("name","somePropertyname");
```

Not Equal Predicate

A predicate that returns true if the compared object value is not equal to the value given in the predicate's creating method “value” field:

```
ne(String propertyName, Object value)
ne (String propertyName, Object value, Class<? extends RegistryBean> beanType)
```

The value must be one of the following types: Number, Date, Calendar, String, Key, RegistryBean.

If the value is of type RegistryBean then the comparison is made by the RegistryBean's key.

Example:

```
Predicate predicate = Predicates.ne("name","somePropertyname");
```

AND Predicate

A predicate that joins two predicates in a logical conjunction. The method that creates this predicate:

```
public static Predicate and(Predicate p1, Predicate p2)
```

expects two predicates as arguments.

Example:

```
Predicate predicate1 = Predicates.eq("name","somePropertyname");
Predicate predicate2 = Predicates.eq("name","somePropertyname2");
Predicate andPredicate = Predicates.and (predicate1, predicate2);
```

OR Predicate

A predicate that joins two predicates in a logical disjunction. The method that creates this predicate:

```
public static Predicate or(Predicate p1, Predicate p2)
```

expects two predicates as arguments.

Example:

```
Predicate predicate1 = Predicates.eq("name","somePropertyname");
Predicate predicate2 = Predicates.eq("name","somePropertyname2");
Predicate orPredicate = Predicates.or (predicate1, predicate2);
```

3. Order can be created by using one of the Order (com.softwareag.centrasite.appl.framework.persistence.search.Order) static methods: `asc(String propertyName)` or `desc(String propertyName)`, creating ascending or descending order respectively for a given property. The rules for the property name when creating Order are the same as when creating a Predicate. The user must know whether the bean types added to the search object support the property passed to the Order asc(String propertyName) or desc(String propertyName) methods. Multiple orders may be added to the search object.

   Example:

   ```
   Order order = Order.asc("description");
   ```

4. After adding the necessary predicates and orders to the search object, the search can be executed by invoking the `result()` method on the search object. It returns a list of all RegistryBean objects in the registry that applied the predicate conditions in the specified order. The result is lazy loading compatible.

   Here is an example of a Search lifecycle:

   ```
   List searchTypes = new ArrayList();
   searchTypes.add(ReviewRequestOutcome.class);
   searchTypes.add(ServiceInterfaceVersion.class);

   Search search = beanPool.createSearch(searchTypes);

   Predicate predicate1 = Predicates.eq("ExternalLink.URI", ↩
   "http://www.softwareag.com");
   Predicate predicate2 = Predicates.eq("name","somePropertyname2");
   Predicate orPredicate = Predicates.or (predicate1, predicate2);
   ```

```
Search.addPredicate(orPredicate);

search.addOrder("name");

List<RegistryBean> result = (List<RegistryBean>) search.result();
```

This means that all ReviewRequestOutcomes and ServiceInterfaceVersions will be searched and the ones that have name equal to “somePropertyname2” or ExternalLink with URI equal to “http://www.softwareag.com” will be returned in the resulting List of RegistryBean objects ordered by name.

Extensibility

There are several points where the user can extend the existing Application Framework functionality.

Properties

Each Java bean property is internally represented as a *com.softwareag.centrasite.appl.framework.mapping.Property* instance.

The recommended way of creating a new property is by extending, directly or indirectly, the `BaseProperty` class (com.softwareag.centrasite.appl.framework.mapping.BaseProperty).

In order to map the information from a given annotation to the new Property correctly, a user-defined Property Processor that implements the PropertyAnnotationProcessor (com.softwareag.centrasite.appl.framework.PropertyAnnotationProcessor) must be created.

Then the newly created PropertyProcessor must be added to the list of processors in the BeanTypeAnnotationProcessor (com.softwareag.centrasite.appl.framework.BeanTypeAnnotationProcessor) using the `addAnnnotationProcessor(Class<?> annotationType, PropertyAnnotationProcessor annotationProcessor)` method.

Property Mapper

Each property value must be transferred to/from the underlying registry object. For that purpose, CSAF provides the (com.softwareag.centrasite.appl.framework.persistence.mapper.PropertyMapper) interface.

Users can provide their own implementation of the PropertyMapper interface by hooking it to a given type of Property. Such a property mapper is registered using the `com.softwareag.centrasite.appl.framework.persistence.mapper.PropertyMapperFactory.addHandler(PropertyMapperFactory.Handler)` method.

Predicate

The preferred method of creating a custom defined predicate is to extend the `DefaultPredicate` (com.softwareag.centrasite.appl.framework.persistence.search.impl.DefaultPredicate) class directly or indirectly. Another way is to directly implement the Predicate interface (com.softwareag.centrasite.appl.framework.persistence.search.Predicate), although this is not recommended because it does not offer default behavior.

In order to use this newly-created predicate, the user must create a custom defined predicate handler, which must implement the PredicateHandler interface (com.softwareag.centrasite.appl.framework.persistence.search.PredicateHandler). This predicate handler must be added to the PredicateFactory (com.softwareag.centrasite.appl.framework.persistence.search.impl.PredicateFactory) list of predicate handlers by calling addPredicateHandler(PredicateHandler handler).

Application Framework JAXR-Based Search

Whereas the BeanPool interface takes care of the standard CRUD operations to the registry, the queries are performed using the Query interface (*com.softwareag.centrasite.appl.framework.persistence.Query*):

