

Ericsson Order Care

Realize Higher Consistency for Faster Time-to-Revenue

Service Registry Configuration Guide



© Ericsson AB 2014

All rights reserved. The information in this document is the property of Ericsson. Except as specifically authorized in writing by Ericsson, the receiver of this document shall keep the information contained herein confidential and shall protect the same in whole or in part from disclosure and dissemination to third parties. Disclosure and disseminations to the receiver's employees shall only be made on a strict need to know basis.



Contents

1	Introduction.....	5
1.1	Purpose and Scope	5
1.2	Reader's Guideline	5
1.3	Overview.....	5
1.4	Software Terms of Use	6
1.5	Service Registry Features.....	6
2	Installation and Setup.....	6
2.1	Quick Start	6
2.2	Detailed Start	7
2.2.1	Configuration Application	8
2.2.2	Velocity Studio	11
2.2.3	System Administration Application	13
2.3	Add Code Tables	14
3	Logical Model.....	16
4	Data Model.....	16
4.1	Main Table Definitions	16
4.1.1	Entity (CWT_SR_ENTITY).....	17
4.1.2	Association (CWT_SR_ASSOCIATION)	19
4.1.3	Extension Fields in the Entity Table	21
4.1.4	Association Value (CWT_SR_ASSOCIATIONVALUE).....	21
4.2	Active Data	23
4.3	Static Variables.....	24
5	Compatibility and Migration.....	24
5.1	Change the Client Metadata	24
5.2	Migrate Data	25
6	API Data.....	26
6.1	Base Structures	26
6.1.1	Request Base	26
6.1.2	Response Base	26
6.1.3	Managed Entity Key.....	27
6.1.4	Array of Characteristic Specifications and Values	28
6.1.5	Time Period	29
6.1.6	Array of Inventory Value.....	29
6.1.7	Array of Inventory Association Value.....	30
6.2	API Functions Parameter Structures.....	31
6.2.1	createRequest	31
6.2.2	createResponse.....	32
6.2.3	updateRequest	32
6.2.4	updateResponse.....	33
6.2.5	deleteRequest.....	33
6.2.6	deleteResponse	34
6.2.7	getByAssociationRequest	34
6.2.8	getByAssociationResponse	35
6.2.9	getByCharacteristicRequest.....	35
6.2.10	getByCharacteristicResponse	36



6.2.11	getModificationTimesRequest	36
6.2.12	getModificationTimesResponse	37
7	API Functions.....	37
8	Permission Control.....	39
9	Acronyms	39
10	Reference List	39
11	Trademarks	40
12	Disclaimer.....	40



1 Introduction

This document provides an introduction to installing, configuring, and implementing the Service Registry module.

1.1 Purpose and Scope

The purpose of this document is to provide information on how to install, configure, and implement Service Registry. To perform the tasks in this document requires that you have system administration or application development experience.

1.2 Reader's Guideline

This section describes the version syntax covered in this document and any additional, required information.

Commands that you enter on the command line appear in courier font, such as the following:

```
svnadmin dump C:\SVN\myProject > C:\backupFolder\myProject.bak
```

Document names and sections within documentation are set in italics, such as the following:

For more information on making a copy of your project metadata, see the *Velocity Studio User Guide*, under *Velocity Studio User Interface > Common Actions Outside Velocity Studio*.

Note: To navigate the documentation, an arrow appears (>), which separates each hyperlink to be clicked.

1.3 Overview

Service Registry is a module that stores information about entities, such as products, services, resources, and so on, and also stores relationship information. It is a standard module that can be implemented out-of-the-box with little or no configuration. However, this generic module can be extended to fulfill end user-specific requirements.

Full CRUD (Create, Retrieve, Update, Delete) API operations are exposed for the external systems through a full SID-based interface. All changes are versioned for easy historical data retrieval. Service Registry uses the SIDCommon library.



1.4 Software Terms of Use

The Service Registry (SR) will be known in this Terms of Use as the Product. The Product can be used by an authorized user to perform the functions outline in this document and summarized in the software features section of this document.

Legal Activity: You will not use the Product to engage in or allow others to engage in any illegal activity. You will not use the Product for any purpose that is unlawful or prohibited by these Terms of Use.

Unauthorized Customization or Reverse Engineer: You may not use the Product to obtain information necessary for you to design, develop or update unauthorized software. You may not reverse engineer, decompile, disassemble, derive the source code of, modify, or create derivative works from the Product.

Third party: You will not engage in use of the Product that will interfere with or damage the operation of the services of third parties by overburdening or disabling network resources through automated queries, excessive usage or similar conduct. You may not authorize any third party to use the Product on your behalf without a separate written agreement.

1.5 Service Registry Features

The Service Registry's main feature is the ability to list and store all services attached to a customer's billing account. However, it also contains the following features:

- Ready-to-use service inventory that provides you with the necessary tools to deploy your product quickly, while giving you the flexibility to customize your inventory to suit your business needs.
- Stores a history of customers' product, service, and resource information.
- Provides a viewable central storage repository of your customer's preferences.

2 Installation and Setup

The installation and setup consists of two different streams:

- Quick start
- Detailed start

2.1 Quick Start

These are the quick start steps for installing Service Registry:



- 1 Install the latest version of Velocity Studio by following the directions from the *Velocity Studio Installer User Guide* to install the product, initialize the database, and configure the Configuration application.
- 2 Install the latest version of the Catalog Management module by following the directions from the *Catalog Installer User Guide*.
- 3 Add the Service Registry library files to Velocity Studio (<product_installation>\modules\). Add both the required and the recursive JAR files to your Velocity Studio project:

Module	Required JAR Files	Recursive JAR Files
Service Registry	cwl_service_registry.jar catalog.jar customer.jar	address.jar api_common.jar billing.jar catalogClient.jar cwl_report.jar data_dictionary.jar notification.jar party.jar report.jar Service_registry.jar serviceOrchestrationFramework.jar SIDCommon.jar

- 4 Run the **service_registry.sql** script found in the <product_installation>\modules\service_registry\DDL folder, which calls the individual SQL scripts to create indexes and active tables.
- 5 Run the **catalog.sql** script found in the <product build>\modules\catalog\DDL folder, which calls the individual SQL scripts to create indexes and active tables.
- 6 Upgrade the database and run the associated sql file.
- 7 Run Velocity Studio.
- 8 In the System Administration application, create a default calendar.
- 9 Import code tables located in <product_installation>\modules\service_registry\code_tables:
 - cwtsr_applicationContext.xml
 - cwtsr_entityType.xml
 - cwtsr_associationType.xml

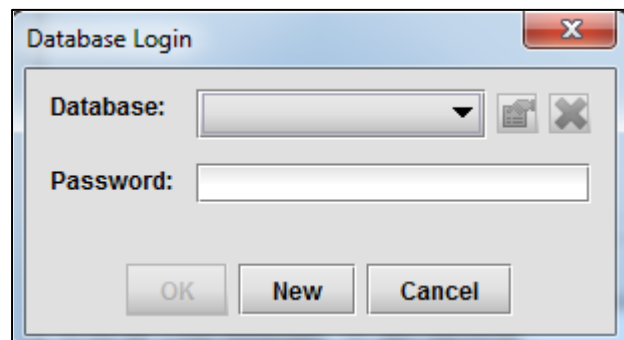
2.2 Detailed Start

This section contains more details pertaining to the installation of Service Registry described above in the Quick Start section.

- 1 Install Velocity Studio - see the *Velocity Studio Installer User Guide*.
- 2 Run the CW.sql script from the \DDL folder of your Velocity Studio folder.



- 3 Start Velocity Studio by either clicking the icon on your desktop or opening the designer\env\startDesigner.cmd file from your Velocity Studio installation folder.
- 4 Create a new project in Velocity Studio.
 - a From the menu bar, click **File > New > New Project**.
 - b From the **Select an empty directory** dialog, specify the folder where you want your new project to be saved.
- 5 Connect to your newly created schema in Velocity Studio by clicking **Database > Connect** from the menu bar.
- 6 From the Database Login dialog, click the **New** button to configure the database connection settings.



- 7 From the Connection Properties dialog, click the **New** button.
- 8 From the Driver Properties dialog, click the **Driver type** field's drop-down menu and select **Oracle thin**. Proceed to enter the **Host**, **Port**, and **Service** information, and then click the **OK** button.
- 9 The Connection Properties dialog reappears. Enter the Connection **Name** and **User**. Click the **OK** button to return to the Database Login dialog, and then click the **OK** button to connect.
- 10 Run the framework by clicking **Runtime > Run** from the menu bar.

Note: Velocity Studio only runs in Configuration mode until the Configuration tool is properly set up and the application metadata has been run.

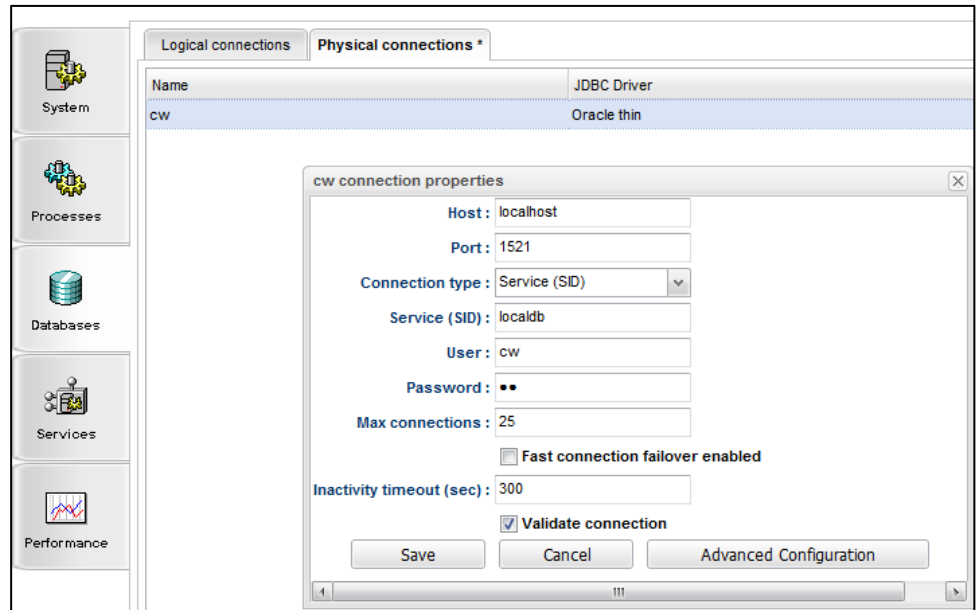
2.2.1 Configuration Application

Once you have set up the database configuration in Velocity Studio, you can run and set up the Configuration application.

