Ericsson Order Care

Realize Higher Consistency for Faster Time-to-Revenue

Order Negotiations Configuration Guide

5/127 01-CSH 109 420/1 Uen Rev A

Contents

1 Introduction

This document provides information on configuring the Order Negotiations (ON) module.

1.1 Purpose and Scope

This guide provides the reader with an understanding of the data model and features supported by the ON module. To perform the tasks in this document requires that you have metadata 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 Software Terms of Use

The Order Negotiations (ON) 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.

2 Overview

The ON module is an application for entering orders. It includes functionality from product definition and management, to order creation, checkout and credit authorization in the form of a workflow process. Once the order is submitted, it can then be processed through the Order Management (OM) module.

2.1 Features and Functions

The ON module provides the ability to create and manage a quotation. This section contains a list of features available in the ON application. However, it also contains the following features:

- **Create quote**: Creates a quote with a list of existing customer owned services.
- **Add offer to quote**: Adds an instance of an offer (offer and dependent-related items as specified by catalog) to the quote. Model customer's request for a new service for provisioning.
- **Change offer on quote**: Changes a specific offer on the quote to a new offer. Model customer's request to change a service (for example, upgrade/downgrade existing offerings) for provisioning.
- **Transfer**: Transfers the ownership of a service from one account to another for a specific customer. Model customer's request to change the owner of the service (hence all relevant charges involved on the service).
- **Relocate**: The Relocate feature provides the ability to move services from one location to another within a customer's account. Model customer's request to change the location for which a service needs to be delivered (e.g. customer moves or is relocated and hence Internet service needs to be delivered to a different address)
- **Remove offer from quote**: Removes a specific offer on the quote. Model customer's request to remove a service (for example, disconnect a service) for provisioning.

- **Validate quote**: Validates the created quote satisfies all requirements and business rules as defined in catalog. (for example, Internet Modem must be ordered with an Internet service)

- **Submit quote**: Indicates the quotation has been finalized. Order Management will start processing the order to fulfill services requested by the customer. Once order has been submitted, modifications cannot be made.

- **Revise order**: This provides the ability for a quotation or order to be revised. Historical records of each revision will be kept.

- **Bundle support**: Supports Catalog Bundles (for example, Triple Plan Bundle includes subscription of Internet Service, Voice Service, and Cable Service).

3 Quick Start

The following are the quick steps to install the ON module:

1. Install Velocity Studio; follow the directions from the Velocity Studio's Installer User Guide to install Velocity Studio, initialize the database, and configure the System Configuration application.
2. Create a project in Velocity Studio and set the internal name of the project.
3. Add the library JAR files for the ON module.
4. Start the runtime and update the logical connection in the System Configuration application.
5. To create indexes and database tables, run the **order_negotiations.sql** file from <installation_folder >\modules\order_negotiations\DDL folder and run **catalog.sql** file found in the <installation_folder >\modules\catalog\DDL folder.
6. Upgrade the database and run the associated SQL file.
7. Click the **Runtime** > **Run** from the Velocity Studio's menu bar.
8. Log in to the User Profile Manager application using *upadmin* credentials for **Username** and **Password** fields.
9. Assign the privileges required to run the ON module and create a default calendar.
10. Logout and log back in from the runtime.
11. Double-click the Order Negotiations application's icon.

4 Installation and Setup of ON

This section provides the details on how to install and configure ON module.

4.1 Set up Velocity Studio and Database Schema

1. Install Velocity Studio as outlined in the *Installer User Guide* > *Standard Install.*
2. Create a new schema in your database. Refer to the *Installer User Guide* > *Standard Install* > *Database Initialization.*
3. Run the *<EOC-ECM_installation_folder>*\DDL\CW.sql file. Use your newly created database schema to run this script. Update the `CW.sql` with the new user name.
4. Open the *<EOC-ECM_installation_folder>*\designer\env\startDesigner.cmd file to start Velocity Studio.
5. Create a Project in Velocity Studio.

    a. Click **File > New > New Project** from the menu bar.
    b. From the Select an empty directory dialog, specify the folder where you want to save your new project.
    c. Click the root metadata node, by default appears as AVM Metadata. On the General properties of this node, enter project's internal name in the **Internal Name** field.

6. Click **Database > Connect** from the menu bar to connect to your newly created database schema.
7. From the Database Login dialog, click the **New** button to configure the database connection settings.
8. The Connection Properties dialog appears; enter the name for the connection and the name of your database schema user in the **Name** and **User** fields, respectively.

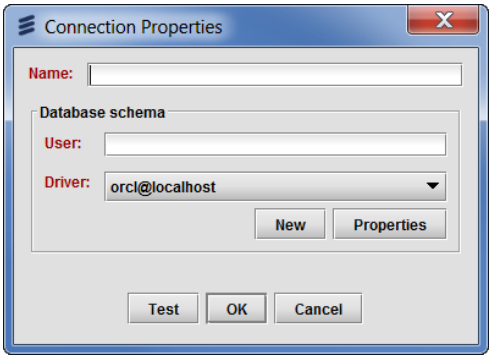


Figure 1 Connection Properties Dialog

9. Click the **New** button.
10. The Driver Properties dialog appears; click the **Driver type** field's drop-down menu and select **Oracle thin**. Proceed to enter the **Host, Port, Connection Type,** and **Service** information.

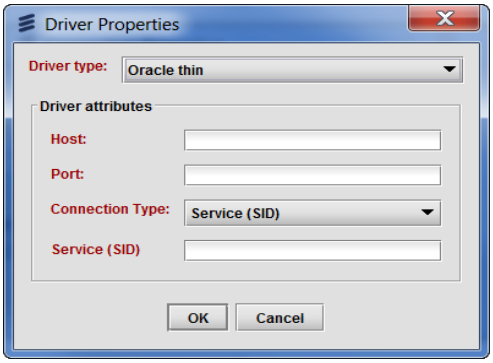


Figure 2 Driver Properties Dialog

11. Click the **OK** button.
12. The Connection Properties dialog reappears. You can click the **Test** button and then enter the password to test the connection.
13. Click the **OK** button to return to the Database Login dialog.
14. Enter the value in the **Password** field and click the **OK** button to connect.
15. Click **Runtime** > **Run** from the menu bar to run the framework.
16. The Select Application dialog appears; click the **New** button.

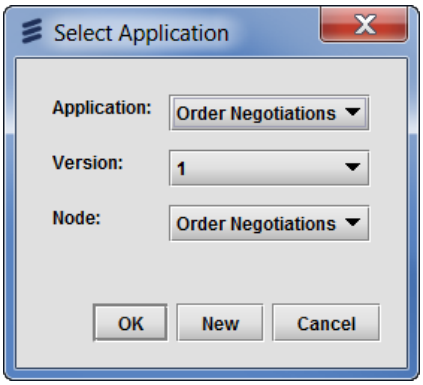


Figure 3 Select Application Dialog

17. The Add dialog appears; enter the value for the **Version** and **Description** field.
18. Click the **OK** button from the **Add** and **Select Application** dialogs.

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

4.2 Set Up the System Configuration Application

The following are the steps to set up the system configuration application.

1. In your Web browser, access the System Configuration application by entering `http://localhost:8080/cwf/config` as the URL.
2. Enter **upadmin** as both your Username and Password, and then press the **Enter** key to login. The main screen of system configuration application displays:

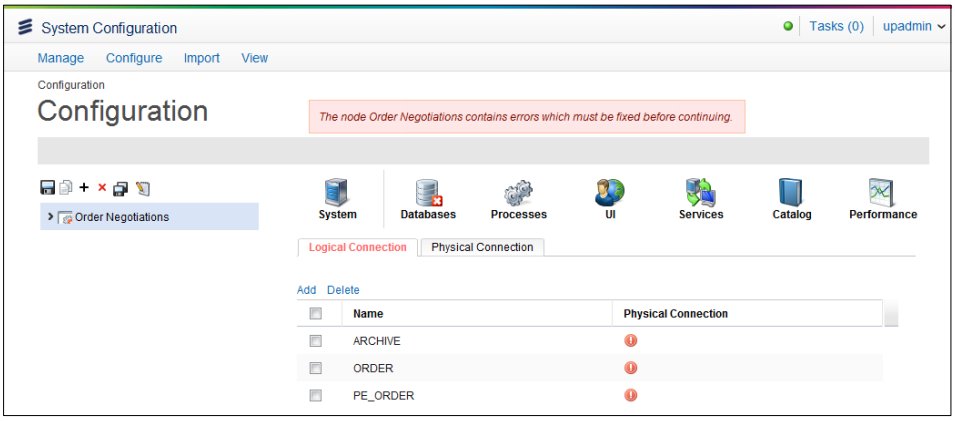


Figure 4 Main Screen of System Configuration Application

3. Select the main node (for example, Customer Information) from the node menu section, and then click the **Database** tab.
4. The **Logical Connection** tab displays the logical connections to the database, which are responsible for enabling the AVM to send database commands to the database, to carry out different functionalities.
5. Clicking **Databases** > **Physical Connection** displays the physical connections to the database, which are responsible for defining database connection parameters to be used by logical connections. Complete these steps to add a physical connection:

    a. From the Physical Connection page, click the **Add** button.
    b. The New Physical Connection dialog appears. Enter your database credentials.
    c. To test your connection settings, click the T**est** button. If your connection settings are properly set up, a *Successful Connection* confirmation message appears.
    d. Click the **Apply** button. A message appears, indicating that you have successfully updated your Oracle thin connection.
    e. Click the **Close** button.

Note: The database attributes in the System Configuration application need to match the database attributes in the Velocity Studio. For more information, refer to the *System Configuration User Guide*.

6. Click the Logical Connection tab to associate your logical connections to the physical connection you have just created, double-click each of the following logical connections and select your newly created physical connection from the drop-down list:

    - ARCHIVE
    - ORDER
    - PE_ORDER

7. Click the Save button to save your configuration settings, and exit the system configuration application.

    Note: You can select the **Active configuration** checkbox for **PE**, **PE_UI**, or **UI** node.

4.3 Add JAR files

Go back to the Velocity studio, and follow these steps to continue the configuration on the Velocity Studio side.

1. Click **Runtime > Stop** from the menu bar, to stop the runtime in Velocity Studio.
2. Click the root metadata icon (for example, AVM Metadata) in the left navigation menu, and then click the **Library** tab.
3. Click the Add button ( ) to launch the Select an template JAR dialog.

Figure 5 Add Library Files

4. Select the following JAR files required for ON module from < *installation_folder*>\modules folder. You must add all the required and recursive JAR files.

Module	Required JAR Files	Recursive JAR Files
Order Negotiations	order_negotiations.jar order_negotiations_lib.jar	address.jar api_common.jar billing.jar catalog.jar catalogClient.jar customer.jar cwInitialization.jar cwl_pscm.jar cwl_report.jar cws_pscm.jar data_dictionary.jar data_order.jar ecm.jar notification.jar party.jar pscm.jar report.jar serviceOrchestrationFramework.jar SIDCommon.jar wizard.jar

5. A Copy File dialog appears; select Yes to copy the JAR files to your local < <*project_folder*>\templates folder. If No is selected, the file path is added to your template folder.

6. Once the files are added, save the project metadata.
7. Reload or open the project for the library files to take effect.
8. To create indexes and database tables, run the **order_negotiations.sql** file from <installation_folder >\modules\order_negotiations\DDL folder and run **catalog.sql** file found in the <installation_folder >\modules\catalog\DDL folder.
9. Click **Runtime** > **Run** from menu bar to start the runtime. The Velocity Studio runs in configuration mode. New logical connections are available in the System Configuration application.
10. Follow the steps described in the previous section of this document to login to the System Configuration application, and to associate new logical connection to the physical connection.
11. Click the **Save** button and exit the System Configuration application.
12. To make the existing database schema compatible with the new files and settings, upgrade the database by following 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 (for example, upgrade.sql), and then click the **Save** button to create the file.
    c. Use SQLPlus or SQL Developer to connect to the appropriate database and run this upgrade file.

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

4.4 Assign the Privileges

To assign the privileges, complete the following steps:

1. In the Velocity Studio, click **Runtime > Run** from the menu bar or click the run button () to start the framework.
2. Open your Web browser and enter the `http://<localhost>:<port>/cwf/login` Web address. For example, `http://localhost:8080/cwf/login`.
3. Enter the username and password to login (for example, `upadmin` for both the **Username** and **Password** fields), and then click the **OK** button.
4. Open the User Profile Management application and click the **Manage > Groups** from the menu bar.
5. On the Search Group page, click on the **Search** button to get the list of the user groups.
6. Double-click the appropriate user group (for example, User Profile Administrators).
7. On the Select Privileges page, first click the **Edit** button and then click the **Add** button. The Search Privileges page appears with the available privileges for that group.
8. Select all privileges and click the **Select** button. A message appears that the privilege has been added successfully.
9. Click **Upadmin** option from the menu bar, and then click the **Logout** option.
10. Log back in to the application; follow the steps defined previously in this section.
11. The **Application Selection** page appears with the available applications.

12. Double-click the icon for Order Negotiations module. The main screen of the application appears as follows:

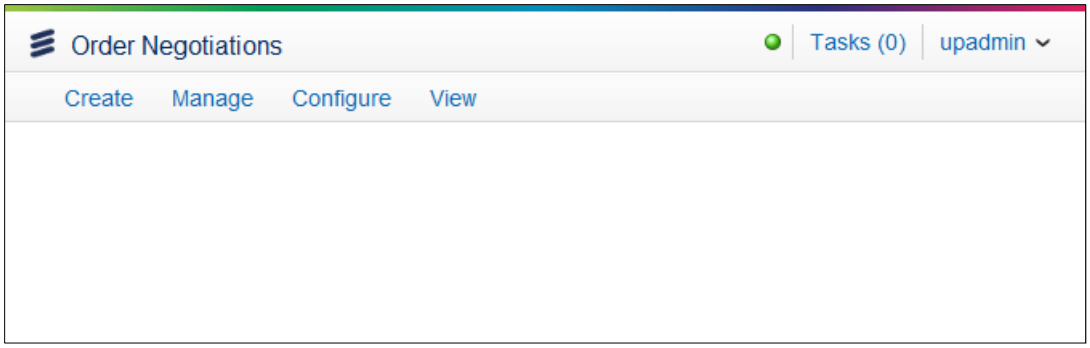


Figure 6 ON User Interface

Notes:

- If the login screen does not load, verify that either the Web address is correct or contact your system administrator to verify that you have the correct Web address.
- The ON application does not contain any data. To use this application, you need to import the data from code tables.