```
package com.softwareag.centrasite.appl.framework.persistence;
public interface Query<T extends RegistryBean> {
    List<T> run(QueryContext pContext) throws JAXRException, CSAppFrameworkException;
}
```

In order to do a query, one should implement this interface and place the querying routines in the `run()` method implementation. The query is then executed via `BeanPool.run()`:

```
<T extends RegistryBean> List<T> run(Query<T> pQuery) throws CSAppFrameworkException;
```

The returned data is then in the form of beans.

This mechanism still requires knowledge of JAXR. The benefit is that JAXR is isolated in this interface. Below is a sample implementation of Query:

```
final Query<EntryCode> q = new Query<EntryCode>() {
            public List<EntryCode> run(QueryContext context) throws JAXRException {
                final RegistryAccessor regDAO = context.getRegistryAccessor();
                final Concept concept = regDAO.findConceptByPath("CSAF-Taxonomy",
                                "/ClassificationInstances/EntryCodeType");
                final List<EntryCode> result = new ArrayList<EntryCode>();
              for (Concept c : (Collection<Concept>) concep.getChildrenConcepts()) {
                    try {
                        EntryCode ec = ↩
context.getCurrentBeanPool().read(EntryCode.class, c.getKey().getId());
                        result.add(ec);
```

```
                    } catch (Exception e) {
                        throw new RuntimeException(e.getMessage(), e);
                    }
                }
                return result;
            }
        };
List<RegistryBean> queryResult = getBeanPool().run(q);
```

In general, a Query would use the JAXR-based API to find and retrieve the data, and then the keys of registry objects that were found are passed to the BeanPool to build the beans. These beans are then returned as the result of the query execution.

5 Event Mechanism

The CSAF allows the user to register and receive notifications when certain events occur. Currently, three persistence events are supported: objectDeleted, objectCreated, objectUpdated. These events can be intercepted by implementing the interface *com.softwareag.centrasite.appl.framework.persistence.PersistenceEventListener*. Such listeners are registered via the BeanPool, which has methods for adding, removing and retrieving listeners.

All of the supported events are post events; in other words, they are fired after an action has been performed. CSAF currently does not support any pre-action events. However, such events are being planned for upcoming versions.

6 Asset Types

Type Management provides CRUD (create, read, update and delete) operations for custom object types. CSAF provides its own classes describing object (asset) types and their attributes. Type Management supports operations on the following attributes: `file`, `classification`, `relationship` and `slot`, where `slot` can be one of the following types:

- xs:boolean
- xs:dateTime
- xs:date
- xs:time
- xs:duration
- xs:anySimpleType
- xs:integer
- xs:string
- xs:anyURI
- xs:double
- xs:decimal

Type Management also provides CRUD operations for profiles, and functionality to associate attributes with profiles and attach profiles to types. A manager interface *com.softwareag.centrasite.appl.framework.types.TypeManager* is the entry point for the application that uses CSAF.

Note: In order to use Type Management functionality, the *StandaloneRegistryProvider* instance must be created with the browserBehaviour flag set to "true". For more information on how to do this, please check the ***Introduction***.

For more information about the methods and functionality supported by Type Management, please check the Javadoc of the framework.

Usage Sample for Type Management

```
private String TYPE_LOCAL_NAME = "TypeLocalName";

private String TYPE_NAMESPACE = "http://test.namespace.test";

private String TYPE_NAME = "{" + TYPE_NAMESPACE + "}"
                                  + TYPE_LOCAL_NAME;