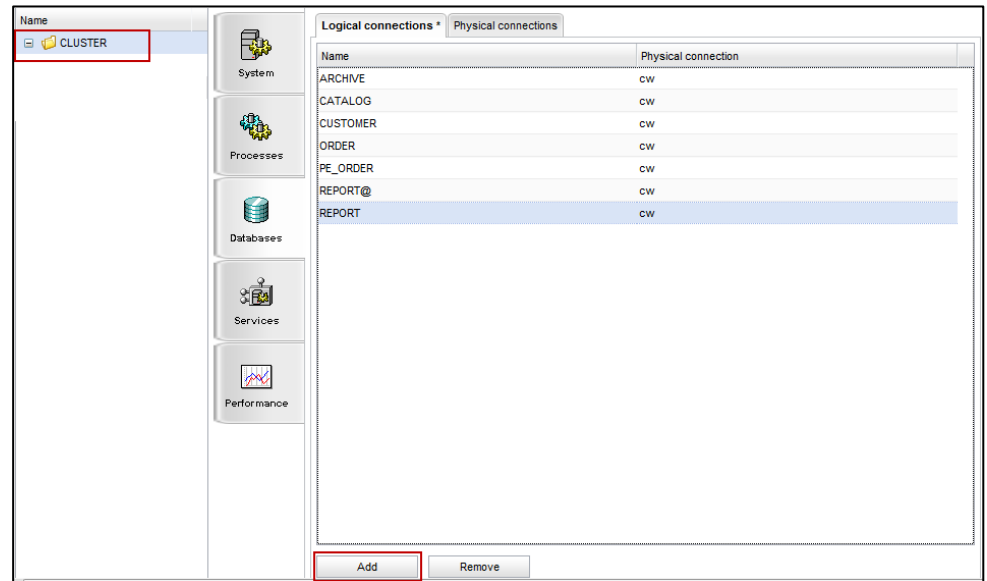
- 11 Open a Web browser and do the following:
 - a Enter the following Web address in a Web browser:
<http://localhost:8080/cwf/config>



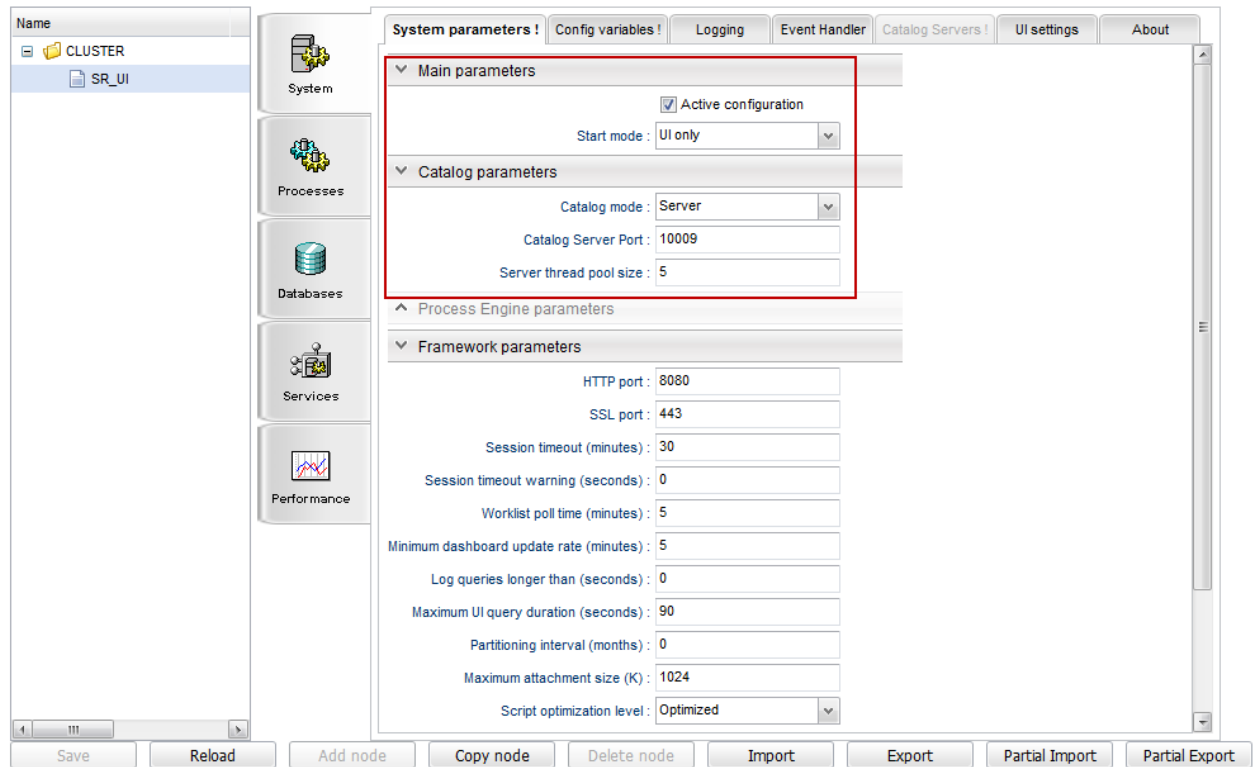
- b Log in by entering your administrative credentials (e.g. `upadmin` in both the **Username** and **Password** fields) and then click the **OK** button.
- 12 Create a new Physical connection. Navigate to the **Databases** vertical tab and then the **Physical connections** horizontal tab.
 - a Click the **Add** button to add a new physical connection. Enter the name of the physical connection.
 - b Highlight the new connection just created and click the **Edit** button to add the connection properties.



- 13 Add Logical connections.
 - a At the Cluster level, navigate to **Databases** on the vertical tab and click the **Logical** connections horizontal tab.
 - b Click the **CLUSTER** node and then Add a new CATALOG, CUSTOMER, REPORT and REPORT@ logical connection by clicking the **Add** button and entering `CATALOG`, `CUSTOMER`, `REPORT`, and `REPORT@` as your new logical connections.




- c Associate these connections to a physical connection by double-clicking the **Physical connection** column of newly added logical connection and selecting the **Physical connection** from the drop-down menu.
 - d Click the **Save** button to save changes.
- 14 Create a node-level configuration. Click **Add node** from the Cluster level to create a new node. **Enter new node name** (for example, SR_UI) and click the **OK** button.
 - 15 Configure the SR node for the Catalog parameters by selecting your **SR_UI** node and navigating to the vertical **System** tab and the associated **System Parameters**.



2.2.2 Velocity Studio

Return to Velocity Studio to continue the configuration.

- 16 Stop the runtime in Velocity Studio by clicking **Runtime > Stop** from the menu bar.
- 17 Update the Node pointer in Velocity Studio to point to the SR_UI Node created in the Configuration Tool. Navigate to **File > Preferences > Project** and in the Node ID field, type the name of the **SR_UI** node set in the Configuration Tool and click the **OK** button.
- 18 Within Velocity Studio, add the Service Registry (SR) JAR files that contain the metadata to enable SR to run.
 - a Navigate to the root metadata header. Click the **Library** tab, and then click the **Add** button (.
 - b The Service Registry JAR files are located in the `<product_installation>\modules` folder. Add all required and recursive JAR files to Velocity Studio's Library. The complete list of all required JAR files for Service Registry are as follows:

Module	Required JAR Files	Recursive JAR Files
Service Registry	catalog.jar customer.jar cwl_service_registry.jar	address.jar api_common.jar billing.jar



Module	Required JAR Files	Recursive JAR Files
		catalogClient.jar cwl_report.jar data_dictionary.jar notification.jar party.jar report.jar Service_registry.jar serviceOrchestrationFramework.jar SIDCommon.jar

- c Click **Open**. A dialog appears asking whether you want to copy this .jar file into the project template folder. Clicking the **No** button adds this folder path to your template lookup directory list.
 - d Click either the **Yes** or **No** button in the dialog. If you click the **Yes** button, the JAR files are copied locally to your <project folder>\templates folder. Otherwise, clicking the **No** button means that the JAR files are not copied locally.
 - e Once the files are added, click the **Save** button to save your project metadata, and then either reload or open the project for the library files to take effect.
- 19 Run the **service_registry.sql** script found in the <product_installation>\modules\service_registry\DDL folder which calls the individual SQL scripts to create indexes and active tables. This script calls the following other sql scripts:
- cwtsr_cleanup.sql (create cleanup procedure and schedule Oracle job)
 - cwtsr_drop.sql (drop the tables that are used to store Service Registry)
 - cwtsr_indexes.sql (create indexes)
 - cwtsr_tables.sql (create Active tables)
 - cwtsr_triggers.sql (create triggers)
 - cwtsr_up.sql (add privilege to view Service Registry's application (user interface))
- 20 Run the **catalog.sql** script found in the <product_installation>\modules\catalog\DDL folder which calls the individual SQL scripts to create indexes and active tables. This script calls the following other SQL scripts:
- code_table.sql
 - defaults.sql
 - models.sql
 - privileges.sql
 - procedures.sql
 - sof.sql (located in the \modules\sof\DDL folder)
- 21 To make the existing database schema compatible with the new build, upgrade the database by completing these steps:
- a Select **Database > Upgrade System** from the menu bar to open the Upgrade SQL dialog.