4.5 Add Code Tables

The installation of the ON results in a blank (no data) database. This section describes the process of importing the code tables through Ericsson Catalog Manager (ECM) application.

1. Login to the ECM application; follow the steps described in this document.
2. From the Overview page of the ECM application, select **Tecnical Configuration** > **Code Tables** from the Quick Start section.
3. The Code Tables page appears; click the **Import** button.
4. From the Import dialog, click the **Browse** button.
5. Select one of the following files available in the <*installation_folder*>\modules\order_negotiations\code_tables), and then click the **Import** button:

   - cwtWizardCTs.xml
   - ocategory.xml
   - ofamily.xml
   - cwt_creditCheckResult.xml

Code Table File	Label	Description	Module
ocategory.xml	Offer Category	List of offer categories.	Catalog Management

Code Table File	Label	Description	Module
ofamily.xml	Offer Family	List of offer families.	Catalog Management
cwtWizard CTs.xml	cwt_wizardStepType cwt_wizardStepSubType cwt_extScriptType	Code Tables used in Wizard framework	Wizard
cwt_credit CheckResults.xml		Code table for the credit check results of customers	

4.6 Sample Order Entry Flow

The order_negotiations_lib.jar file contains a sample order entry flow that can be exported from Velocity Studio and Imported into Order Negotiations:

1. Export the ON default flow and default external scripts from Velocity Studio:

   a. Click the **Resources** tab and open the **ONImports** folder.

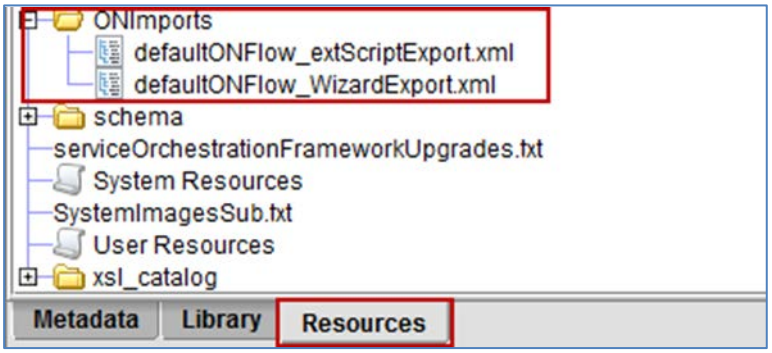


Figure 7 ONImports

   b. Right-click the following ONImport files and save the files to a local directory:

      - **defaultONFlow_extScriptExport.xml**
      - **defaultONFlow_WizardExport.xml**

2. Import the External Script into the ON application.

   a. From the Order Negotiations application, click **Configuration** > **External Scripts** option from the menu bar.
   b. Click the **Import/Export** button, and then select the **Import From File** option.
   c. Enter the path of the exported **defaultONFlow_extScriptExport.xml** file and then click the **Upload** button.

d. Follow the same steps to import the defaultONFlow_WizardExport.xml file.

4.7 Customer Information Management and Service Registry

Order Negotiations is integrated with Customer Information Management (CIM) and Service Registry (SR). To successfully utilize the integration to these modules, they must be installed. For detailed information, refer to the associated configuration guides for each module. The following are some quick start steps for installing the CIM and SR modules:

1. To use ON with Customer Information Management (CIM) and Service Registry (SR), add the required and recursive JAR to the library files as follows:

Module	Required JAR Files	Recursive JAR Files
Customer Information Management	catalogClient.jar customer_information_management.jar	address.jar api_common.jar billing.jar customer.jar cwl_address.jar cwl_customer.jar cwl_party.jar data_dictionary.jar notification.jar party.jar
Service Registry	catalog.jar customer.jar cwl_service_registry.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

2. Run the appropriate SQL files:

   - service_registry.sql (<installation_folder>\modules\service_registry\)
   - customer_information_management.sql (<installation_folder>\modules\customer_information_management\)

3. Import the code tables for SR and CIM:

   a. The following SR code tables can be found in the < installation_folder >\modules\service_registry\code_tables.

      - cwtsr_applicationContext.xml
      - cwtsr_entityType.xml
      - cwtsr_associationType.xml

   b. The following CIM code tables can be found in the <installation_folder > \modules\customer_information_management\code_tables.

      - cwt_addressRole.xml
      - accountStatusCT.xml
      - associationType.xml
      - customerStatusCT.xml
      - cwt_addressRole.xml
      - cwt_contactMediumType.xml
      - cwt_creditCardType.xml
      - cwt_creditCheckResult.xml
      - cwt_customerSubType.xml
      - cwt_customerSubType_Commercial.xml
      - cwt_customerSubType_Residential.xml
      - cwt_customerType.xml
      - cwt_externalSystem.xml
      - cwt_IndIdentificationType.xml
      - cwt_noteSubType.xml
      - cwt_noteType.xml
      - cwt_orderStatus.xml
      - cwt_organizationType.xml
      - cwt_OrgIdentificationType.xml
      - cwt_OrgNameType.xml
      - cwt_party_PartyRole.xml
      - cwt_SecurityQuestions.xml
      - cwt_siteResponsibility.xml
      - cwtdictOccupancyType.xml
      - cwtdictTechnologyType.xml
      - Industry.xml
      - iso3166.xml
      - iso4217.xml
      - iso6391.xml
      - iso6392.xml
      - provinces.xml

Note: You must upgrade your database, configure the logical connections and assign the privileges for each module.

5 Data Model

5.1 Catalog Items Logical Model

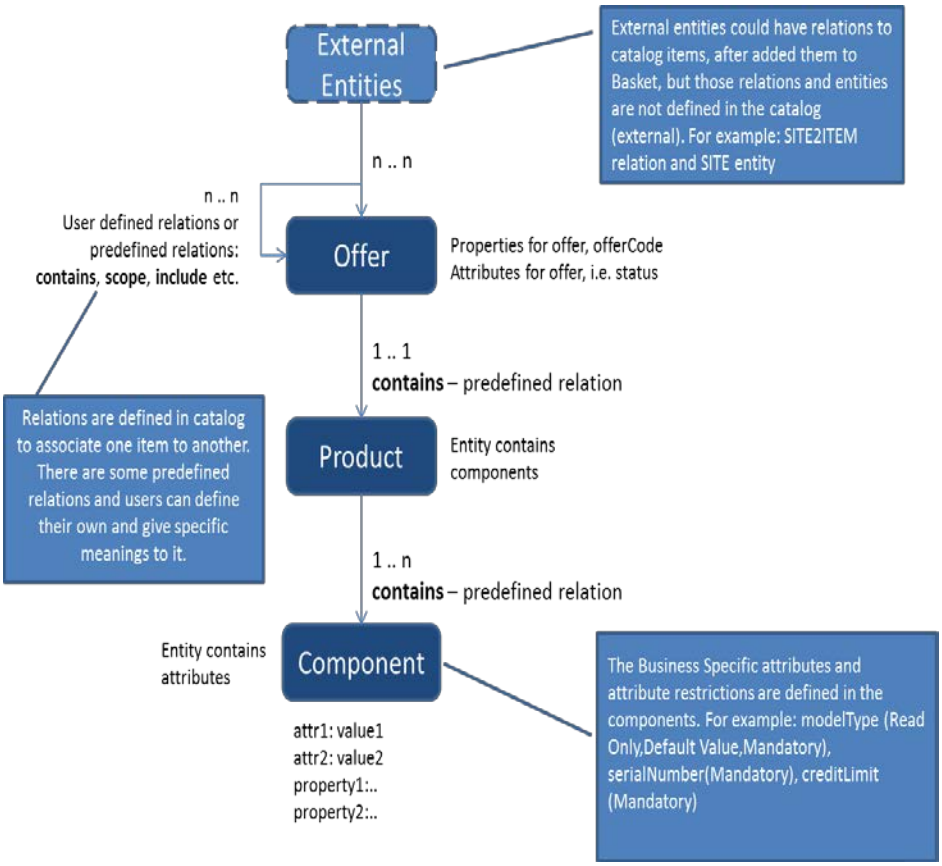


Figure 8 Catalog Items Logical Model

ON requires the user to define catalog items and their structures representing offers, products and business rules in the Product Catalog before they can be used in ON for order capturing. It is important to define the entities and relations according to the logical model shown previously. The following is an example.

Node	Mandatory	Permissions	Min	Max	Instance key
DataOrder					
Sites	☐		0	100	
Site	☑			Wizard	☑

Product Catalog, user should use “Relations” to link SCOPE, CONTAIN or BUNDLE relations between offers.

5.2 Orders

5.2.1 Data Order (cwt_do: dataOrder)

Data Order is the internal data representation of an order and shared among many other models. It contains a hierarchy of collections and instance of documents, which can be populated, stored in the database, and retrieved accordingly.

5.2.2 UI Order (uiOrder:uiOrder)

UI Order is a more user friendly view of the Data Order used by Order Negotiations and Order Management and it is never persisted in the database. It also has its own data hierarchy but most of the collections are references to Finders which retrieve information from the Data Order’s nodes.

Hierarchy

Node	Mandatory	Permissions	Min	Max	Instance key
UI Order					
Summary	☑		0	1	
Sites	☑		0	50	
Node Visual Key	☑	Hidden			☑
Offer List	☑		0	1	
Family	☑	Hidden	0	20	
Node Visual Key	☑	Hidden			☑
Family List	☑		0	1	
Category	☑		0	50	
Node Visual Key	☑	Hidden			☑
Category List	☑		0	1	
Configuration|	☑		0	1	

5.3 Documents

5.3.1 Order Instance (cwt_do : orderInstance)

Entity/Table Name	Description	
Order Instance	It stores all additional information about an order including customer information, order number, type and more. It is always accessible directly from an order (for example, order.orderInstance)	
Attribute/Column Name	**Type**	**Description**
(PK)cwDocId	string(16)	ID
state	string(3)	Order state
metadataType	int(9)	Numeric metadata type
status	string(1)	Order validation status
visualKey	string(64)	Visual Key of Order (for example, AD 47397 – Draft)
productCode	string(256)	Product code
parentOrder	string(16)	Parent of Order
owner	string(64)	Order owner
state2	string(3)	Order state 2
HasAttachment	Boolean	Whether order has attachment
metadataVer	decimal(9)	Metadata Version
orignalOrderId	string(16)	Original Order ID
sourceOrderId	string(16)	Source Order ID
kindOfOrder	string(1)	Kind of Order (Order, Change Order)
orderPhase	string(1)	Order phase (In Entry, In Progress, Completed)
projectId	string(16)	Project ID
processId	int(16)	Process ID

dueDate	date	Due Date
oState	string(16)	Order state (for example, Draft, Presented, Cancelled)
customerId	string(32)	Customer ID
accountId	string(32)	Account ID
orderType	string(16)	Order Type (for example, Add, Move, Change, Disconnect)
orderSubType	string(16)	Order Subtype (for example, Cancel, Revise)
srcSiteId	string(32)	Source Site ID
desSiteId	string(32)	Destination Site ID
relatedOrder	string(18)	Related Order ID
orderNum	long	Order Number
ordVer	int(16)	Order Version
effectiveDate	dateTime	Target Effective Date
submittedBy	string(64)	Submitted By
submittedDate	dateTime	Submitted Date
price	decimal, 14.4	Total Price
onetimePrice	decimal, 14.4	Total One Time Price

Attribute/Column Name	Type	Description
srcSiteId	Source Site ID	Source Site ID
desSiteId	Destination Site ID	Destination Site ID
relatedOrder	Related Order ID	Related Order ID
orderNum	Order Number	Order Number
ordVer	Order Version	Order Version
effectiveDate	Target Effective Date	Target Effective Date

submittedBy	Submitted By	Submitted By
submittedDate	Submitted Date	Submitted Date
price	Total Price	Total Price
onetimePrice	Total One Time Price	Total One Time Price
pricedOn	Priced On	Priced On

5.3.2 Offer (cwt_do:basketItemEx)

Entity/Table Name	**Description**	
BasketItemEx	It stores the instances of a specific Catalog Offer, Product and Component during the Order Entry process. Some important fields are GUID (which is the globally unique id of this particular offer instance), and others like state and source. It's also extensible by users who wish to include additional fields.	
Attribute/Column Name	**Type**	**Description**
basketcreationdate	dateTime	Catalog Basket Creation Date
basketid	string(10)	Catalog Basket ID
(PK) basketitemid	string(10)	Basket Item ID
cwcreated	dateTime	Item Creation Date
cwcreatedby	string(64)	The Application user who create the Item
itemcode	string(16)	Item Code
lastupdateddate	dateTime	The Date when the Item updated Last time
parentbasketitemid	string(16)	Parent Item ID
Attribute/Column Name	**Type**	**Description**
relationname	string(32)	Relation Name
requesttime	dateTime	Request Time
sequenceno	Decimal(4)	Item sequence No
state	String(16)	Item State
guid	string(16)	GUID

status	string(8)	Offer status (For example, Added, Deleted, Changed, Active)
prevStatus	string(16)	Previous Offer status (For example, Added, Deleted, Changed, Active)
source	string(3)	Source (For example New, Existing)
ItemsNotes	string(3200)	User note on the Item
effectiveDate	dateTime	Effective date
serviceStartDate	dateTime	Service Start Date
isCompleted	string(1)	Shows whether Item has completed its lifecycle successfully or not .