//Get a sessionContext instance

SessionContext sessionContext = initSessionContext();
```

```
// Get a TypeManager instance from sessionContext

TypeManager typeManager = sessionContext.getTypeManager();

// Create a custom object type

TypeDescription typeDescription = typeManager.createType("TypeDisplayName",
                     "TypeDescription",   TYPE_LOCAL_NAME, TYPE_NAMESPACE);

// Create a Classification Attribute

AttributeDescription attrClass = typeManager.createClassificationAttribute(
        "ClassificationAttributeName","ClassificationAttributeDescription",
         Constants.CLASSIFICATION_SCHEME_PRODUCTS);

//Add attribute to custom type

typeDescription.addAttribute(attrClass);

//Create Profile

Profile profile = typeManager.createProfile("ProfileName");

// Create a File Attribute

AttributeDescription attrFile = typeManager.createFileAttribute(
                 "nameFileAttribute", "descriptionFileAttribute");

//Add attribute to profile

profile.addAttribute(attrFile);

//Add profile to custom type

typeDescription.addProfile(profile);

// Save custom type

typeManager.saveType(typeDescription);

//Get custom type by name

TypeDescription type = typeManager.getType(TYPE_NAME);

//Delete custom type

typeManager.deleteType(type);
```

7 Association Types

In general, registry objects can be related to each other via associations. An association belongs to a specified association type. CentraSite supports predefined association types, such as "HasParent" and "Uses"; in addition, you can create custom association types.

In CentraSite, an association type is uniquely identified by its value (for example: "HasParent", "Uses", etc.). The value is specified when the association type is created; it cannot be subsequently modified.

An association type can optionally have one or more locale-specific display names. If no locale-specific display names are specified, the association type's value is used by default.

Each association type has a forward label; this is shown, for example, when a corresponding association is displayed by the impact analysis. See the section *Impact Analysis* in the document *Using the Asset Catalog* for related information.

You can optionally specify a backward label. Multiple association types can share forward and/or backward labels.

The CentraSite Application Framework type management feature provides methods for creating, updating, deleting and finding association types. For details of the methods, please refer to the Javadoc.

Usage Sample for Association Type Management

```
//Get a sessionContext instance
SessionContext sessionContext = initSessionContext();

// Get a TypeManager instance from sessionContext
TypeManager tm = sessionContext.getTypeManager();

AssociationType at = tm.createAssociationType("MyAssociationType", "MyDisplayName", ↩
"MyForwardLabel",
                                              "MyBackwardLabel", Locale.EN);
tm.saveAssociationType(at);

// find an association type by its value
AssociationType myAssociationType = tm.getAssociationType("MyAssociationType");

// find an association type by its display name
myAssociationType = tm.getAssociationTypeByName("MyDisplayName");

// add a display name with a different locale
myAssociationType.setName("MonNom", Locale.FRENCH);
tm.saveAssociationType(myAssociationType);
```

```
// delete an association type
tm.deleteAsssociationType(myAssociationType);
```

8 Lifecycle Management

The Application Framework supports the Lifecycle Model (LCM) functionality. The LCM provides the ability to define and track the life-cycle of a service and also provides a way to define and enforce policies that govern the path of an asset through the lifecycle. As a result, these policies can be automated or enforced consistently. Using registry beans, we now support lifecycle-aware registry beans.

The definition of an LC Model starts with the definition of an LC Model taxonomy. The state model of an LC Model is a standard state model (deterministic finite automaton, DFA). The model itself is represented as the concepts of the LC Model taxonomy. A taxonomy is not defined for this, so associations are used to represent the state transitions. The states themselves are just concepts within the taxonomy.

In order to create a lifecycle-aware registry bean, the user must create a registry bean that extends *com.softwareag.centrasite.appl.framework.lcm.beans.LifeCycleAware*. Also, the implementation of this registry bean must extend the *com.softwareag.centrasite.appl.framework.lcm.beans.LCAwareDynamic-RegistryBean*. This ensures that the registry bean is lifecycle-aware and is ready to use for lifecycle operations.

In order to manage the lifecycle models and states, the LCM Manager must first be initialized:

```
com.softwareag.centrasite.appl.framework.SessionContext
    sessionContext = initSessionContext();
com.softwareag.centrasite.appl.framework.lcm.LCMAdminManager
    lcmAdminManager = sessionContext.getLCMAdminManager();
```

The *com.softwareag.centrasite.appl.framework.lcm.LCMAdminManager* provides all operations for creating, modifying and deleting LCModels. State models for Lifecycle Management models can theoretically be complex and encompass multiple machines and LCStates.

For more information about the methods and functionality supported by *LCMAdminManager*, please check the Javadoc of the framework.

LCModels are state machines for Lifecycle Management and the state machines may not have any states that cannot be reached. The *com.softwareag.centrasite.appl.framework.lcm.LCModel* provides methods for all operations that can be performed on an LCModel. When the LCModel becomes active, no changes to the LCModel are possible; instead, a new version of the LCModel can be created using `LCModel.createVersion()`.

The *com.softwareag.centrasite.appl.framework.lcm.LCState* provides access to the LCState and state specific operations.

For more information about the methods and functionality supported by *LCModel*, please check the Javadoc of the framework.

Usage Sample for LCM

```
// initialize SessionContext

SessionContext sessionContext = initSessionContext();

// get the LCMAdminManager

LCMAdminManager lcmAdminManager = sessionContext.getLCMAdminManager();

// Create a LCModel

LCModel lcModel = lcmAdminManager.createLCModel();
lcModel.setDisplayName("DisplayName");
lcModel.setDescription("Description");

// the LCModel must set a standard mapping Organization:
//com.softwareag.centrasite.appl.framework.beans.standard.Organization

lcModel.setOrganization((Organization)organization, false);

// Create LCStates

LCState lcStateA = lcModel.createLCState();
String stateAName = "State A";
lcStateA.setName(stateAName);
lcStateA.setDescription("stateADesc");
Collection<LCState> states = new ArrayList<LCState>();
states.add(lcStateA);

// add LCStates to lcModel

lcModel.addStates(states);

//lcModel must set an initial State

lcModel.setInitialState(lcStateA);

// add the keys of all Types that should be enabled for LCM

Collection<String> typesToBeEnabledForLCM = new ArrayList<String>();
typesToBeEnabledForLCM.add(typeToEnableForLCMKeys);
lcModel.addEnabledTypes(typesToBeEnabledForLCM);

//Save the lcModel using the LCMAdminManager

lcmAdminManager.saveLCModel(lcModel);

//Find existing LCModel.
```

```
//The result will contain all LCModels (active and inactive)
//that have the corresponding display name.
List<LCModel> listOfModels = ↩
lcmAdminManager.findLCModelByDisplayName("DisplayName",false);
```

9 Revision Management

CentraSite versioning capabilities make it possible to create a new version of an object at any point in time. However, the new version is per definition a new object instance which has to go through the whole lifecycle again, firing creation policies etc. There is often a demand for versioning capabilities that allow a defined state of the same object to be restored and referenced. Such a defined state is referred to as a checkpoint in the remainder of this document.

The CSAF interfaces related to versioning are *com.softwareag.centrasite.appl.framework.persistence.revision.RevisionManager* and *com.softwareag.centrasite.appl.framework.beans.RevisionBean*.

The CentraSite revisioning feature can be enabled system-wide, which means that every object modification (create/update) of any instance of any type leads to the creation of a checkpoint.

A checkpoint has the following identifying attributes: a minor version number, a label and a timestamp. The minor version number is incremented each time a checkpoint is created. The label is an optional description that can be used to add information about the change. Also a timestamp that reflects the date of the checkpoint creation is recorded with the checkpoint. The creation of a new checkpoint is recorded in the audit log.

It is possible to reference one specific checkpoint of an object directly and retrieve all of its data as it was at the point in time when the checkpoint was created. This implies that changes made to the object after the checkpoint took place are not reflected in the retrieved checkpoint. Note that the checkpoints provide read-only access to the data; any attempt to update a checkpoint raises an exception. However the current object can be updated.

Reading a bean instance from the registry using *BeanPool.read()* always returns the current (latest) state of an object.

Deleting an object also deletes all of its checkpoints.

It is possible to purge a set of checkpoints to reduce the amount of data consumed by keeping older states of the object.

Note that in order to use the Revision functionality, the *StandaloneRegistryProvider* instance must be created with the browserBehaviour flag set to false. For more information on how to do this, please check the ***Introduction*** section.

Usage Sample for Revision Management