- b Specify the directory and the name of the SQL file to be generated, and then click the **Save** button to create the file.
- c Use SQLPlus to connect to the appropriate database and run the SQL file.

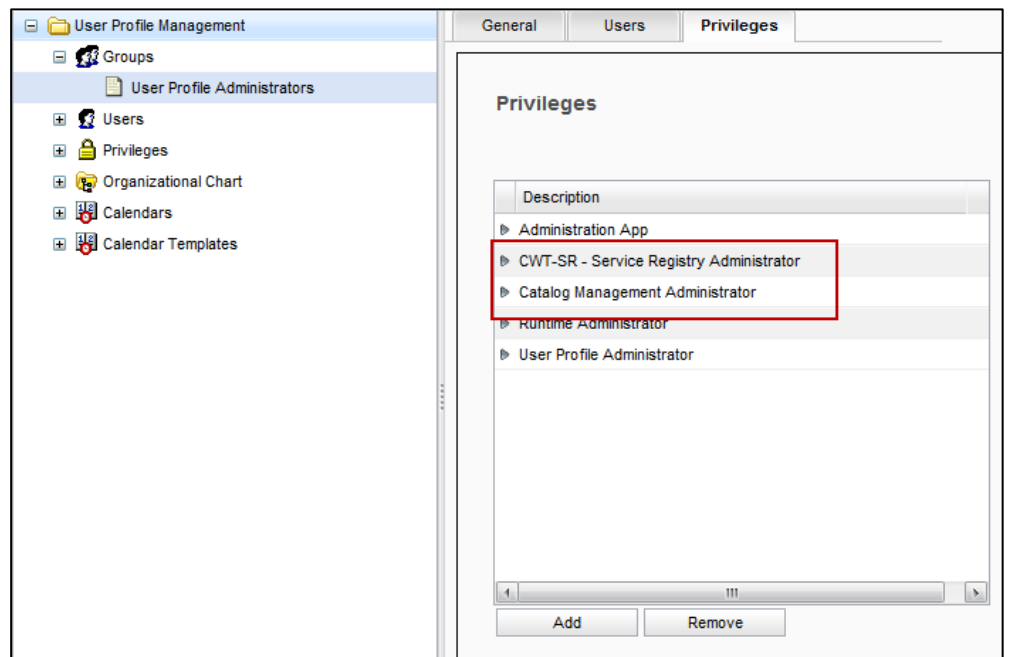
Note: If there are no system upgrades, a dialog box appears, indicating that no upgrades are available.

- 22 Within Velocity Studio, restart the runtime by clicking **Runtime > Start** from the menu bar. Proceed to connect the database by clicking **Database > Connect** from the menu bar.

2.2.3 System Administration Application

Once you have successfully configured all settings in Velocity Studio, you can then import code tables, create a default calendar and create the appropriate privileges to access Service Registry and Catalog Management.

- 23 Access the Administration application by doing the following:
 - a Enter the following Web address in a Web browser:
http://localhost:8080/cwf/admin
 - b Enter your user credentials (example, upadmin) in the **Username** and **Password** fields and click the **OK** button.
- 24 Assign the Service Registry and Catalog privileges.
 - a Navigate to **User Profile Manager** on the menu bar and click **Groups**.
 - b Highlight the appropriate group and navigate to the **Privileges** tab.
 - c Click the Add button to add the Catalog Management Administrator and CWT-SR – Service Registry Administrator privilege.






- 25 Create a default calendar. Create a default calendar by following the steps provided:
- Navigate to the **User Profile Manager** menu and click **Calendars**.
 - Click the button **New Calendar Type** to add a new Calendar.

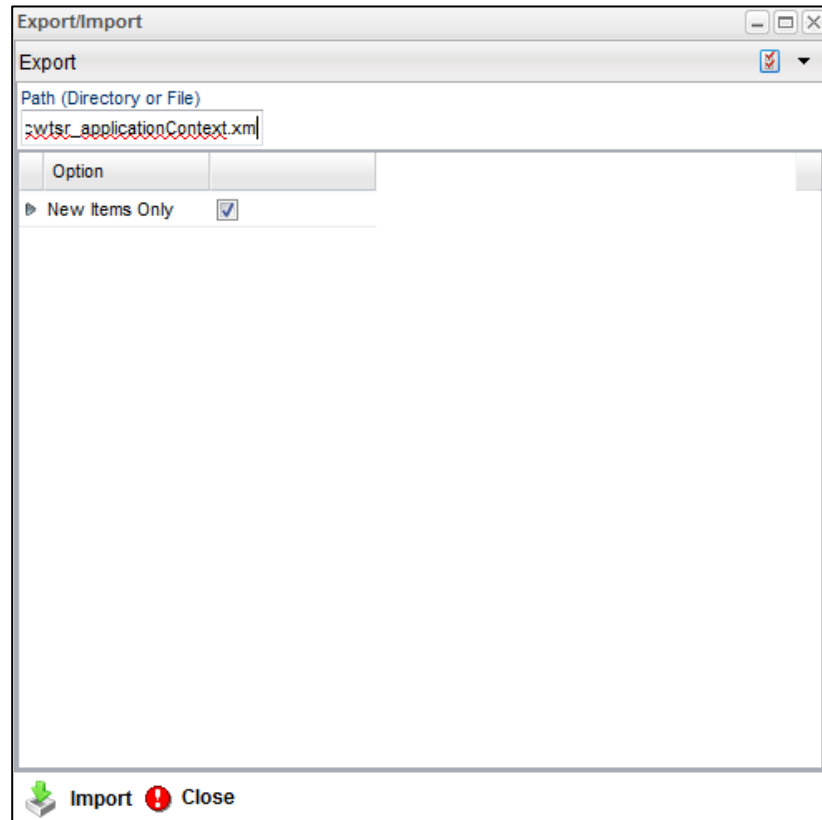
The screenshot shows the 'User Profile Management' application. In the left sidebar, the 'Calendars' menu item is selected, showing a tree view with 'Canadian Working Calendar' and 'Calendar Templates' (containing 'Canada' and 'USA'). The main content area displays the 'Calendar' configuration form. The form includes fields for 'Name' (Working Calendar), 'Label' (Canadian Working Calendar), 'Calendar Template Reference' (Canada), and a 'Work Hours' section with 'First Day' (Sunday), 'From Hour' (08), 'From Minute' (00), 'To Hour' (17), and 'To Minute' (00).

- Enter the appropriate working calendar information and click the **Save** button. Note that the Administrative Application has two canned Calendar Template Reference (US and Canada) but others can be created.
- Click the **Set As Default Calendar** button to make this the default calendar.

2.3 Add Code Tables

The installation of the Service Registry product results in a blank (no data) database. This section describes the process for importing the code tables either through the Catalog Management application or the Administration application.

- 26 Start the process of importing using one of the following methods:
- In the Administration application, click the **Tools** menu, and then click **Code Tables** from the menu. In the page that opens, click the **Import** button at the bottom.
 - In the Catalog Management application, select **Configuration** menu () and click on **Code Tables**. In the page that opens, click the **Import** button at the bottom.
- 27 Import the following files by entering the path (`<product_installation>\modules\service_registry\code_tables`) of the each file and click the **Import** button:
- `cwtsr_applicationContext.xml`
 - `cwtsr_entityType.xml`
 - `cwtsr_associationType.xml`





3 Logical Model

The SUF graphical user interface (GUI) contains the following elements: The Service Registry logical model consists of four main tables:

- Entity
- Entity Values
- Association Values

Data Element	Description	Key
Entity	Stores product, service, term data, and more.	<ul style="list-style-type: none"> • Application Context • Entity Type • DN • Valid From Date
Association	Stores associations between Service Registry entities as well as associations with entities from external systems	<ul style="list-style-type: none"> • Application Context • Entity Type • DN • Valid From Date • Entity Key • Association Entity Key
Association Values	Stores association-specific characteristics such as name-value pairs.	<ul style="list-style-type: none"> • Association Application Context • Association Type • Association DN • Characteristic • Valid From Date

4 Data Model

Service Registry's data model consists of the following information:

- [Main table definitions](#)
- [Active data](#)

4.1 Main Table Definitions

Service Registry contains three main tables that stores entity information and relationships:

- [Entity \(CWT_SR_ENTITY\)](#)
- [Association \(CWT_SR_ASSOCIATION\)](#)
- [Association Value \(CWT_SR_ASSOCIATIONVALUE\)](#)



4.1.1

Entity (CWT_SR_ENTITY)

The CWT_SR_ENTITY table stores the following information:

Variable	Type	Column	Description
Application Context	varchar2(32)	APPLICATIONCONTEXT	Composed key, identifying the service registry entity.
Entity Type	varchar2(32)	ENTITYTYPE	
Distinguished Name	varchar2(128)	DN	
Valid From Date	date	VALIDFROMDATE	Valid from timestamp for the record. Along with the above key fields – uniquely identifies the entity and its version.
Valid To Date	date	VALIDTODATE	Valid to timestamp for the record. Can be null.
System created by	varchar2(64)	CREATEDBY	ID of the user created the record (system field)
System created timestamp	date	CREATEDDATE	Timestamp the record was created in DB (system field).
System updated by	varchar2(64)	UPDATEDBY	ID of the user updated the record (system field).
System last updated timestamp	date	UPDATEDDATE	Timestamp the record was last updated in DB (system field).
State	varchar2(16)	STATE	State of the record (reserved for future use)
State Date	date	STATEDATE	Timestamp the state was changed (reserved for future use)
Hierarchical Naming	boolean	HIERARCHICALNAMING	Is hierarchical naming used (reserved for future use)
Auto Named	boolean	AUTONAMED	Is auto-naming of the record used (reserved for future use)
Data	nclob	DATA	Data field contains item specifications (that is, all static



Variable	Type	Column	Description
			variables and dynamic variables from basket objects) in JSON format
Search Text	varchar2(4000)	SEARCHTEXT	Special Oracle text field with an index that is built for a specified period (for example, every five minutes)

Notes:

- Only one table, CWT_SR_ENTITY, stores both entity and entity values.
- Only entities that are real instances (that is, products, offers, and services) are stored as separate rows.
- All entity values and non-instance children are stored as the entity data's JSON part.
- Additional entity table columns are automatically mapped from entity values. The mapping logic can be customized depending on the implementation.
- The SEARCHTEXT column's content is completely customizable.
- The default query is also completely customizable depending on the implementation.