Customer-specific attributes per offer can be stored in documents by extending the Offer/Product/Component Instance documents. For example, the component “Set Top Box” has two additional fields ATSTBMAC and ATSTBSNO. The Component document can be extended by mapping these 2 attributes to a new custom table:

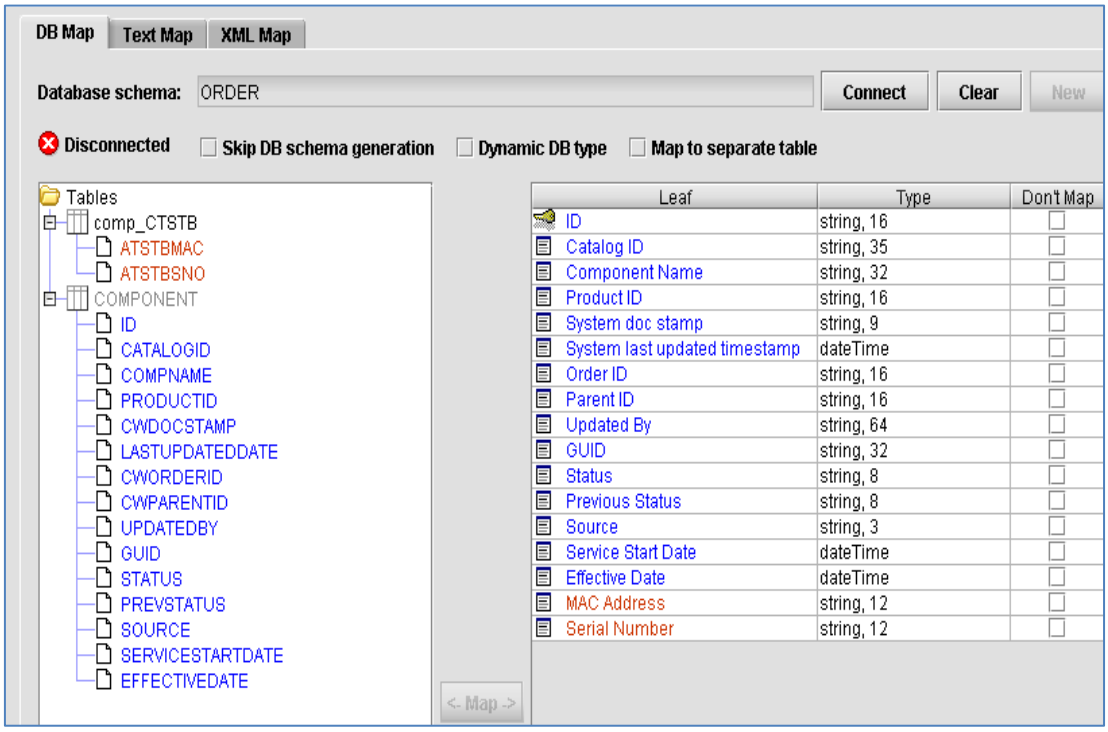


Figure 9 Database schema

5.3.3 Charge (cwt_pcoe:charge)

Entity/Table Name	Description	
Charge	It stores the instances of a specific charge associated with an Offer/Product during the Order Entry process. Some important fields are Charge Type (which is the Charge's identifier), GUID (which is the globally unique id of this particular charge instance), and others like Offer/Product Instance ID, Type, Price Type and Charge Amount.	
Attribute/Column Name	**Type**	**Description**
(PK)id	string(16)	Price ID
chargeID	string(32)	Charge ID
chargeType	string(16)	Charge Type
mrc	decimal(14.4)	Monthly Recurring Charge
amount	decimal(14.4)	Charge Amount
otc	decimal(14.4)	One Time Charge
tax	string(16)	Tax
Attribute/Column Name	**Type**	**Description**
productId	string(16)	Offer/Product Instance ID
compName	string(32)	Component Name
compID	string(32)	Component ID
type	string(1)	Type (For example, Charge, Tax, Discount, Group Charge)
period	int(4)	Period
priceType	string(32)	Price Type (Monthly Recurring Charge, One Time Charge)
priceVersion	string(32)	Price Version
priceListQuantity	decimal(12.4)	Quantity
displayOnly	Boolean	For display only

currency	string(3)	Currency

5.4 Catalog Object Filter (cwt_on:catalogObjFilter)

Entity/Table Name	**Description**	
catalogObjFilter	Catalog Object Filter is a document used to define how the offers would appear in the UI Order tree. Normally it uses Offer Family, Offer Category and a sequence number to sort the collection on the UI Order Tree.	
Attribute/Column Name	**Type**	**Description**
(PK)code	string(16)	Code
(PK)cttype	string(16)	Type
Attribute/Column Name	**Type**	**Description**
(PK)family	string(16)	Family
label	string(128)	Label
seq	integer(3)	Sequence
active	boolean	Active

6 ON Extensions

In principle, every script/document/finder with open access can be extended/overwritten, but the most important items affecting the business logic and display are described below in bold font.

6.1 Scripts

6.1.1 cwt_on_ovr

The cwt_on_ovr namespace contains all scripts/data types that expected to be extended in order for ON to function. Refer to the cwtPlugin_on_cim for details regarding utilization of these API scripts.

- **_startON(contexts)** – Deprecated, **use _startONWizard(dataOrder)** instead. Display the Customer Finder to start an ON session.
- **_startONWizard(dataOrder)** – Starts the wizard defined for the current ON and initialize wizard global parameter object.
- **_exitON(uiOrderHelper)** – defines what to do when user decides to exit ON's UI
- **_customerValidate(dataOrder,msgs)** – does a simple customer validation to see if this customer/account is able to create an order.
- **cimCopyQuote(orderId)** – copies an order.
- **cimNewQuote(customerId, accountId)** – create a new quote and UI Order for display.
- **cimOpenQuote(orderId)** – open an existing data order and create UI Order for display.
- **cimQuoteFinderSeelct(customerId)** – returns an array of quotes to be displayed in the UI Order tree.
- **createCustomerDataObject(customerId)** – returns the customer data helper for access to customer API.
- **gotoCustomer360** – returns the UI of the main customer view.
- **sendOrderToSR(dataOrder)** – sends the order in parameter to SR.
- **getAccountServices(ord, accountId, customerId)** – retrieve all SR assocoations related to the account and populate to data order.

6.1.2 cwt_do

- **_getVisualKey(ord)** – overwritten by **cwt_on_do_ext.doGetVisualKey(ord)**.
- **_getSiteVisualKey(siteId)** – overwritten by **cwt_on_ovr.getSiteVisualKeyById**.
- **ClsCatalogHelper** - This class provides a wrapper to Catalog
- **ClsDataOrderHelper(order)** – data order helper to cache data order objects and provide operations on data order instance.
- **ClsInterface** – define constants related to Data Order Helper.

6.1.3 uiOrder

- **ClsUIOrderHelper(order)** – a helper class that stores the current state of the UI Order (For example, current data order, catalog filter, finder, etc.), and provides access functions to retrieve these objects.
- **createUIOrder(dataOrder, useExistingCategory)** – create UI Order based on data order and generate the family/category collection on the UI Order tree.
- **generateSummaryList()** – generate the order summary table.
- **getDataOrderMT()** – returns the metadata name of the data order.
- **getOfferFinderFilter(finder)** – find out the currently selected item and its associated finder on the UI Order tree.
- **getOfferFinderResults(finder, objectList)** – call generateSummaryList if top tree item is selected. In all other cases, displays the list of added offers based on the selected family/category on the UI Order tree.
- **permission_viewUIOrderMenu** – do not display any of the UI order menu item if no data order is set in the context.
- There are more finder action scripts (in green) and permission scripts (in maroon) that are in this namespace to allow changes or control the UIOrder.

6.1.4 cwt_on

- **$CONSTANTS** – define ON-related constants to be used throughout the application, it's been shifted to external script ON$getConstants for more convenient customization
- **_abandonOrder(orderInstance, order, override)** – set data order to Abandoned state for non-running/parking (if override is true, running orders also) orders.
- **_addOfferToOrder(doh, offerDoc)** – add offer to order using Data Order Helper.
- **_canChangeOp(from,to)** – a check to make sure one can change the orderType or operation type form one to another
- **_canReviseInflightOrder(dataOrder)** – should be overwritten to determine whether an order can be revised when it is inflight. Default is true.
- **_checkoutValidate(dataOrder)** – instantiate the cwt_on.ClsValidate class and perform validation.
- **_cleanUp()** – clean up context session parameter and delete the UI Order.

- **_completeErrorOrder(oi, dataorder, override)** – set order state to Completed for error orders.
- **_configValidate(dataOrder)** – should be overwritten to provide validation during configuration, achieved by double clicking on a row within the Product Configuration Finder.
- **_creditCheck(doc)** – should be overwritten to perform credit checks.
- **_customerValidate(dataOrder)** – perform validation on the customer during cwt_on_cim_ext.cimNewQuote()
- **_exitON(uoh)** – should be overwritten, perform clean up when ON session ends and display Customer 360.
- **_gotoStepFromState(uiorder)** – redirect to different steps of the ON session, based on the order state.
- **_initOfferFields(dataOrd)** – to initialize some offer's fields when the order is first created
- **_performDisconnect(dataOrder)** – perform disconnect action on the data order when user changes the order type to Disconnect/Move-Disconnect.
- **_performUndelete(dataOrder)** – perform undelete action on the data order when user changes the order type to Change from Disconnect/Move-Disconnect.
- **_removeOfferFromOrder** – dataorder manipulation script to remove an offer.
- **_submitFinalCheck(dataOrder)** – Performs final checks (_checkoutValidate) before submitting order to OM. Can be overwritten.
- **_submitOrder(ord, doc)** – Sets the effective date on the order and its offers during Submit order and Cancel Order (for inflight order). Can be overwritten.
- **_updateOrderStatus(dataOrder, state)** – update data order to the specified state.
- **abandonOrders(accountId, curOrdId)** – expires all other non-submitted orders with this accountId when the current order is submitted and not inflight.
- **ClsFilterHelper(obj)** – helper class for Catalog Filter.
- **ClsONContext(ord)** – stores all the relevant information (For example, current customer, order, catalog offer, etc.) as session parameter during the ON session.

- **ClsValidate(dataOrder)** – shifted to external script ON$ClsValidate perform order validation during checkout.
- **copyOrder(srcOrd)** – duplicate a copy of the source data order.
- **displayMessages (header, msgArray)** – display validate messages in a dialog box.
- **filterOffersForON(dataOrder)** – filter/remove offers in data order according to the exclude list defined in external script: ON$getConstants.$excludedSRCategorySeq
- **getConfigVariable(variable)** – get cwt_on config variables.
- **getCustDataFromCache (customerId)** – instantiate a cwt_cim.ClsCustomerDataHelper class that retrieves customer information and stores it in cache.
- **getCustomerDetails(dataOrder, customerId, accountId)** – retrieve customer information from cache.
- **getNextOrderNum()** – generate and return the next order num.
- **getONContext()** – retrieve ON Context instance from session.
- **getOrderDetails(object)** – calls cwt_onom.getOrderFromObject(object)[cwt_on.getOrderDetailsOrderItemName()].
- **getOrderDetailsOrderItemName()** –defines the order item name that contains the order details. Can be overwritten.
- **getOrderFromObject(object)** – gets order from various kind of objects (For example, ProcessAcitvity/Process/Order/WorklistItem/Finder/Document).
- **getOrderMetadataType()** –returns the metadata name of data order. Can be overwritten.
- **getProcessMetadataType()** –returns the metadata name of process. Should be overwritten.
- **getUIOrderMT()** –returns the metadata name of UI Order. Can be overwritten.
- **initCurrentContextObject(uiOrder)** – initialize the UI Order and ON Context. Set ON Context in session and display UI Order.
- **isOrderManuallySubmitted()** – indicates if an order is manually submitted.
- **modifyCompValue(compCls, attrName, attrVal)** – Deprecated. Modify the corresponding component value.

- **orderWizard_*()** – a set of functions to handle actions performed on the ON wizard, many were deprecated and moved to wizard flow.
- **reviseOrder (dataOrder)** – copy existing order and set to Draft state, set existing order to Revise-Cancelled state, display the new Order.
- **reviseOrderCAD (dataOrder)** – copy existing order and set to CAD Configuration state, set existing order to Revise-Cancelled state, display the new Order.
- **moveOffers(dataOrder,fromSiteId,toSiteId)** – moves the offers in the dataOrder from one site to another
- **undoMoveOffers(dataOrder,fromSiteId,toSiteId)** – reverse the move offers operation
- **migrateOffer** - upgrade or downgrade offer
- **moveAccess** - relocate access and offers in it
- **moveOffer** - relocate individual offer
- **moveOffers** - relocate all accesses and offers in one site
- **transferItems** - Transfer account for access and offer

6.1.5 uiOrder

- **doubleClickAction_migrateItem** - Double click action handler for items migration on UI
- **moveAccess** - Action handler of access relocation on UI
- **moveOffer** - Action handler of offer relocation on UI
- **moveOffers** - Action handler of site relocation on UI
- **transferAccess** - Action handler of access transfer on UI
- **transferOffer** - Action handler of offer transfer on UI
- **transferOrder** - Action handler of order transfer on UI
- **transferSite** - Action handler of site transfer on UI

6.2 Documents

In principle, every document with open access can be extended/overwritten.

6.2.1 cwt_do

- **basketItemEx** – items(offer, product, component) in data order.
- **basketItemRelation** – Relation in data order.
- **orderInstance** – Data Order Instance.
- **orderInstanceSearch** – used by Data Order Finder and Order Finder in cwt_on.
- **siteDoc** – Site in data order.