```
package com.softwareag.centrasite.appl.framework.persistence.tests;

import java.util.ArrayList;
import java.util.Collection;

import com.softwareag.centrasite.appl.framework.SessionContext;
import com.softwareag.centrasite.appl.framework.beans.RevisionBean;
import com.softwareag.centrasite.appl.framework.beans.standard.Service;
import com.softwareag.centrasite.appl.framework.persistence.BeanPool;
import com.softwareag.centrasite.appl.framework.persistence.revision.RevisionManager;

public class Revisioning {
    private static String checkpointName = "MyLabel";

    public void revisioning() throws Exception {
        SessionContext sessionContext = initSessionContext();
        BeanPool beanPool = sessionContext.getCurrentBeanPool();

        RevisionManager revManager = sessionContext.getRevisionManager();

        //enable the feature if needed
        if (!revManager.isRevisioningEnabled()) {
            revManager.enableRevisioning();
        }

        // create new checkpoint
        Service bean = beanPool.read(Service.class, "uddikey");
        revManager.setCheckpoint(bean, checkpointName);

        // get all checkpoints including the current state object
        Collection<RevisionBean> checkpoints = revManager.getRevisionBeans(bean);

        // restore to the only checkpoint
        Collection<RevisionBean> restoreObjs = new ArrayList<RevisionBean>();
        for (RevisionBean rev : checkpoints) {
            if (rev.isRevision()) {
                restoreObjs.add(rev);
                break;
            }
        }

        revManager.restoreBeans(restoreObjs);

        // delete checkpoints based on label
        revManager.deleteBeans(checkpointName);
    }
```

```
    private SessionContext initSessionContext() {
        //initialize CSAF
          return null;
    }

}
```

10 Multi-User Scenarios

In order to address multi-user scenarios successfully, several aspects of the framework should be noted.

A *SessionContext* is an expensive-to-create, threadsafe object intended to be shared by all application threads. It is created once, usually on application startup, from a *Configuration* instance. A *BeanPool* is an inexpensive, non-threadsafe object that should be used once, for a single request (single unit of work) and then discarded. The *CurrentBeanPoolContext* interface defines the contract for implementations which knows how to scope the notion of a current bean pool. *ThreadLocalCurrentBeanPoolContext*, which maintains current bean pools for the given execution thread, is provided as an example implementation of this interface.

The specification of JAXR does not support transactions or locking. CSAF and CentraSite's implementation extend the API with some locking and transaction capabilities. Here are some points to note:

- Transactions are handled internally and control over them (including isolation, demarcation, etc.) is not exposed through CSAF. There is only support for bulk operations by using the `BeanPool.delete(java.util.Collection)` and `BeanPool.update(java.util.Collection)` methods. These methods guarantee the atomicity of the performed operation. There is also a `BeanPool.flush()` which performs one bulk operation for the deleted beans and one for the created and updated beans.
- Each modification to a registry bean (*RegistryBean* instance) leads to obtaining an exclusive lock for writing on the whole registry object in the database. This is a pessimistic locking strategy, as the lock is obtained when the object is modified and not when it is actually persisted.
- Whenever a lock on a registry object cannot be obtained (because it is taken by another client), a *com.softwareag.centrasite.appl.framework.persistence.LockNotAvailableException* is thrown.
- The notion of an outdated object denotes a registry object whose database representation has been changed since it was read. This is usually caused by a different client modifying the same instance. Trying to modify an outdated object leads to a *com.softwareag.centrasite.appl.framework.persistence.ObjectOutdatedException*. CSAF supports automatic re-reading of outdated objects; this

forces a re-read of the object from the database before applying the changes. See ***Configuration*** for more details on how this can be configured.

In general, the application should minimize the time a registry object is kept locked in the database, i.e., the time during which there are ongoing modifications on it.

11 Setting the Classpath

In order to be able to use the CentraSite Application Framework features, the Java classpath must include all the relevant class files. The easiest way to do this is to include all the JAR files that are contained in the folder *redist* (including the subfolder *redist/csaf*). The redist folder is typically located at *C:\SoftwareAG\CentraSite\redist* (Microsoft Windows) or */opt/softwareag/CentraSite/redist* (UNIX).

12 Examples

The CentraSite Application Framework SDK comes with two examples. One is for the persistence functionality and the other is for the validation functionality.

CRUD Example

The CRUD example demonstrates the abilities of the persistence framework. It shows how the BeanPool is initialized, configured and connected to the registry. Also it shows how CRUD (create, read, update and delete) operations are performed and queries implemented and executed. It also includes the bean model and sample mapping of the most commonly used bean relationships and their JAXR-based representation.