4.1.1.1

SEARCHTEXT Column

The CWT_SR_ENTITY table's SEARCHTEXT column contains variable names (that is, attribute names) and values to allow for searching based on entity attributes.

Note: The **Date** value is not searchable.

This column's content is not customizable. Saving the API publishes an event to retrieve the searchable list.

The searchable list is user-defined and contains all attribute or variable names that must be set in the search text. You also need to implement an event handler return for that list.

The search format is also not customizable. The default format is as follows:

```
<ATTRNAME> <ATTRVALUE> <ATTRNAME> <ATTRVALUE>
```

The following is an example:

```
Speed 25 Technology ADSL
```

Note: You must grant the following database privilege to perform the Oracle text search:



GRANT CREATE JOB TO USER

One event is for the search text column. When saving a basket item, the `getEntitySearchList` event is published to get the search text key (that is, attribute name).

4.1.1.2 Extensions Fields in the Entity Table

The entity base implementation contains only required properties and associations. Any other properties that make sense to keep in the table can be added as extension field.

All dynamic variables that have the same name in persistent mode are mapped. The default implementation is mapping dynamic variables to extension variables by variable name.

To implement extra mapping logic, use the `mapEntityExtensionLeaf` event.

4.1.1.3 Extension Tables

You can extend the Entity table with an additional table for specific item types. It is recommended that you use extension tables to provide additional search attributes only, since all entity data is saved as a BLOB item.

The saving API publishes an event that determines the model type to use for a given basket item type. You need to implement the event handler (input consists of both the basket object and basket item object) to decide which model to use for which item.

The following is information about the following events:

- Use the `getEntityExtensionDocument` event to populate the extension document name for each entity item
- Use the `mapEntityExtensionDocument` event to map the entity item to the extension document

The Service Registry has default mapping logic to map an Entity item to an extension document by variable name if the `mapEntityExtensionDocument` event is not implemented.

Note: The Service Registry implementation does not handle the delete operation for an extension table. You must create a foreign key with either a cascade delete or a trigger to handle an extension document delete operation when an item has been deleted from the Entity table.

4.1.2 Association (CWT_SR_ASSOCIATION)

The `CWT_SR_ASSOCIATION` table stores the following information:



Variable	Type	Column	Description
Application Context	string(128)	APPLICATIONCONTEXT	Composed key, identifying the service registry association.
Association Type	string(32)	ASSOCIATIONTYPE	
Distinguished Name	string(128)	DN	
Valid From Date	dateTime	VALIDFROMDATE	Valid from timestamp for the record. Along with the above key fields – uniquely identifies the association and its version.
Valid To Date	dateTime	VALIDTODATE	Valid to timestamp for the record. Can be null.
System created by	string(64)	CREATEDBY	ID of the user created the record (system field)
System created timestamp	dateTime	CREATEDDATE	Timestamp the record was created in DB (system field).
System updated by	string(64)	UPDATEDBY	ID us the user updated the record (system field).
System last updated timestamp	dateTime	UPDATEDDATE	Timestamp the record was last updated in DB (system field).
State	string(16)	STATE	State of the record (reserved for future use)
State Date	dateTime	STATEDATE	Timestamp the state was changed (reserved for future use)
Self	boolean	SELF	Is pointing to self (reserved for future use)
Entity Application Context	string(128)	ENTITYAPPLICATIONCONTEXT	Composed key to identify the entity used in this association.
Entity Type	string(32)	ENTITYTYPE	
Entity Distinguished Name	string(128)	ENTITYDN	
Is External Entity	boolean	ISEXTERNALENTITY	Is the above entity external (not stored in this SR)



Variable	Type	Column	Description
Associated Entity Application Context	string(128)	ASSOCENTITY APPLICATIONCONTEXT	Composed key to identify the associated entity.
Associated Entity Type	string(32)	ASSOCENTITYTYPE	
Associated Entity Distinguished Name	string(128)	ASSOCENTITYYDN	
Is External Associated Entity	boolean	ISEXTERNALASSOC ENTITY	Is the above entity external (not stored in this SR)
Data	nclob	DATA	Data field contains item specifications (that is, all static variables and dynamic variables from basket objects) in JSON format
Search Text	varchar2(4000)	SEARCHTEXT	Special Oracle text field with an index that is built for a specified period (for example, every five minutes)

Notes:

- The SEARCHTEXT and DATA fields allow saving attribute values.
- The getAssociationSearchList event allows you to get the search list for association attribute values.

4.1.3 Extension Fields in the Entity Table

The following events help extend fields in an Entity table:

- Use the mapAssociationExtensionLeaf event to map a variable value to an extension document variable
- Use the getAssociationExtensionDocument event to populate the extension document name for each association item
- Use the mapAssociationExtensionDocument event to map an entity association item to an extension document

4.1.4 Association Value (CWT_SR_ASSOCIATIONVALUE)



The CWT_SR_ASSOCIATIONVALUE table stores the following information:

Variable	Type	Column	Description
Association Application Context	string(128)	ASSOCAPPLICATION CONTEXT	Association composed key.
Association Type	string(32)	ASSOCTYPE	
Association Distinguished Name	string(128)	ASSOCDN	
Valid From Date	dateTime	VALIDFROMDATE	Association Value valid from timestamp. Along with the above key fields – uniquely identifies the entity value and its version
Valid To Date	dateTime	VALIDTODATE	Association Value valid to timestamp (can be null)
System created by	string(64)	CREATEDBY	ID of the user created the record (system field)
System created timestamp	dateTime	CREATEDDATE	Timestamp the record was created in DB (system field).
System updated by	string(64)	UPDATEDBY	ID us the user updated the record (system field).
System last updated timestamp	dateTime	UPDATEDDATE	Timestamp the record was last updated in DB (system field).
Characteristic	string(32)	CHARACTERISTIC	Name of the characteristic
Value Type	string(16)	VALUETYPE	Value type of the characteristic (string, number, dateTime)
Value – DateTime	dateTime	VALUEDATETIME	DateTime value of the characteristic
Value - Number	decimal(14,4)	VALUENUMBER	Number value of the characteristic



Variable	Type	Column	Description
Value - String	string(256)	VALUESTRING	String value of the characteristic. <i>Note: number and dateTime type characteristics will be converted to string and stored in this column as well (useful for display purposes)</i>

4.2 Active Data

Service Registry is expected to contain an enormous amount of data, as it keeps all customer historical data. To improve search performance, Service Registry uses separate tables for active and inactive customer services. The active data is stored in Active tables and the archive data is stored in the Service Registry's logical model tables.

The active tables are as follows:

Active Table	Related Service Registry Tables
CWT_SR_ENTITYACT	CWT_SR_ENTITY
CWT_SR_ASSOCIATIONACT	CWT_SR_ASSOCIATION
CWT_SR_ASSOCIATIONVALUEACT	CWT_SR_ASSOCIATIONVALUE

The Service Registry module automatically redirects the request to query the Active tables for the customer's current data, or to query the service registry tables for historical data. Active tables are not burdened with history and, as a result, performance remains stable.

The system populates these tables using **on insert** and **on update** triggers. These triggers update the Service Registry tables. All records that are currently active are populated in the Active tables. Once a record has expired (that is, the **ValidToDate** field is in the past), the record is deleted from the Active tables. A nightly batch process is then run to remove old records from the table.

The following is a brief description of the trigger logic. For more information, see the **cwtsr_triggers.sql** file.

On Insert trigger:

- 1 Search for an existing record in the CWT_SR_Entity table (based on the key fields ApplicationContext, EntityType, and DN).
- 2 If the record is found, set the **ValidToDate** field of the previous record version to match the **ValidFromDate** of new record.
- 3 Insert the record in the Active table.
- 4 Update or remove the old (expired) record from the Active table.

**On Update trigger:**

Update the **ValidToDate** field or delete the record from the Active table.

4.3 Static Variables

Service Registry contains a number of commonly stored attributes as static variables that can be used by other applications, such as Order Negotiations.

The following is a list of static variables in Service Registry:

Static Variable	Type
accountId	Varchar(32)
catalogCode	Varchar(32)
catalogType	Varchar(32)
category	Varchar(32)
customerId	Varchar(32)
locationId	Varchar(32)
orderId	Varchar(32)
serviceDate	Date
siteId	Varchar(32)

5 Compatibility and Migration

The Service Registry model is compatible with the previous Service Registry implementation at the business logic level only, but not at the data level (that is, database tables, and SQL and metadata objects). The API is backward-compatible.

The migration procedure consists of the following steps:

- [Change the client metadata](#)
- [Migrate the data](#)

5.1 Change the Client Metadata

Changing the client metadata consists of the following changes:

- Build individual model extensions based on the client business model
- Override finders to build metadata-specific search queries
- Change client metadata to use new signatures of API methods (not an issue since the functionality is only extended) or using an adapter module



5.2 Migrate Data

Migrating data consists of developing a special migration module for data migration, which may be an individual module for each client metadata. The module then reads the existing data, maps it to the new model, and then saves it to the new fields.

Migrating most of the data is straightforward. However, to take advantage of the new model, it is recommended that data migration be done as an individual module.

Note: It may be complicated to either migrate or manually change the client metadata if it uses or extends a metadata object directly. This complication occurs especially when Service Registry-specific objects and scripts are either extended or overridden.



6 API Data

Service Registry's API data can be classified by the following categories:

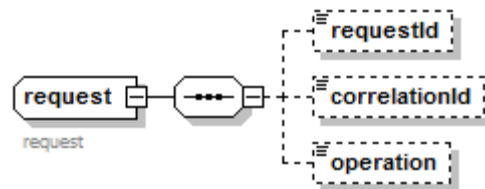
- [Base structures](#)
- [API functions parameter structures](#)

6.1 Base Structures

The following is the request-response data structure description. Although the API functions mostly use same input and output data structure modules, every API function has its own requirements for which fields should be populated, which are mandatory, optional, or ignored. This will be described at API function level.