6.2.2 uiOrder

- **nodeVisualKey** – represent a node in the UI Order tree.
- **offerResult** – used by Offer Finder in uiOrder.
- **offerSearch** – used by Offer Finder in uiOrder.
- **offerConfigSearch** – used by offer configuration finder in uiOrder
- **offerConfigResult** – used by offer configuration finder in uiOrder
- **offerCADResult** – used by offerCADFinder finder in uiOrder
- **updateFinalCustomerConfiguration** – used by offerCADFinder finder in uiOrder
- **paymentHold** – should be overwritten, default payment hold page.
- **creditCheck** – should be overwritten, default credit check page.

6.2.3 cwt_on

- **catalogObjFilter** – used by Offer Category Filter Finder to define the order of families/categories to be displayed in UI Order.
- **contextDocument** – order context document display at the top of UI Order.
- **description** – description popup when clicking on rows in Offers Finder and Catalog Item Pick List Finder.
- **itemPickListResult** – result document for Catalog Item Pick List Finder.
- **itemPickListSearch** – search document for Catalog Item Pick List Finder.

- **orderNumCounter** – stores the current order number, used to generate the next order number.
- **processDocument** – Not used. Process document for Order Negotiations process.

6.3 Finders

In principle, every finder can be overwritten.

6.3.1 uiOrder

- **offerFinder** – Offers Finder that displayed the current list of offers/products in the UI Order.
- **productConfigurationFnd** – Deprecated, user Offer Configuration Finder instead. Product Configuration Finder to display components in the UI Order.
- **offerConfigFinder** – Offer Configuration, it displays detailed Offer-Product-Component information. User is able to change attributes of components in this finder.
- **offerCADFinder** – configures the Committed Activation Date.

6.3.2 cwt_on

- **itemPickList** – Catalog Item Pick List Finder, used for adding offers/products to the order.
- **catalogObjFilterFinderD** – returns a list of catalogObjFilter documents that define the order of families/categories to be displayed in UI Order.
- **catalogObjFilterFinderS** – used by Catalog Item Pick List search document that displays a list of categories. The list of categories is determined by the selected family in UI Order. If no family is selected, all categories in all families will be listed.

6.3.3 cwt_do

- **dataOrderFinder** – Data Order Finder, used by cimQuoteFinderSelect function in cwt_on_cim_ext and Order Finder in cwt_on.
- **onOrderFinder** – ON Order Finder. Queries all the ON orders.

7 OM Extensions

In principle, every script, document, or finder with open access can be extended or overwritten. but the most important items affecting the business logic and display appear in this section in bolded font. The following Order Management extensions are available.

7.1 Scripts

7.1.1 cwt_om

The cwt_om namespace contains the following global scripts:

- **getOrderFromObject** – Different objects (for example, an order item, order group, worklist item, process, and data structure requests) have different ways to get either the order or the order ID.
- **process_resumeFamily** – This method has been re-implemented to use the resumeFamily() API method.
- **process_sendExceptionToChildProcesses** – This method has been re-implemented to use the sendExceptionToFamily(exception, type) API method.
- **process_sendSignalToEntireFamily** – This method has been re-implemented to use the Process.sendSignalToFamily(3) API method, where 3 indicates to send a signal to all process that have the same order ID.
- **process_setPriority** – This method has been re-implemented to use the Process.setProcessPriority() script.
- **process_suspendFamily** – This method has been re-implemented to Use the suspendFamily()APImethod.

7.1.2 Process Utility Java API Methods

The following utility API methods are available for processes:

- **Process.sendSignalToFamily** – This method's family means all processes that have the same Order ID. This method specifies target processes for sending a signal. Possible values for the type parameter are as follows:
    - **0** – Send signal to all immediate children
    - **1** – Send signal to all siblings
    - **2** – Send signal to the entire family (that is, all processes belonging to the same root process)

    - **3** – Send signal to all processes in the order family (that is, with the same order ID)

- **Process.setProcessPriority** – This method accepts the following flags:
    - **processID** – This flag indicates the process identification.
    - **Priority** – This flag denotes a new process priority, which is a number between 1 (highest priority) and 15 (lowest priority). If the priority is less than 1, a priority of 1 is used. If it is more than 15, a priority of 15 is used.
    - **includeChildren**. By default, its value is false. If it is true, all process children also have their priority updated.
    - **changeWorklistPriority**. By default, its value is false. If it is true, all worklist tasks that the process has created also have their priority changed.
- **resumeFamily(type)** – This method allows resuming a family of processes. The **type** parameter defines the scope of resuming a family processes. See Process.sendSignalToFamily for possible values.
- **sendExceptionToFamily** – This method allows sending exceptions to child processes. This method's type parameter defines the scope of sending an exception. See Process.sendSignalToFamily for possible values.
- **suspendFamily(type)** – This method allows suspending a family of processes. The **type** parameter defines the scope of suspension. See Process.sendSignalToFamily for possible values.

7.1.3 Standard Subflow Utilities

The following utilities for standard subflows are available:

- cwt_om.orderDecomposition
    - The before script's decomposeOrder activity publishes the DECOMPOSE_ORDER event. The default event handler invokes the cwt_om.decomposeOrder(this.process); global script.
    - The orderDecomposition method uses the following subflows:
        - **Rollback Point**: Dummy script activity with dummy compensate script activity, which is used to define a rollback point.
        - **Unexpected Event Resolution**: Subflow that handles exceptions.
    - The orderDecomposition method uses the following metadata objects:

- unableToIdentifyWorkflowToInitiate: Exception
- processSupport: Participant to handle an unexpected event
- processSupportIF: Interface for a process support participant

- cwt_om.holdForDueDate
  - The duration script's waitForDueDate script activity publishes the ACTIVITY_GET_DUE_DATE event. It does not have a default handler.
  - The orderDecomposition method waits for the due date, and then calls the global script to get this due date. The method allows you to use the following subflow:
    - **stub**: A dummy subflow that indicates No action on compensate.
  - The orderDecomposition method uses the following metadata objects:
    - reschedulingRequested: Exception
    - activity_waitForDueDate_getDueDate: global script
- cwt_om.forwardExceptionToChildren
  - The before script's sendNoticeToChildren script activity forwards the exception to all children of the process.

8 Data Structures

Data Structures are XSD based objects/models mainly used to allow Order Negotiations to communicate with external systems through interfaces like SOAP, MQ, etc.

All required XSDs may be generated by selecting the cwt_onapi namespace, right-clicking and selecting "Export Schema". The following XSDs are required:

- cwt_dict.xsd
- cwt_up.xsd
- cwf.xsd
- cwt_onapi.xsd

9 Order Negotiations External APIs

ON external APIs would allow external systems to interact with ON as long as they agree on the same data structure model (i.e. WSDL/XSD definition).

Currently supported APIs exposed to 3rd party systems consist of:

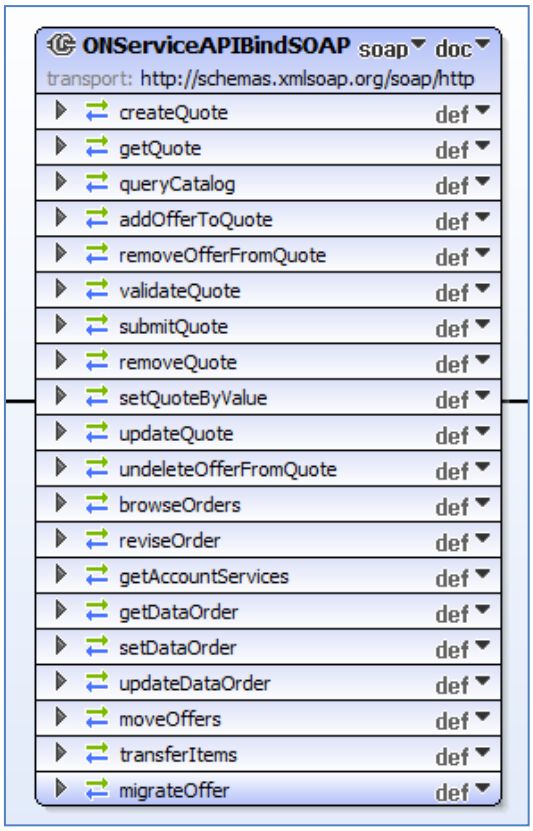


Figure 10 ON Service API

The content of the input and output messages is defined in the XSD and WSDL that may be generated from the product by selecting the **cwt_onApi** namespace, right-clicking and selecting “Export Schema” and “Export WSDL”, respectively.

As each deployment is expected to have a varying order model, it is expected that the default behavior of the module is altered by overwriting the internal methods that implement these functions located under the same namespace. Note that the bolded operations will directly operate on the dataOrder while others work with the UI Order.

Figure 11 ON Service API Methods

Similarly, additional operations may be implemented by extending the ON interface defined by “ONServiceAPIInt”, providing additional methods or functionality.

9.1 Data Model Specification

9.1.1 orderDS

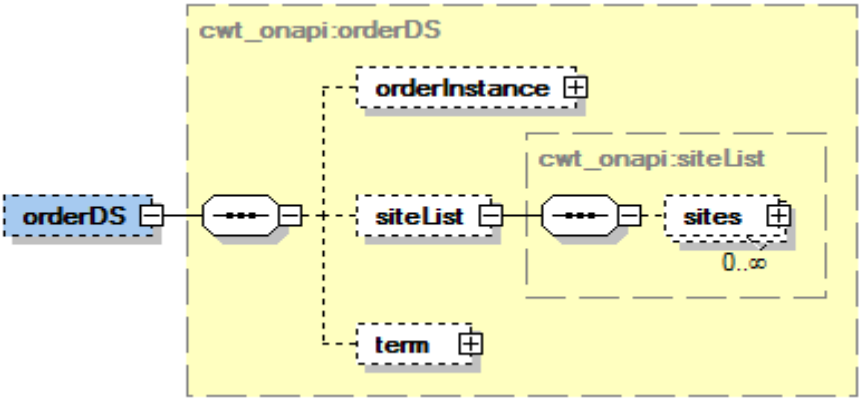


Figure 12 orderDS

Similar to what you see on the UI, it consists of an orderInstance data structure and a siteList, term is not being used. More detail to follow in subsequent sections.

9.1.2 Site

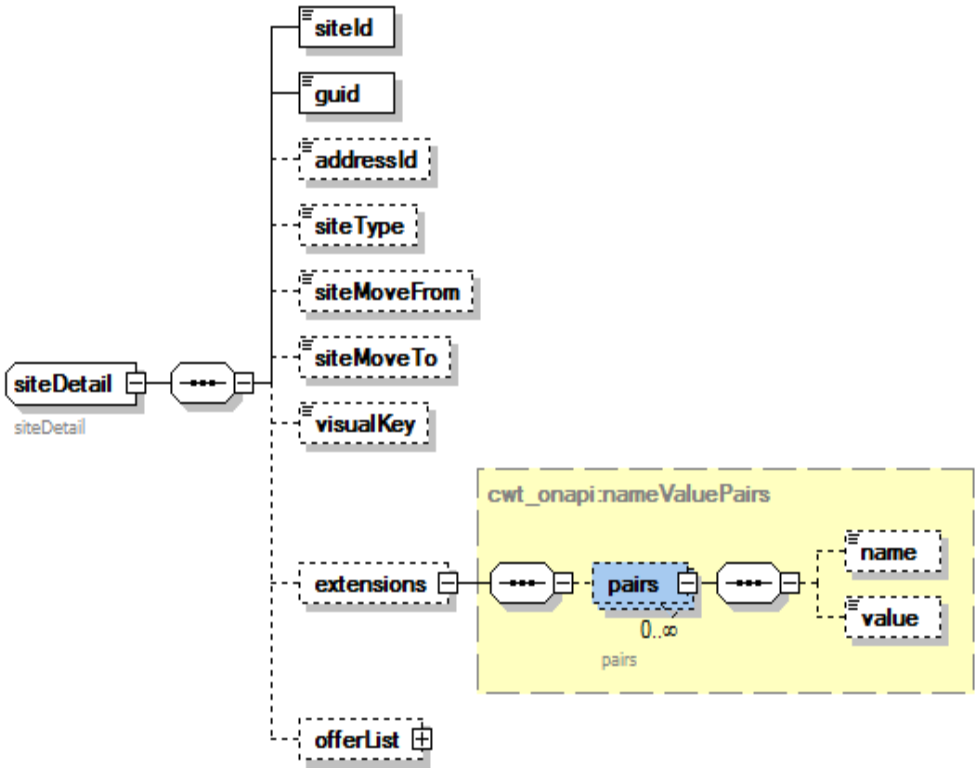


Figure 13 Site

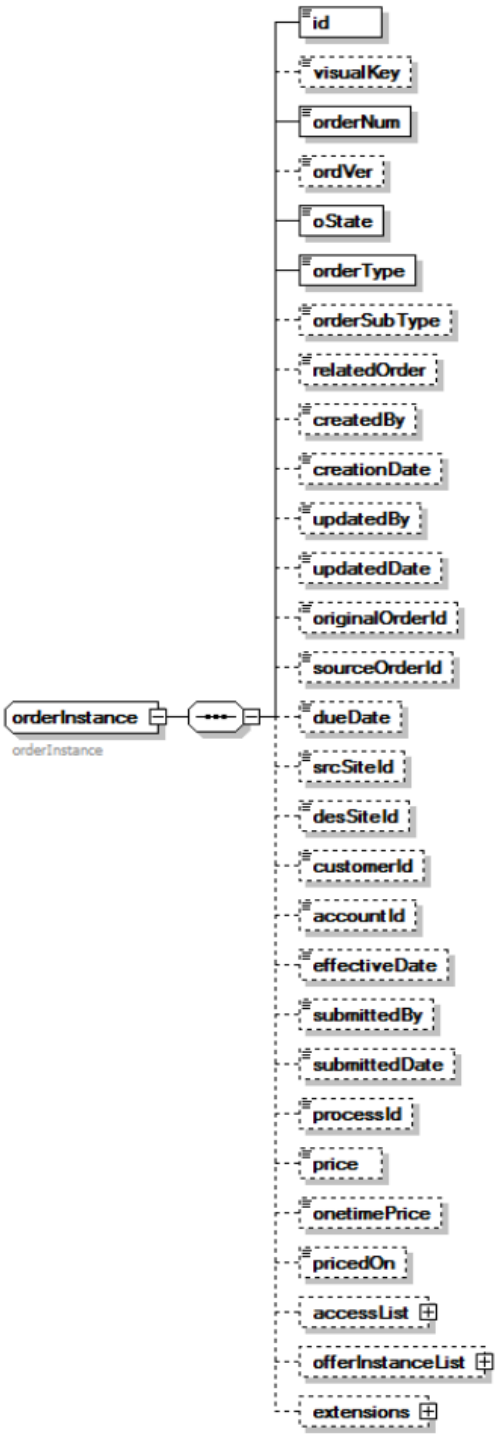
The siteDetail DS contains site information. Key field is siteId. Guid is not needed because siteId serves as the GUID of the element.

siteMoveFrom and **siteMoveTo** are used only in “MOVE” type of order, it specifies the source site and the destination site’s siteId if it’s populated

offerList will be populated **only** in orderDS, to represent the same hierarchy similar to the UI order view. But it’s not needed in dataOrderDS, because it uses the relations to relate from siteId to item’s guid.

extensions – again it is for user document extension.

9.1.3 orderInstance



id – document id and also the dataorder's id

orderNum – Conventional order number

ordVer – order version if order has been revised they will have the same orderNum but different order version number

oState – State of the order, consult data type cwt_do:oState for more info

orderType – current type are defined ADD, CHANGE, MOVE, DISCONNECT, MD(Move-Disconnect) and MA (Move-Add)

orderSubType – used when cancelling or revising an inflight order, possible values are "CANCEL" and "REVISE"

relatedOrder – used only to pair MD order and MA order

originalOrderId – the initial revision's data order id

sourceOrderId – the previous revision's data order id

effectiveDate – the tentative date when the order should be effective, it could be either the date the services in the order went active, disconnected or changed.

submittedBy – the user's id who submitted this order

price – total monthly price of the order

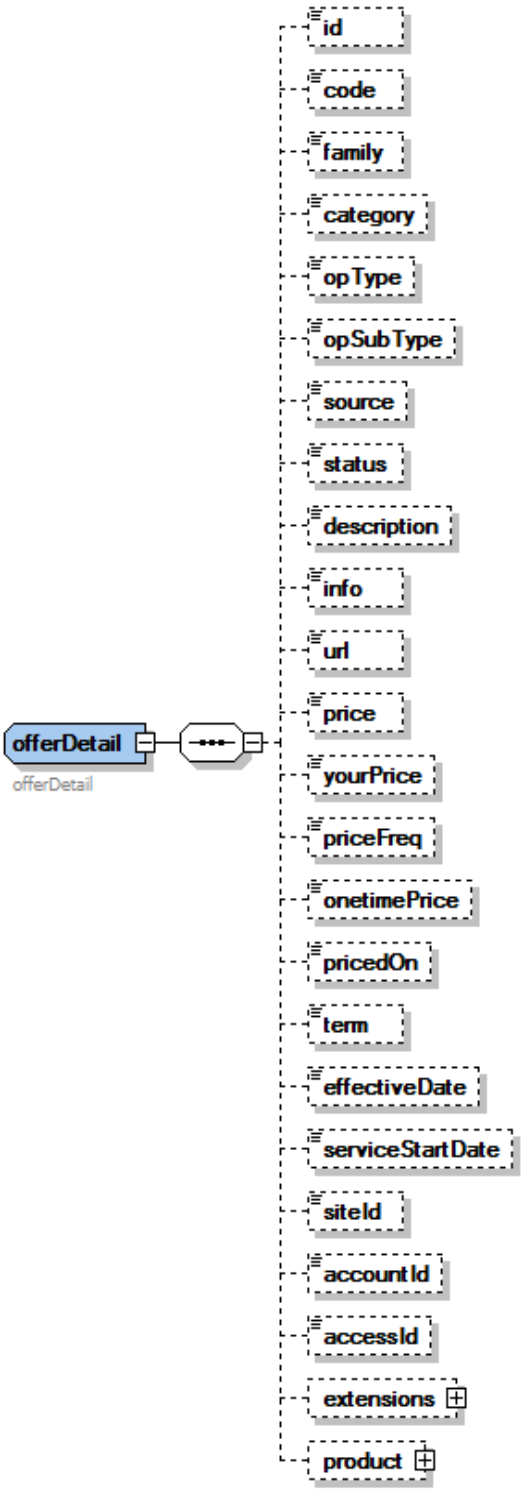
onetimePrice – total one time price of the order

extensions – a list of name value pairs capturing any attributes populated from user extension of the document data model

accessList – a list containing all access offers, it's normally not populated unless the flag in getDataOrderRequest was set

offerInstanceList - a list containing all offers, it's normally not populated unless the flag in getDataOrderRequest was set

9.1.4 offerDetail



id – offer id

code – offer's catalog code

family – catalog family values are defined in ofamily code table

category – catalog category values are defined in ocategory code table

opType – it is an attribute only to access type offers, possible values are ADD CHANGE DISCONNECT

opSubType – not used yet

source – indicates either "EXT" for existing one or "NEW" for newly added one

status – item status is defined as a enumeration, consult the data type cwt_do:orderItemStatus

effectiveDate – the tentative date when the offer should be effective, it could be either the date the services in the offer went active, disconnected or changed.

serviceStartDate – this is the start date of when the offer was in service, it's empty for new ones

price – monthly price of the offer

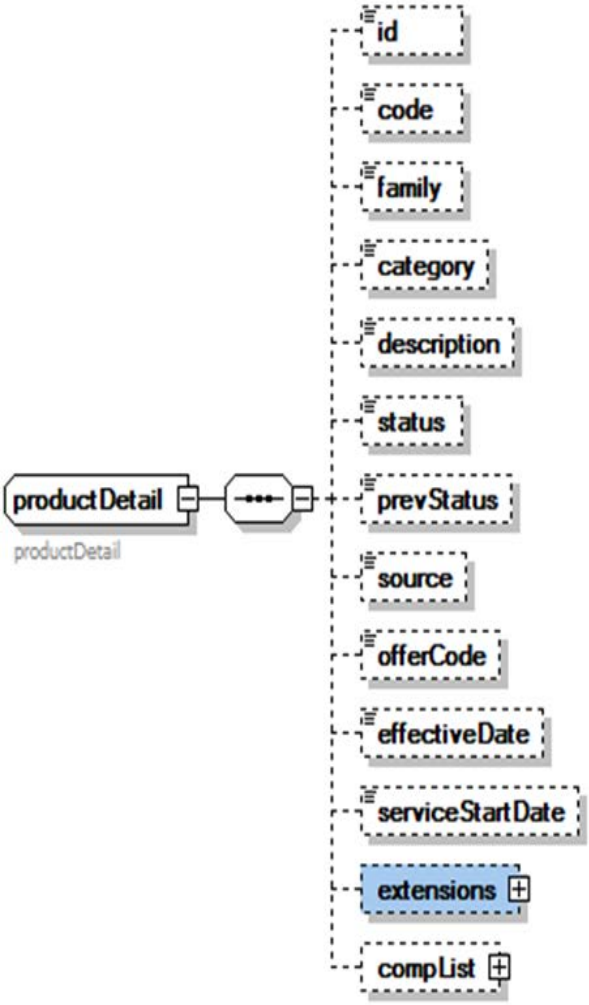
onetimePrice – total one time price of the offer

accessId – the access offer's id, indicating this offer is associated with an access

extensions – a list of name value pairs capturing any attributes populated from user extension of the document data model

product – the product included in the offer, see next section for more details

9.1.5 productDetail



id – product id

code – product's catalog code

family – catalog family values are defined in pfamily code table

category – catalog cateogry values are defined in pcategory code table

source – indicates either "EXT" for existing one or "NEW" for newly added one

prevStatus – the previous status, could be ignored

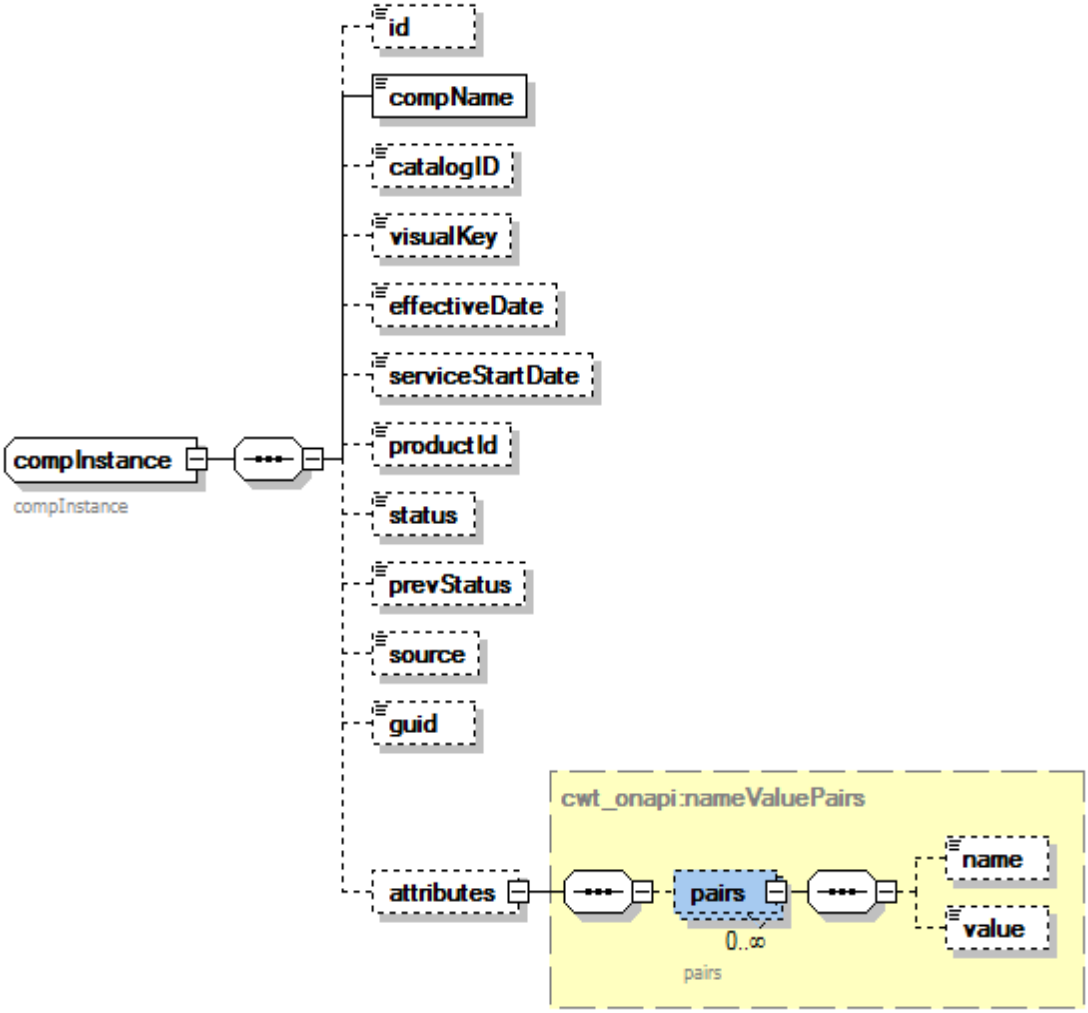
status – item status is defined as a enumeration, consult the data type cwt_do:orderItemStatus

effectiveDate – the tentative date when the product should be effective, it could be either the date the services in the product went active, disconnected or changed.

serviceStartDate – this is the start date of when the offer was in service, it's empty for new ones

compList – the list of components see next section for more details

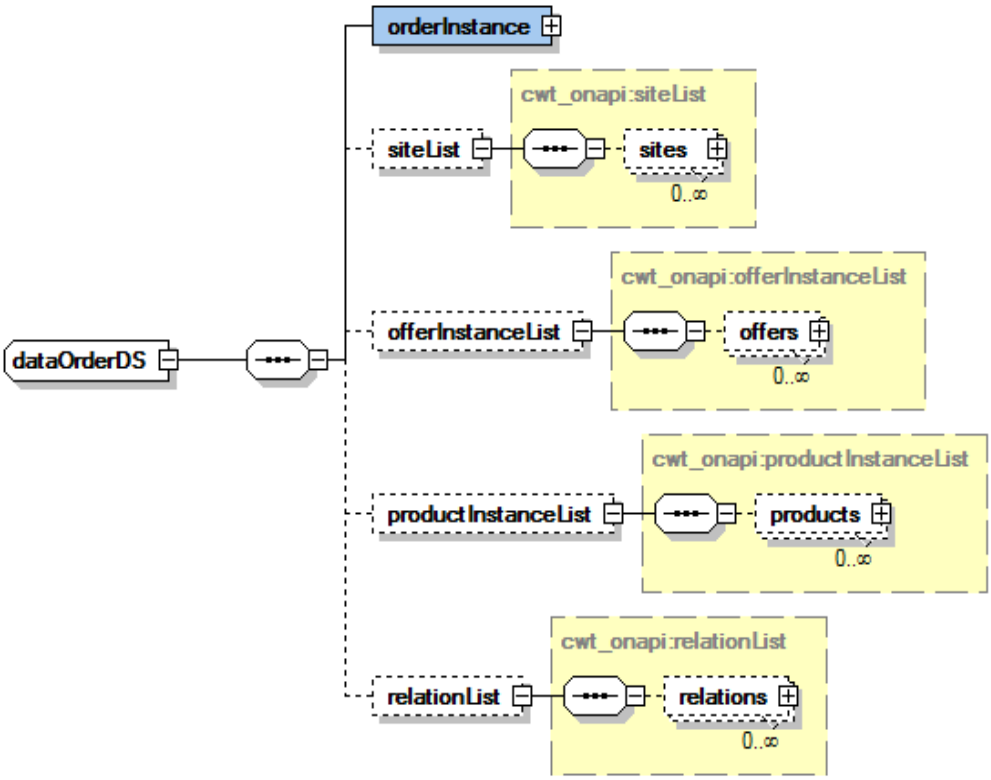
9.1.6 Component Instance



Similar to productDetail and offerDS, the id is the component's id and **compName** is the catalog defined code.

attributes is a list of name value pairs capturing all the attributes within the component.