6.1.1 Request Base

All request data structures extend the base request:



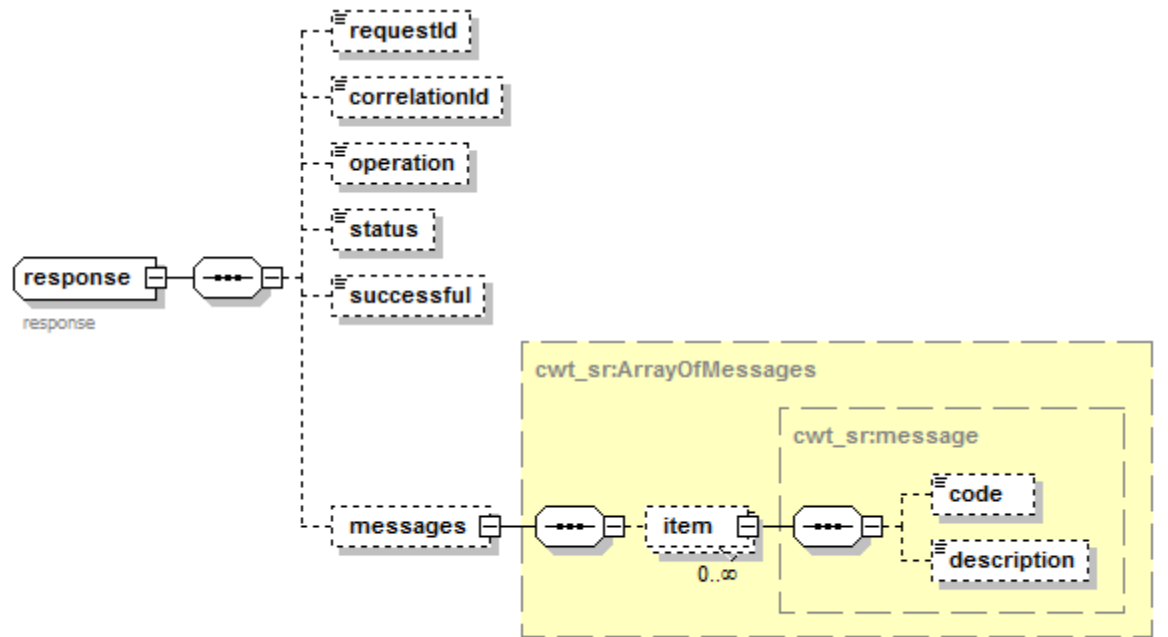
Generated by XMLSpy

www.altova.com

requestId	– optional, will be echoed back with the response
correlationId	– optional, will be echoed back with the response
operation	- not used

6.1.2 Response Base

All response data structures extend the base response:



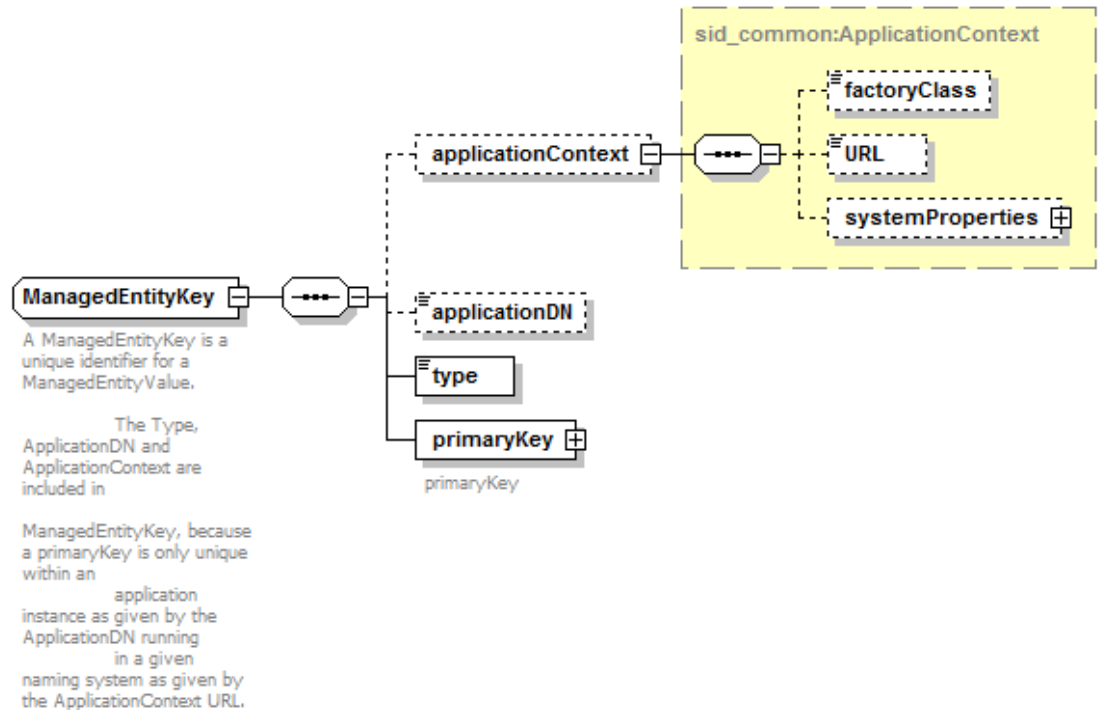
Generated by XMLSpy

www.altova.com

requestId	– [optional] echoed back from the request
correlationId	– [optional] echoed back from the request
messages	– [optional] array of error messages
• code	- [mandatory] error code
• description	- [optional] localized error description
(all others)	– not used

6.1.3 Managed Entity Key

Entity or Association identifier.



Generated by XMLSpy

www.altova.com

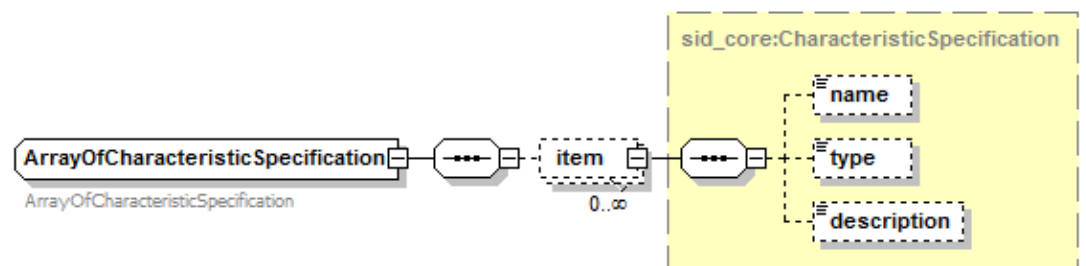
applicationContext -> URL
applicationDN
type
(all others)

- [mandatory] part of item key
- [mandatory] part of item key
- [mandatory] part of item key
- not used

6.1.4

Array of Characteristic Specifications and Values

These are building blocks for request/responses related to Entity and Association Values.

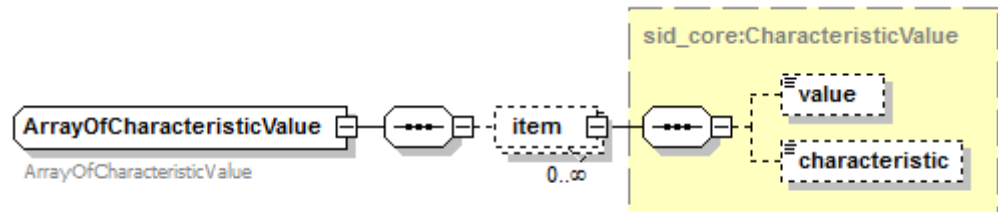


Generated by XMLSpy

www.altova.com

name
type
(all others)

- [mandatory] value name
- [mandatory] value type
- not used



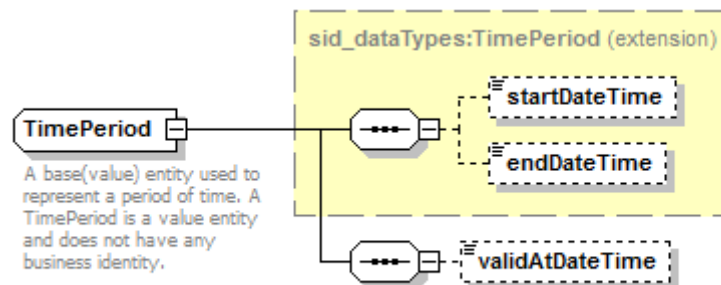
Generated by XMLSpy

www.altova.com

- | | |
|----------------|---|
| characteristic | - [mandatory] value name
(same as characterizedBy->name) |
| value | - [mandatory] value |

6.1.5 Time Period

To describe Start, End or exact time point for request/response.