9.1.7 dataOrderDS



DataOrderDS is the DS equivalence of the internal Data Order. It's very important to fully understand each sub parts to understand the organization of the order when using any direct dataOrder modification calls like **setDataOrder** and **updateDataOrder** operations.

The **orderInstance** DS is the same as the one explained above.

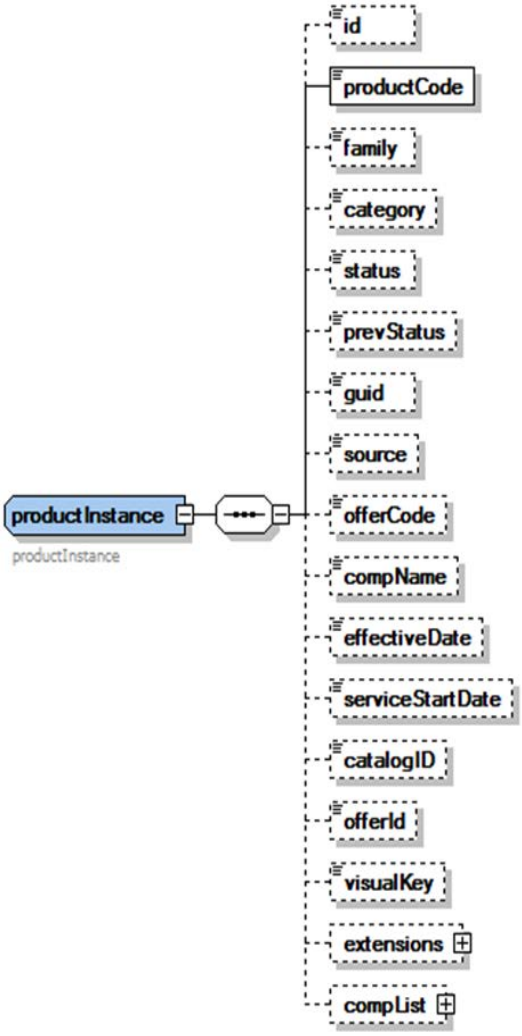
siteList contains a list of siteDetail DS, explained above.

offerInstanceList contains a list of offerInstance DS, explained in above.

productInstanceList contains a list of productInstance DS, explained in above.

relationList contains a list of relationDetail DS, explained in above.

9.1.8 offerInstance



id – product id

productCode – product's catalog code

family – catalog family values are defined in pfamily code table

category – catalog category values are defined in pcategory code table

source – indicates either "EXT" for existing one or "NEW" for newly added one

prevStatus – editable, the previous status, could be ignored

status – editable ,item status is defined as a enumeration, consult the data type cwt_do:orderItemStatus

effectiveDate – editable, the tentative date when the product should be effective, it could be either the date the services in the product went active, disconnected or changed.

serviceStartDate – this is the start date of when the offer was in service, it's empty for new ones

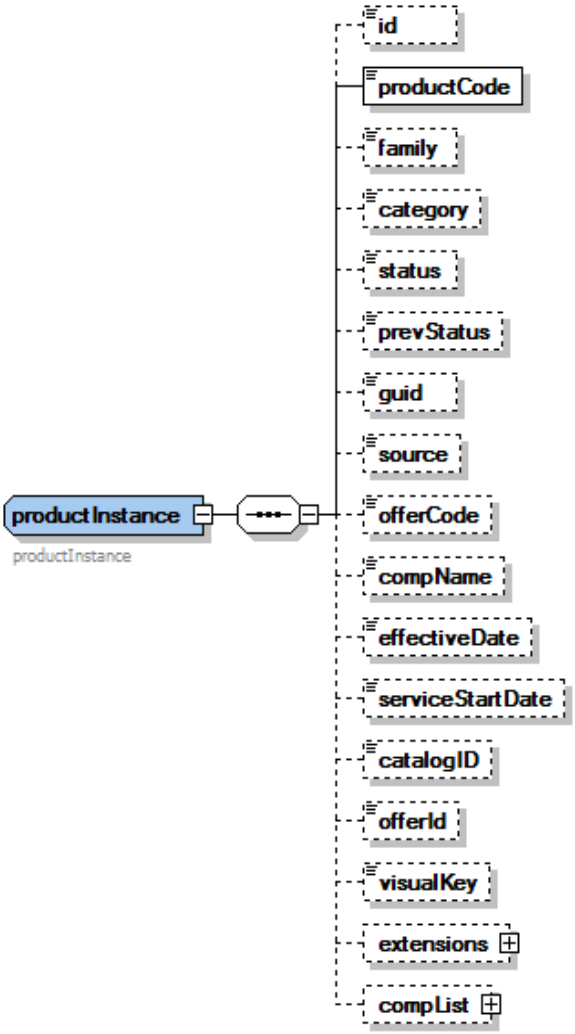
catalogID – read only, the unique catalog documents id

offerId – the offer's id that contains this product

compList – the list of components see section 6 for more details

extensions – a list of name value pairs capturing any attributes populated from user extension of the document data model

9.1.9 productInstance



id – product id

productCode – product's catalog code

family – catalog family values are defined in pfamily code table

category – catalog category values are defined in pcategory code table

source – indicates either "EXT" for existing one or "NEW" for newly added one

prevStatus – editable, the previous status, could be ignored

status – editable ,item status is defined as a enumeration, consult the data type cwt_do:orderItemStatus

effectiveDate – editable, the tentative date when the product should be effective, it could be either the date the services in the product went active, disconnected or changed.

serviceStartDate – this is the start date of when the offer was in service, it's empty for new ones

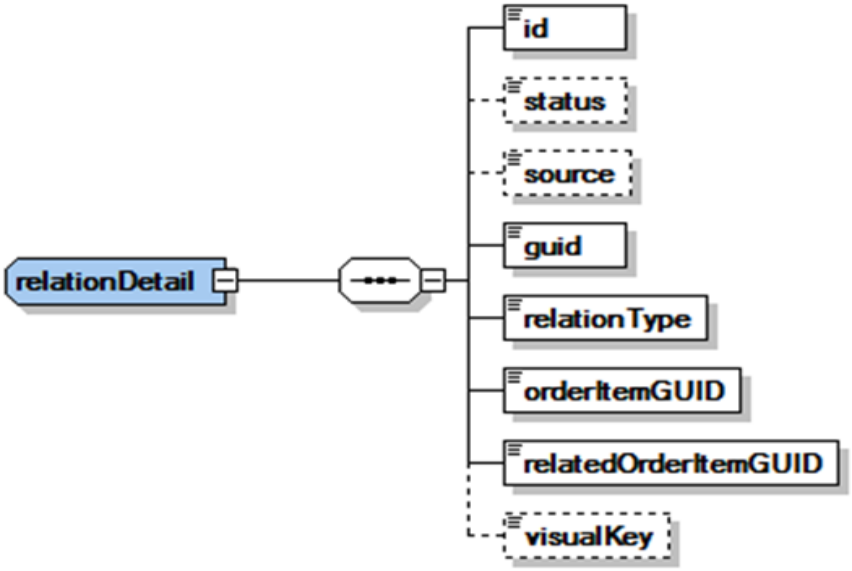
catalogID – read only, the unique catalog documents id

offerId – the offer's id that contains this product

compList – the list of components see section 6 for more details

extensions – a list of name value pairs capturing any attributes populated from user extension of the document data model

9.1.10 relationDetail



relationDetail is a DataStructure that keeps track of the relations between items inside the order. For example, if all the offer OTIPTV with id “abcde” belongs to a customer site with siteId 12345, then the relationDetail DS should set relationType = “SITE2ITEM”, orderItemGUID = “12345”, relatedOrderItemGUID = “abcde” and with a unique guid and a system generated document id.

id – system generated document id

source – indicates either “EXT” for existing one or “NEW” for newly added one

status – editable, item status is defined as a enumeration, consult the data type cwt_do:orderItemStatus

guid – a unique id identifying this particular relation in the SR

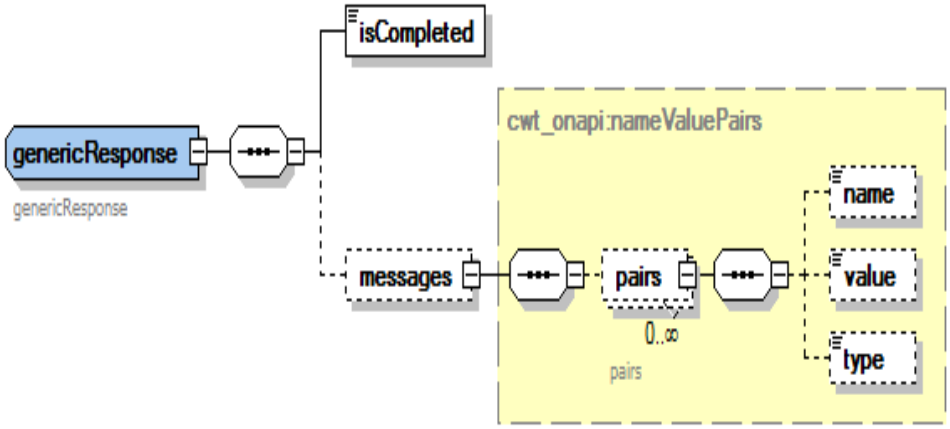
relationType – a enumeration data type defined in cwt_do:relationType

orderItemGUID – the parent item’s GUID in the relation

relatedOrderItemGUID – the child item’s GUID in the relation

visualKey – the string used to display this relation in UI, rarely used

9.1.11 Generic Response



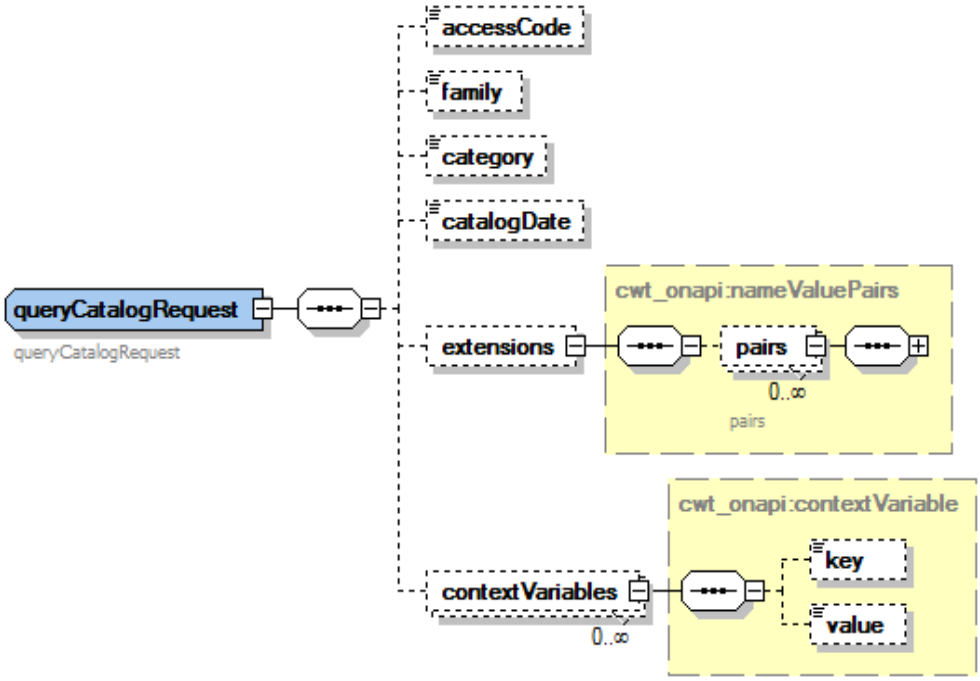
All operation responses are extensions of this genericResponse DS, it captures **isCompleted** to see if the operation finished without any problems and **messages** if there are any error messages or warnings.

isCompleted – takes value 1 for success and 0 for failure

messages – a list of name value pairs capturing any error or warning messages. If messages is not null but isCompleted = 1 then the messages are warnings

9.2 Operations Specification

9.2.1 queryCatalog Operation



This operation queries the current catalog that ON uses for offers that are available for ordering.

accessCode – optionally user can set an access and query the offers belonging to that access. If it's not set the operation will query the entire catalog.

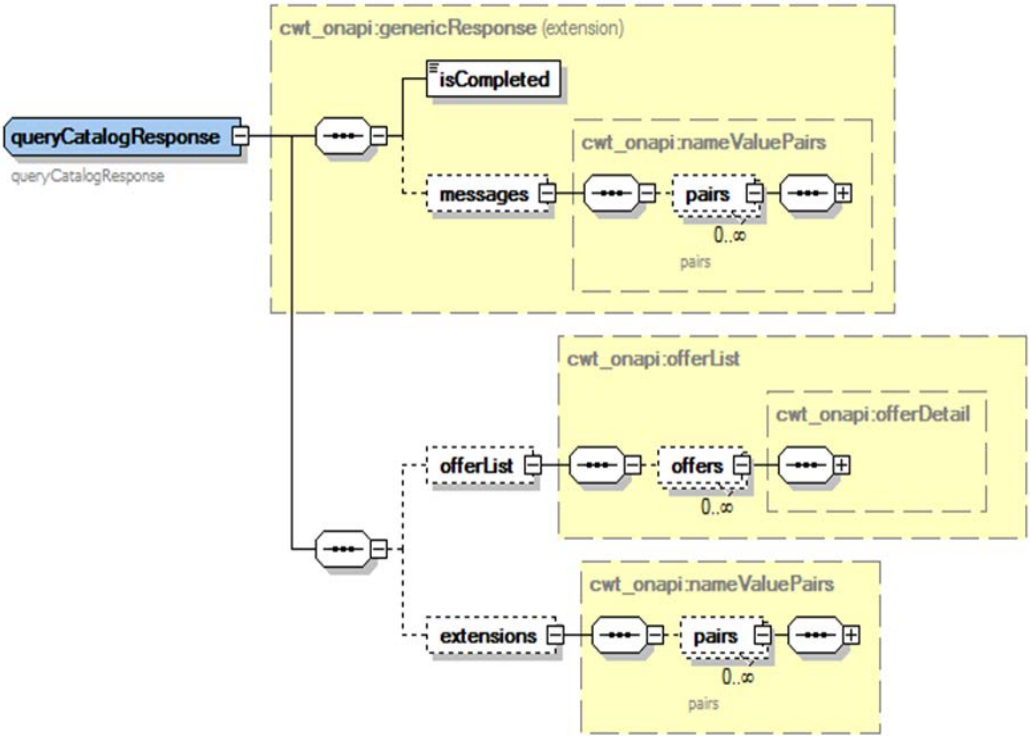
family – specifies the offer family when querying. Catalog offer family values are defined in ofamily code table.

category – specifies the offer category when querying. Catalog offer category values are defined in ocategory code table.

catalogDate – set the current date of the catalog so the result is back dated or future dated. If this field is empty, then the Catalog will use the current date.

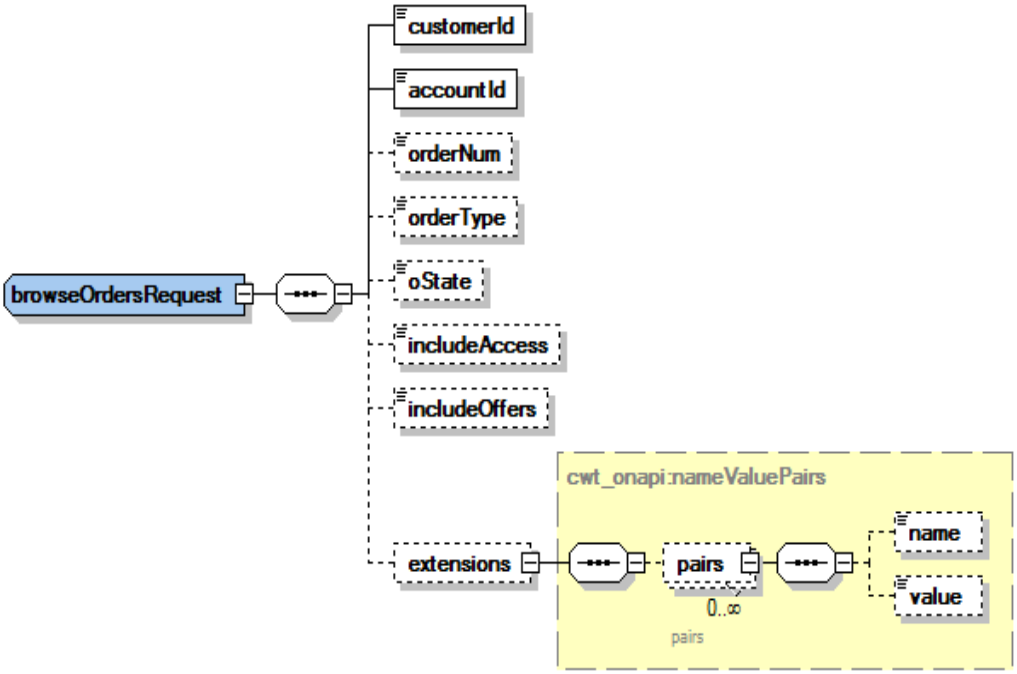
contextVariables – an array of key value pairs passed to catalog. It is typically used during rules evaluation, for example a context variable of name = "CITY", value = "TORONTO" would yield a different set of offers (or different prices) than another city value.

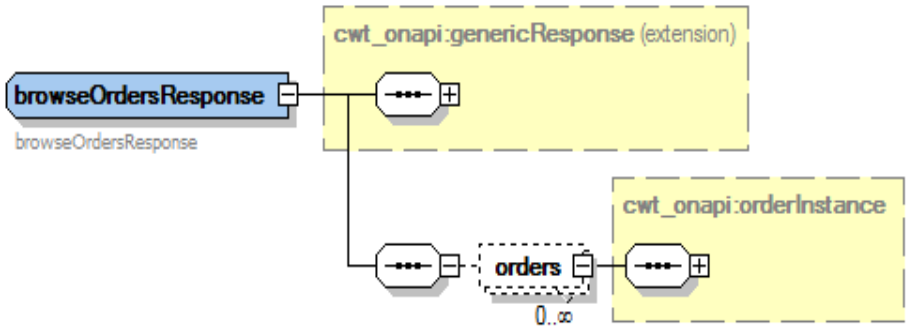
extensions – For user API extension



This operations returns an offerList which is a list of offerDetail DS explained in section 4 of Data Model Specification.

9.2.2 browseOrders Operation





This operation queries the ON database for orders/quotes that were created. It uses the same logic to find orders similarly to the Orders finder on ON UI. It returns an array of orderInstances for faster operation.

customerId – the customerId specified when the order/quote was created

accountId – the accountId specified when the order/quote was created

orderNum – order number of the order/quote, it's meant to be used among CSR and customers and not to be confused with dataOrderId which is the internal dataOrder's id

orderType – current order type are defined as a enumeration of ADD, CHANGE, MOVE, DISCONNECT, MD(Move-Disconnect) and MA (Move-Add)

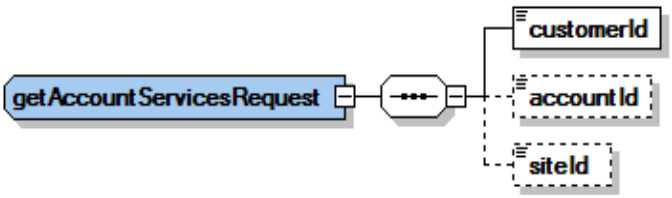
oState – order state, consult data type cwt_do:oState for more info

includeAccess – Boolean flag for returning the list of access offers in the response

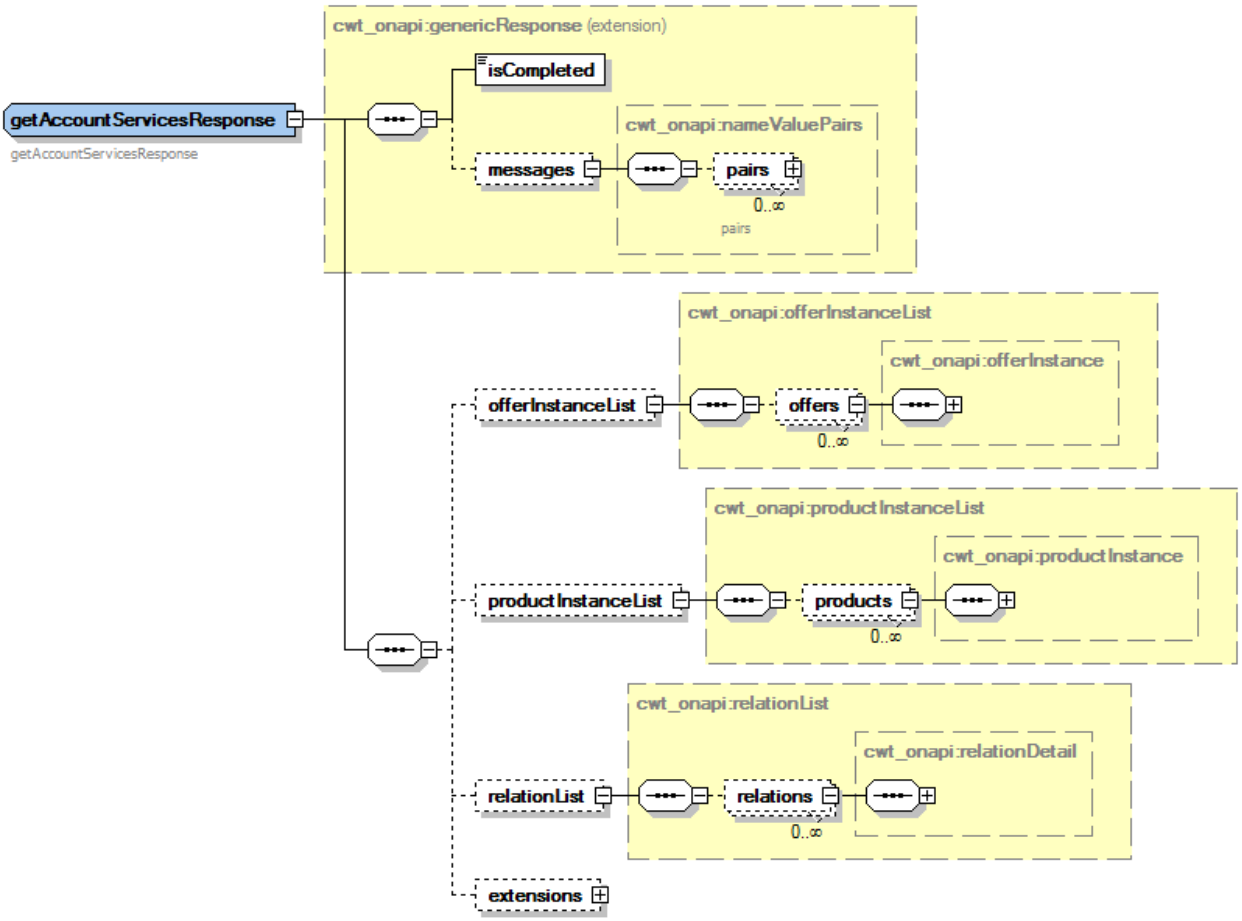
includeOffers – Boolean flag for returning the list of offers in the response

extensions – for user extensions, for example, adding dataOrderId as a search criteria

9.2.3 getAccountServices Operation

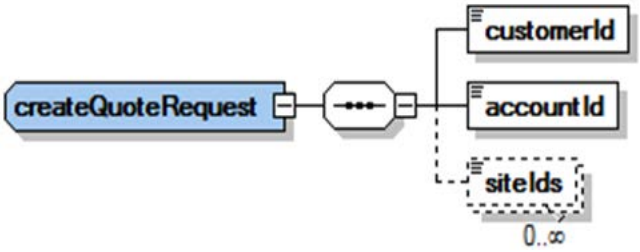


This operation queries the SR (if ON and SR are integrated) and returns the list of offers, products and relations according to the search **criteria customerId, accountId, siteId** in the SR.

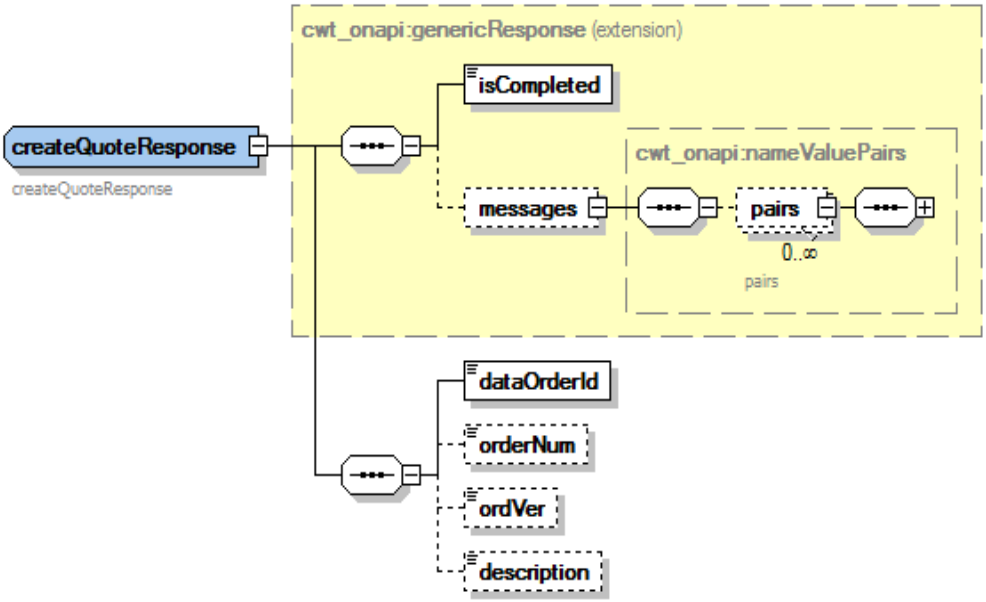


Note that the response structure is very similar to the structure of dataOrderDS, explained in section 7 of Data Model Specification.

9.2.4 createQuote Operation

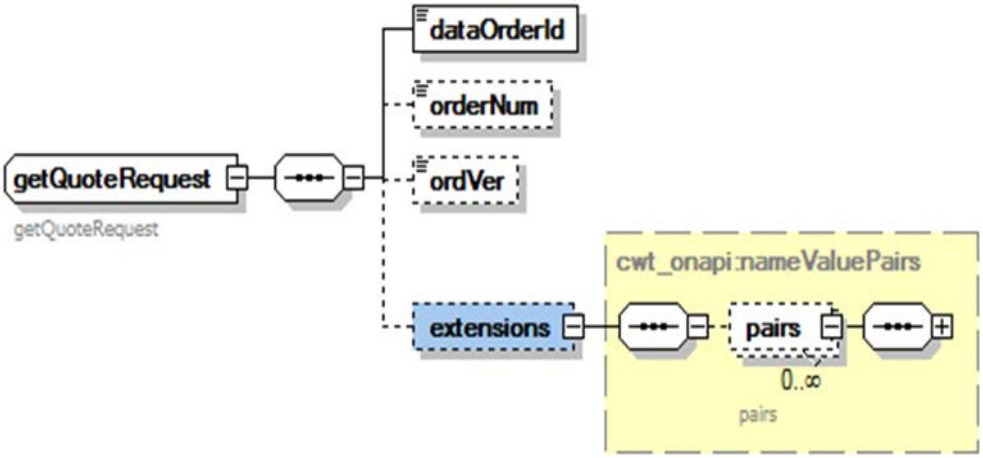


This operation creates an empty quote given the **customerId, accountId** and a list of **siteIds**.