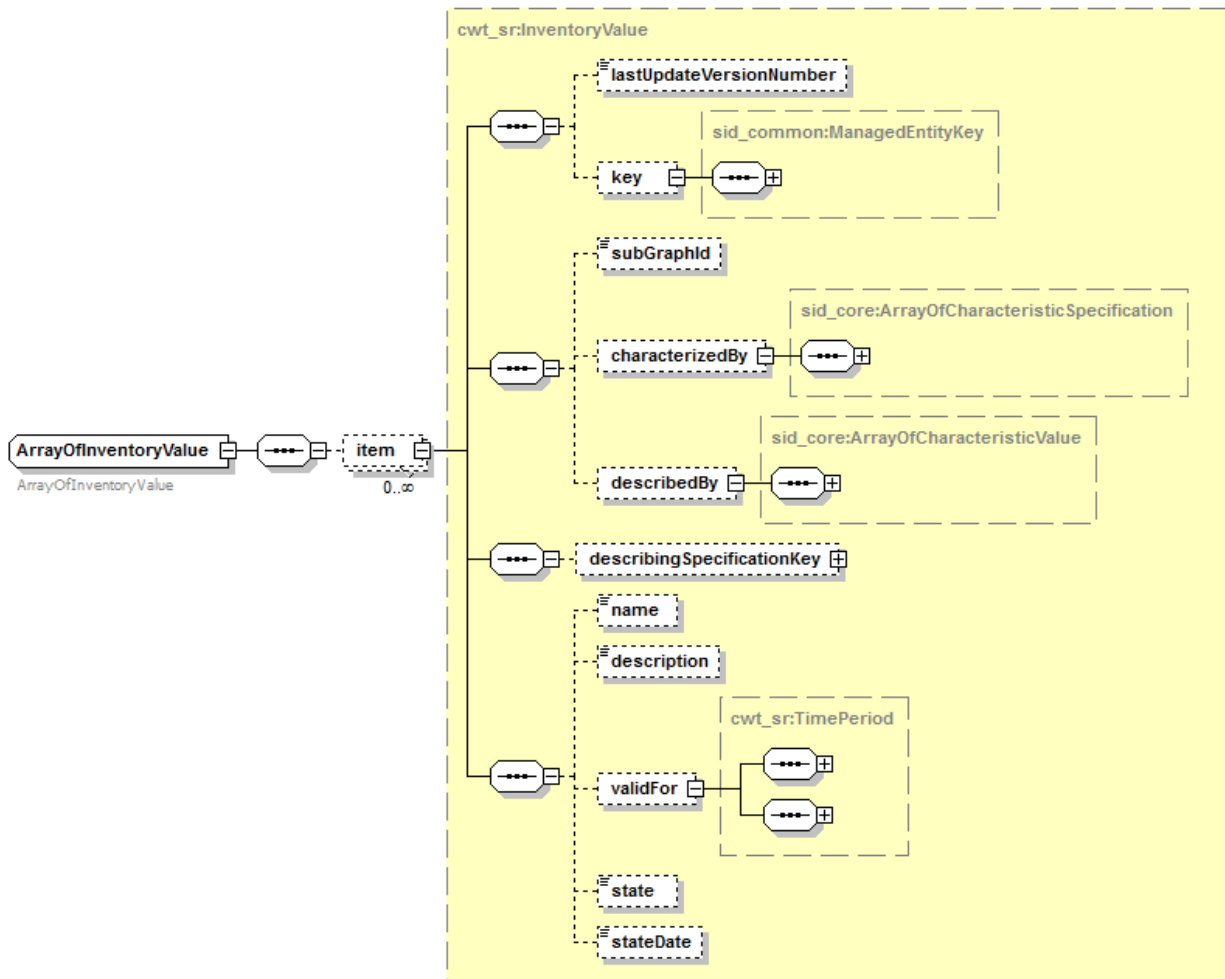
Generated by XMLSpy

www.altova.com

- | | |
|-----------------|--|
| startDateTime | - [per API] item start date |
| endDateTime | - [per API] item end date |
| validAtDateTime | - [per API] used in enquiry requests to specify the exact request time |

6.1.6 Array of Inventory Value

Base building block for the request/response. Contains data stored as Entity and EntityValues.



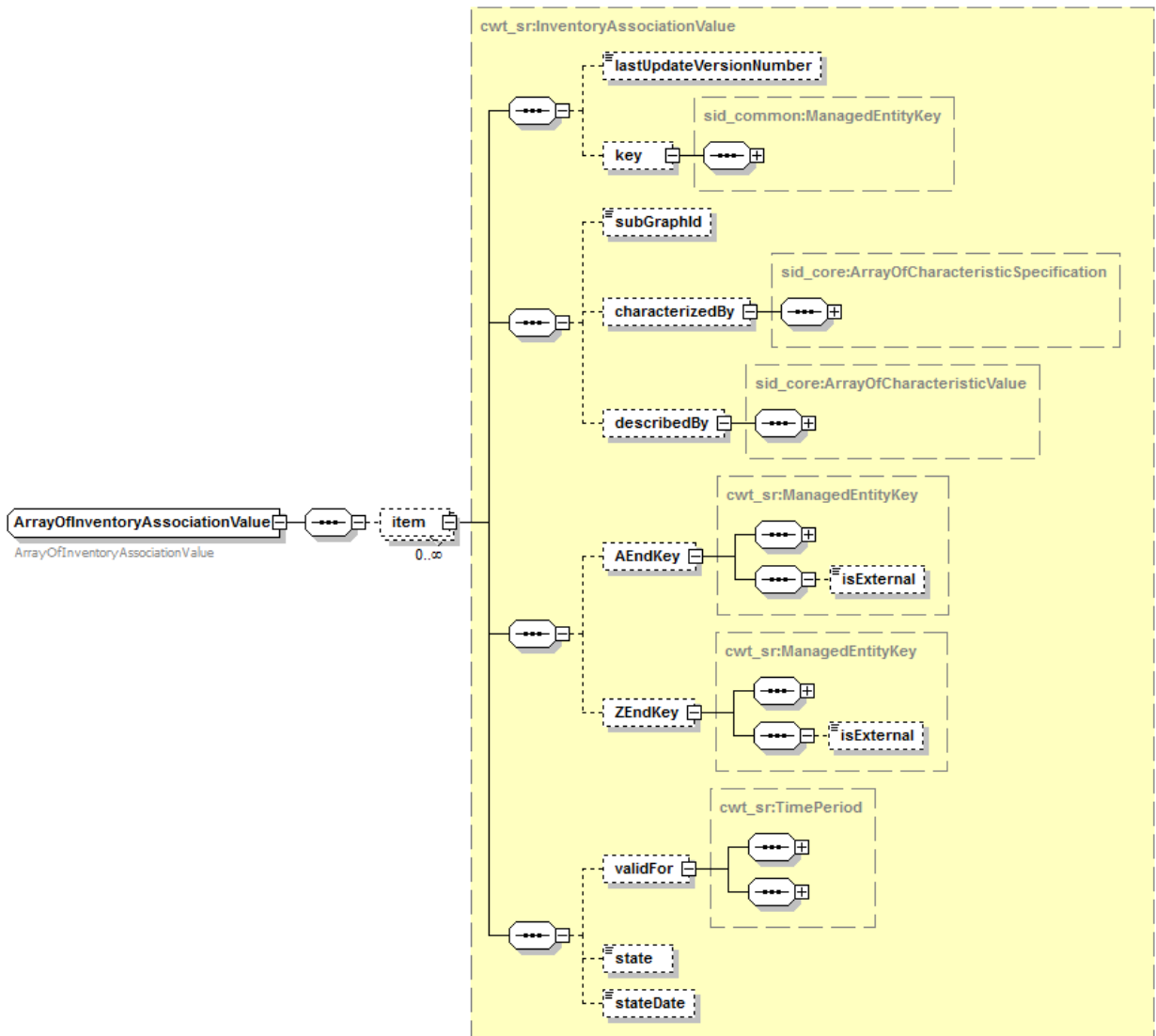
Generated by XMLSpy

www.altova.com

key	- [per API] Managed Entity Key
characterizedBy	- [optional] Array of Characteristic Specifications
describedBy	- [optional] Array of Characteristic values
validFor	- [optional] Time Period
(all others)	- not used

6.1.7 Array of Inventory Association Value

Base building block for the request/response. Contains association between SR Entities or between Entity and external entity.



Generated by XMLSpy

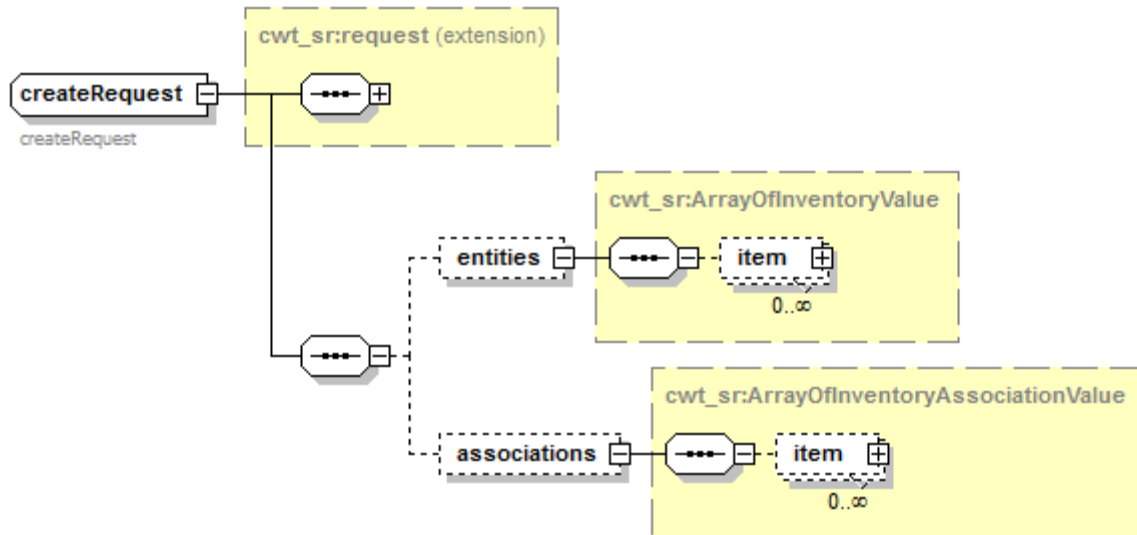
www.altova.com

key	- [per API] Managed Entity Key
characterizedBy	- [optional] Array of Characteristic Specifications
describedBy	- [optional] Array of Characteristic values
AEndKey	- [per API] Managed Entity Key
ZEndKey	- [per API] Managed Entity Key
validFor	- [optional] Time Period
(all others)	- not used

6.2 API Functions Parameter Structures

6.2.1 createRequest

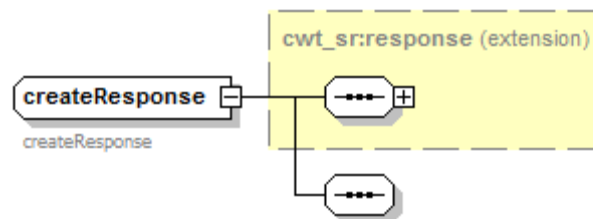
Contains data to create Service Registry items (entities, associations and entity/association values).



Generated by XMLSpy www.altova.com

entities - [mandatory] array of inventory values
 associations - [optional] array of association values

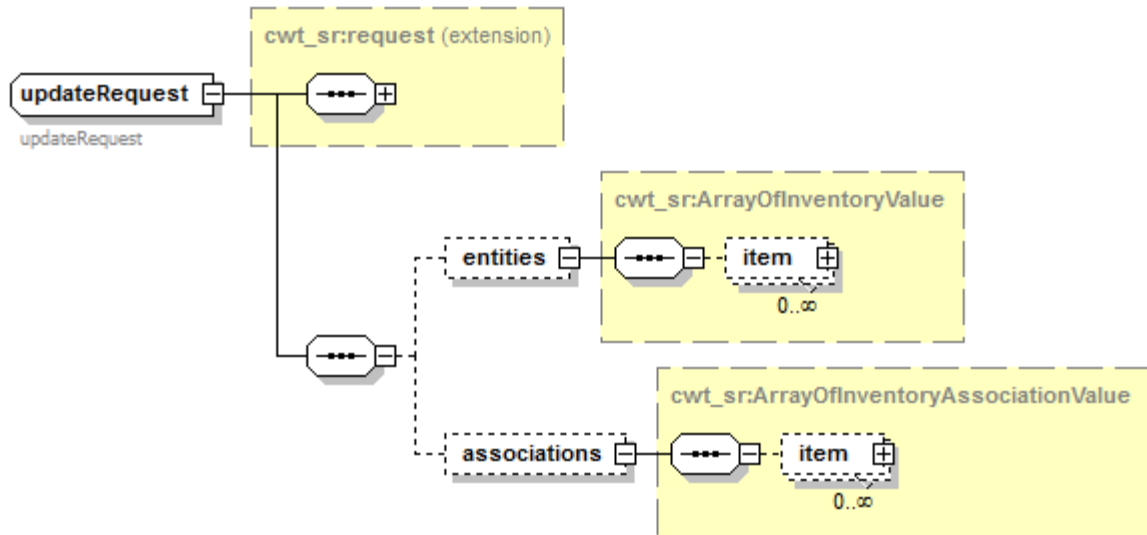
6.2.2 createResponse



Generated by XMLSpy www.altova.com

6.2.3 updateRequest

Contains data to update Service Registry items (entities, associations and entity/association values).

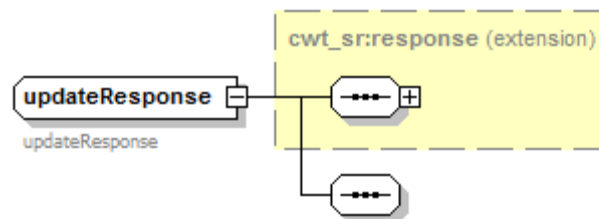


Generated by XMLSpy

www.altova.com

entities - [optional] array of inventory values
 associations - [optional] array of association values

6.2.4 updateResponse

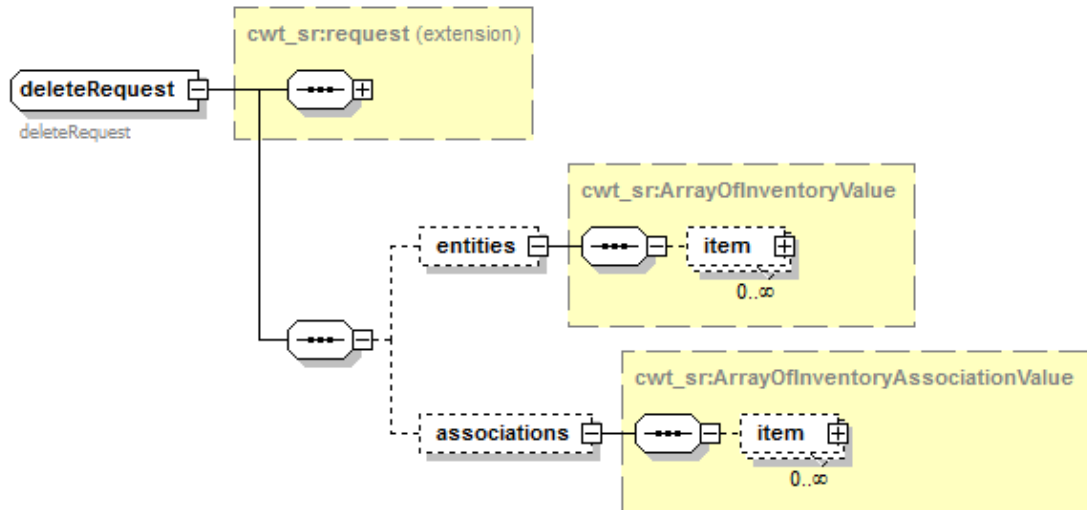


Generated by XMLSpy

www.altova.com

6.2.5 deleteRequest

Contains data to delete Service Registry items (entities, associations and entity/association values).

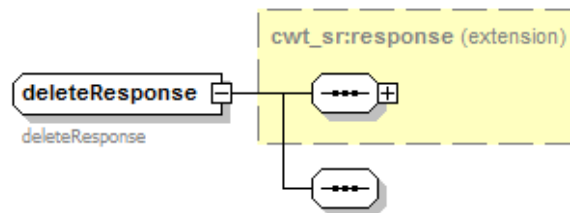


Generated by XMLSpy www.altova.com

entities - [optional] array of inventory values

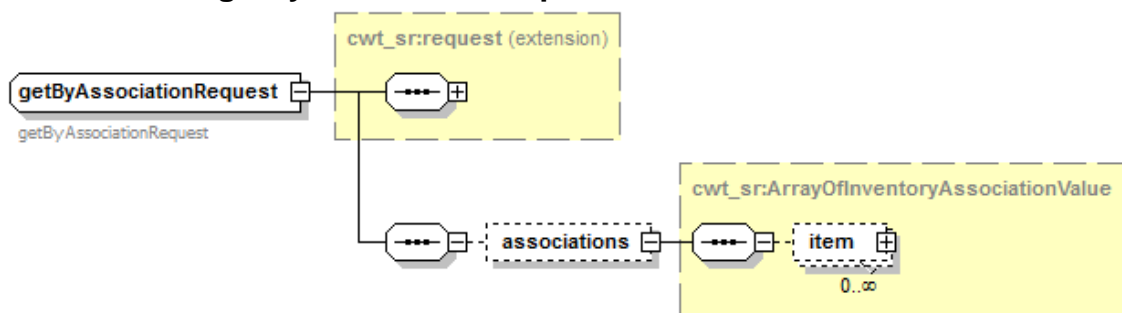
associations - [optional] array of association values

6.2.6 deleteResponse



Generated by XMLSpy www.altova.com

6.2.7 getByAssociationRequest

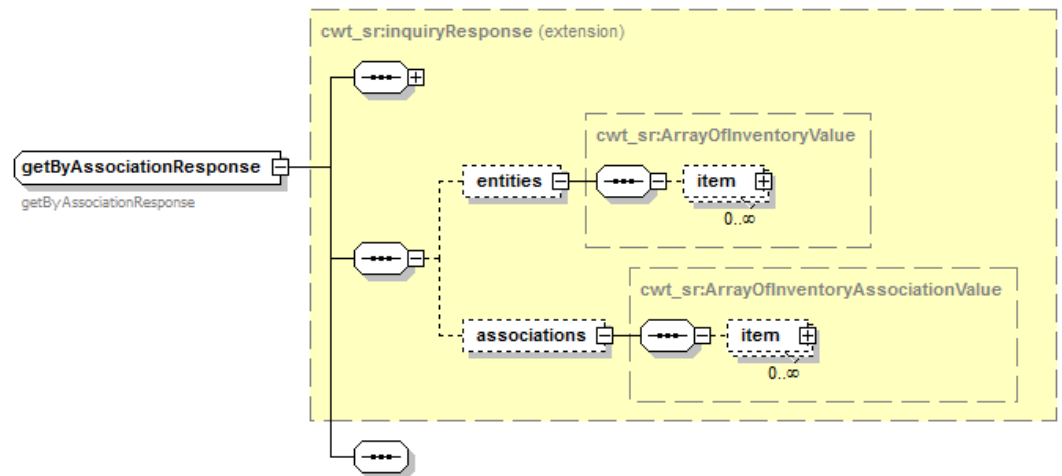


Generated by XMLSpy www.altova.com

associations - [mandatory] array of association values



6.2.8 getByAssociationResponse



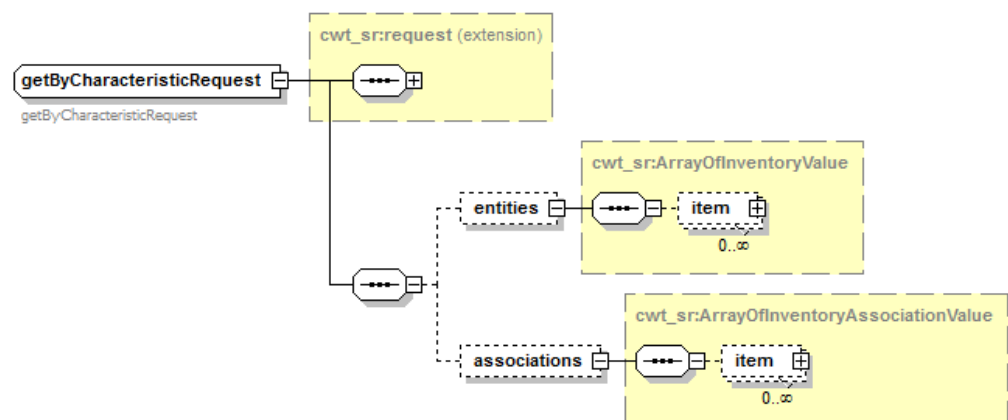
entities
associations

- [optional]
- [optional]

Generated by XMLSpy
array of inventory values
array of association values

www.altova.com

6.2.9 getByCharacteristicRequest



entities

- characterizedBy - [mandatory] One Characteristic Specifications
- describedBy - [mandatory] One Characteristic values
- validFor - [optional] Time Period
- validAtDateTime - [optional] when characteristic was active
- (all others) - not used

associations

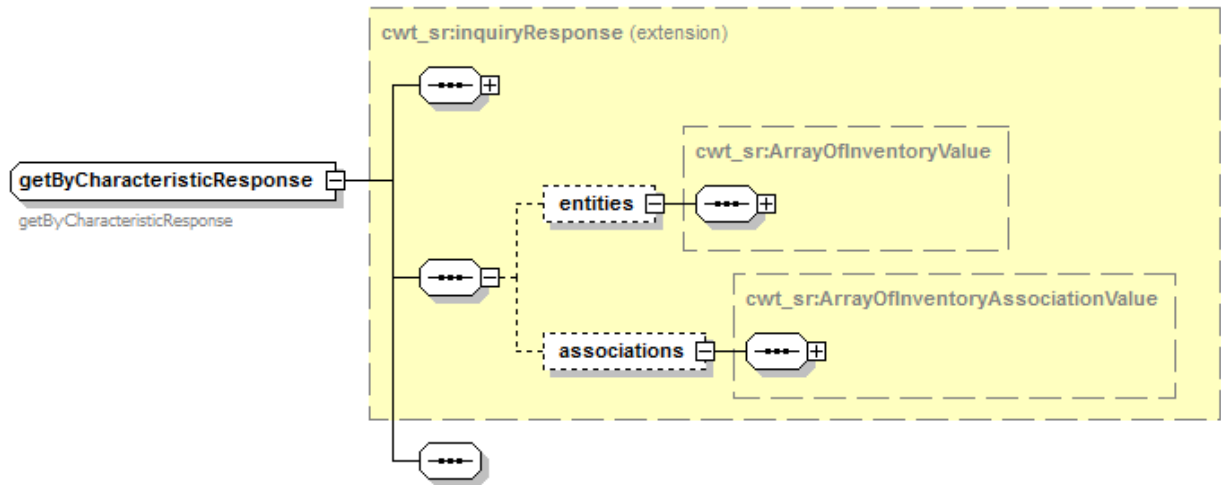
- ZEndkey - [mandatory] association value
- (all others) - [mandatory] association entity key
- (all others) - not used

Generated by XMLSpy

www.altova.com



6.2.10 getByCharacteristicResponse

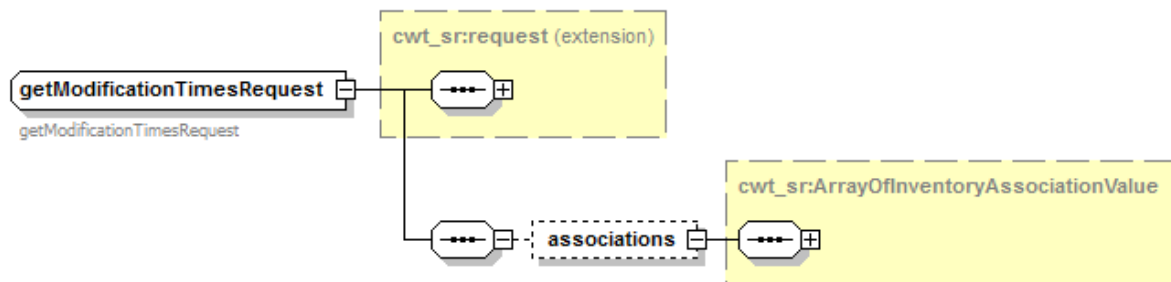


Generated by XMLSpy

www.altova.com

- entities - [optional] array of entities with the requested characteristic
 associations - [not used]

6.2.11 getModificationTimesRequest



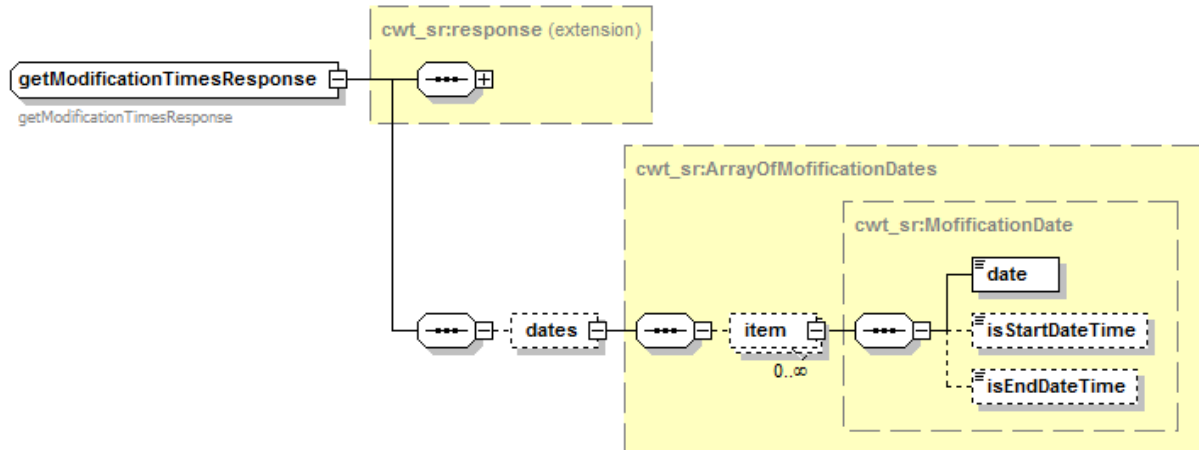
Generated by XMLSpy

www.altova.com

- associations - [mandatory] Exactly one association value
- ZEndkey - [mandatory] Association entity key
 - validFor - [optional] Time Period
 - startDateTime - [optional] Start date
 - endDateTime - [optional] End date
 - (all others) - not used



6.2.12 getModificationTimesResponse



Generated by XMLSpy

www.altova.com

dates - [optional] Modification dates

- date - [mandatory] modification date
- isStartDateTime - [optional] is modification time
'startDateTime'.
- isEndDateTime - [optional] is modification time
'endDateTime'.
- (one of isValidFrom and isValidTo should be populated)

7 API Functions

The following functions will be available through direct call to the JavaScript or through SOAP interface and are currently supported APIs:

API	Description	Request	Return
<i>createService (request)</i>	Create entities/associations from the request. If entity/ association with this key does exist – will return error message.	createRequest data structure	createResponse data structure
<i>updateService (request)</i>	Update existing entities/associations from the request. If entity/association with this key does not exist – will return error message..	updateRequest data structure	updateResponse data structure
<i>deleteService (request)</i>	Delete existing entities/associations from the request.	deleteRequest data structure	deleteResponse data structure



API	Description	Request	Return
	The items are not physically deleted, but rather marked as inactive (end-dated).		
<i>getServiceByAssociation(request)</i>	Gets all items linked via association's end key (example: all items linked to an account or service address). Call is recursive, so it will return items with 2 nd , 3 rd , etc. level of association.	getByAssociationRequest data structure	getByAssociationResponse data structure
<i>getServiceByCharacteristic(request)</i>	Get all items with the specified characteristic. Call is not recursive, only 1-st level entities associated with the request ZEndKey will be returned.	getByCharacteristicRequest data structure	getByCharacteristicResponse data structure
<i>getServiceModificationTimes(request)</i>	Get all modification times (Start Date / End Date) for any record in Service Registry – Entity, Associations, Entity/Association Values for the given period.	getModificationTimesRequest data structure	getModificationTimesResponse data structure



8 Permission Control

Application modules use privileges to control users/user group access to different forms and functions. Each application module (like Order Negotiations, Order Analytics, Customer Information Management and Service Registry), contain module specific permissions definitions that enables the system administrator the ability to assign module specific privileges to user roles. It is also possible for a system administrator to create a unique privilege and assign the privilege to a role through the Administration application and the User Profile options. This section lists the pre-defined privileges for Service Registry.

Privilege ID	Privilege Name	Functions
cwtSRApplication	CWT-SR – Service Registry Administrator	Allow full access to Service Registry application
cwpc_admin	Catalog Management Administration	Allow full access to Catalog Management application

9 Acronyms

GUI – Graphical User Interface
SQL – Structures Query Language

10 Reference List

The following is a list of documentation for reference:

- *Catalog Installer User Guide*
- *System Administration User Guide*
- *System Configuration User Guide*
- *Velocity Studio Installer User Guide*
- *Velocity Studio User Guide*



11 Trademarks

Ericsson, the Ericsson logo and the Globemark are trademarks of Ericsson.

Ericsson is a recognized leader in delivering communications capabilities that enhance the human experience, ignite and power global commerce, and secure and protect the world's most critical information. Serving both service provider and enterprise Customers, Ericsson delivers innovative technology solutions encompassing end-to-end broadband, Voice over IP, multimedia services and applications, and wireless broadband designed to help people solve the world's greatest challenges. Ericsson does business in more than 150 countries. For more information, visit Ericsson on the Web at www.Ericsson.com.

12 Disclaimer

This document may contain statements about a number of possible benefits that Ericsson believes may be achieved by working with Ericsson. These might include such things as improved productivity, benefits to end users or cost savings. Obviously, these can only be estimates. Gains might be qualitative and hard to assess or dependent on factors beyond Ericsson's control. Any proposed savings are speculative and may not reflect actual value saved. Statements about future market or industry developments are also speculative.

Statements regarding performance, functionality, or capacity are based on standard operating assumptions, do not constitute warranties as to fitness for a particular purpose, and are subject to change without notice.

This document contains Ericsson's proprietary, confidential information and may not be transmitted, reproduced, disclosed, or used otherwise in whole or in part without the express written authorization of Ericsson.