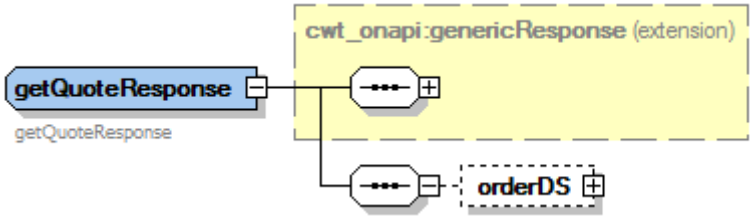
It returns the **dataOrderId** that can be used later to operate the quote. **description** is used to capture the visualKey on the order for UI. **orderNum** and **ordVer** identifies the order when dataOrderId is not available to the interface.

9.2.5 getQuote Operation



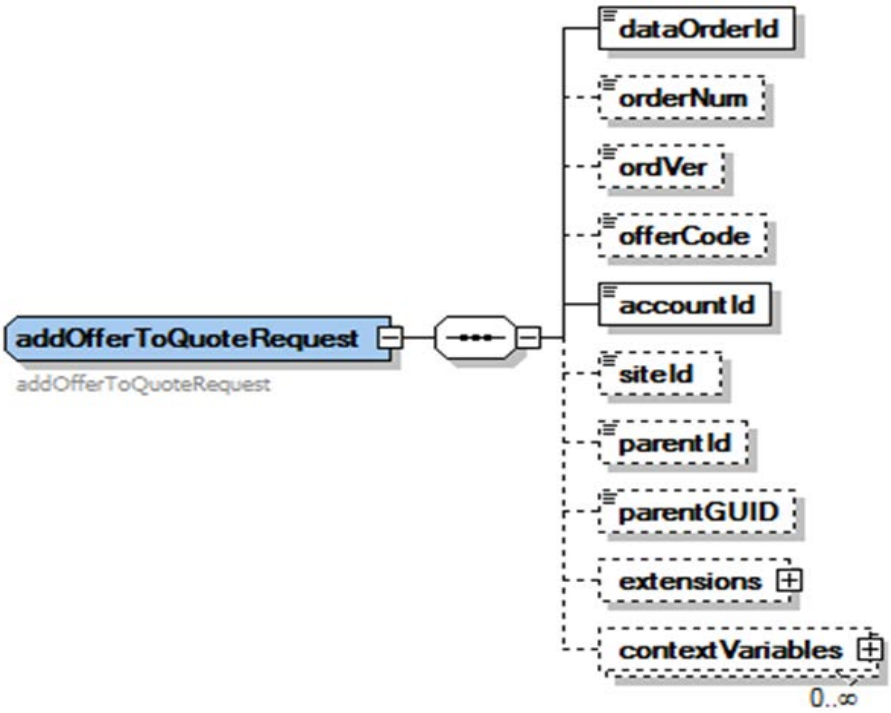
getQuote operation takes the **dataOrderId** and tries to load the orderDS.

extensions is a list of name value pair capturing any user extension, it's been used in many places.



It returns the **orderDS** defined in section 1 of the Data Model Specifications.

9.2.6 addOfferToQuote Operation

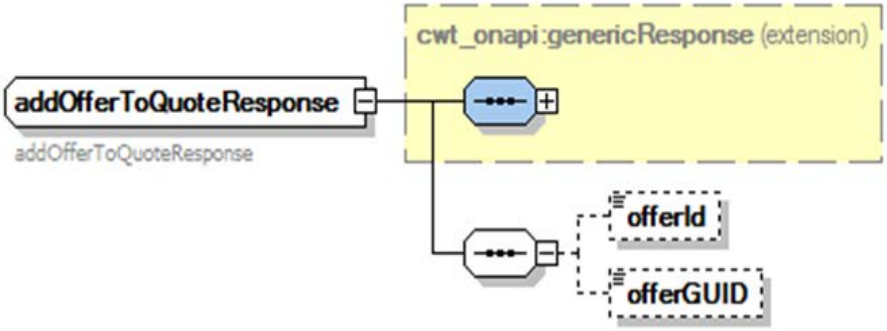


This operation adds an offer (specified by the **offerCode**) to a quote/order (specified by the **dataOrderId** or **orderNum** with **ordVer**)

siteId – the siteId of a customer site where the offer should belong

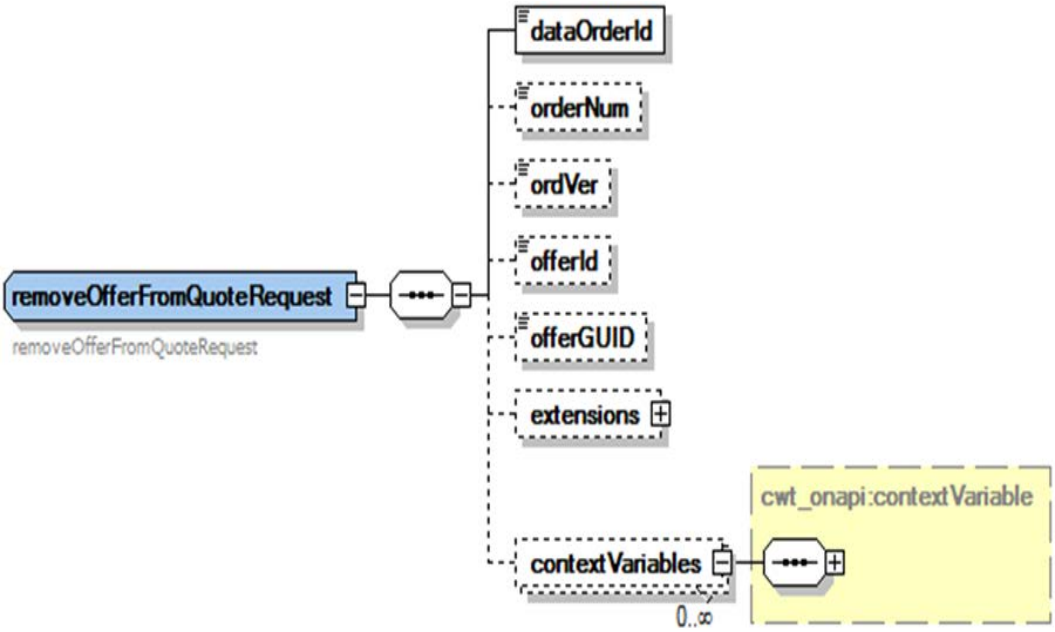
parentId/parentGUID – it normally should be the access offer's ID. If it is null then the offer should be an access or standalone offer that doesn't require an access.

contextVariables – it sets the catalog context while adding the offer to the quote/order. It contributes to the result of catalog rules. It could be null.



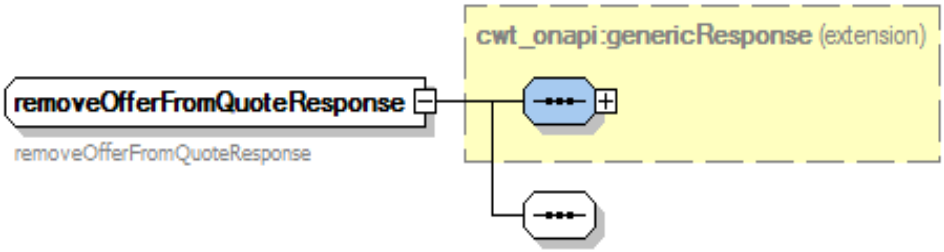
It returns **offerId/offerGUID** which is the offer instance's key after being added to the order.

9.2.7 removeOfferFromQuote Operation



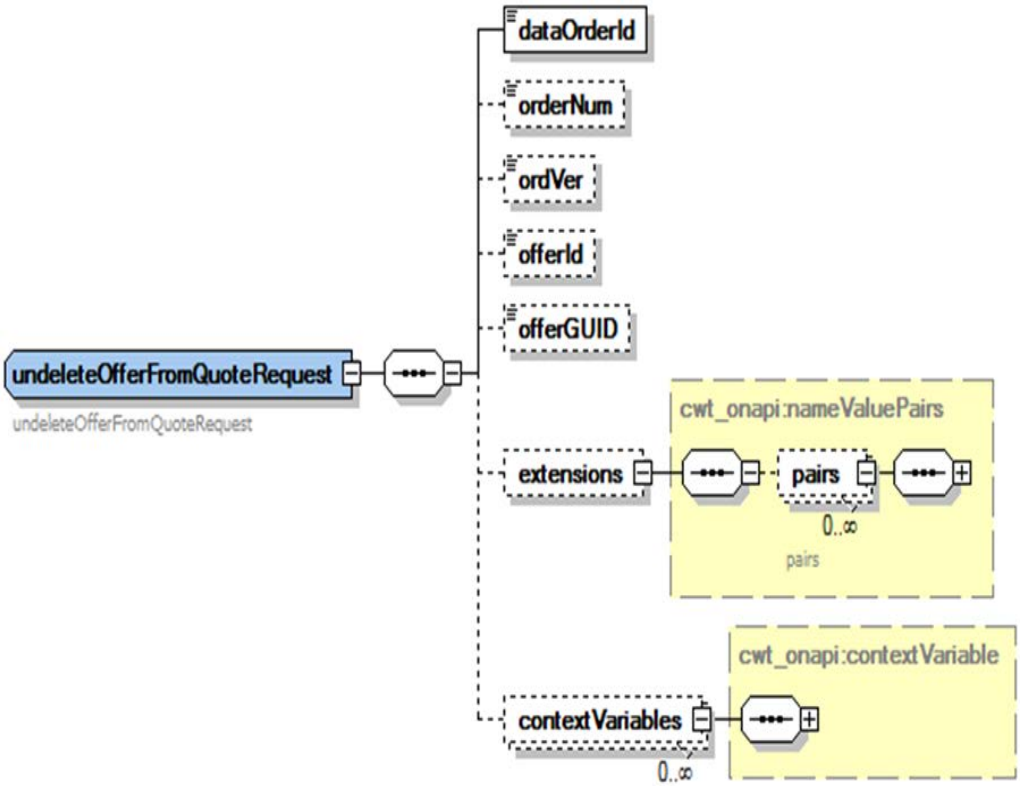
It removes an offer (identified **by offerId/offerGUID**) in the order (identified by **dataOrderId/orderNum+ordVer**)

contextVariables – it sets the catalog context while adding the offer to the quote/order, it contributes to the result of catalog rules, it could be null.



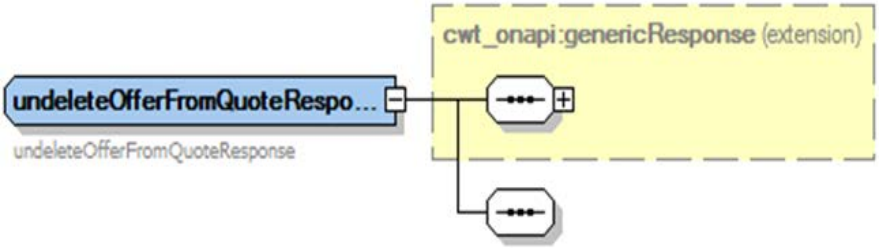
There is no additional information with this call besides what's commonly included in genericResponse.

9.2.8 undeleteOfferFromQuote Operation



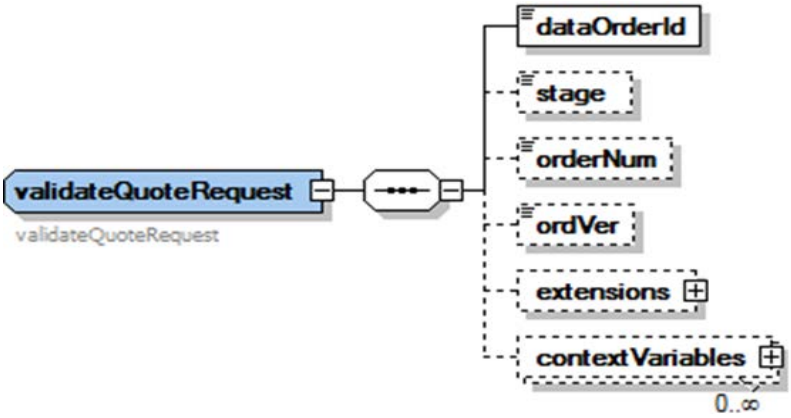
This operation reverts a removed offer (identified by **offerId/offerGUID**) in the order (identified by **dataOrderId/orderNum+ordVer**).

contextVariables – it sets the catalog context while adding the offer to the quote/order. It contributes to the result of catalog rules and it could be null.



There is no additional information with this call besides what's commonly included in genericResponse.

9.2.9 validateQuote Operation

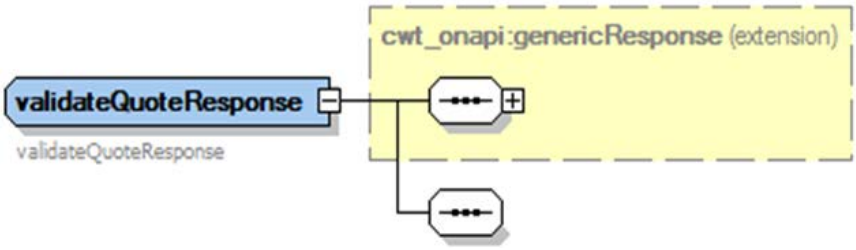


This operation validates to see if the quote passes ON's validation.

dataOrderId – the quote/order's internal ID.

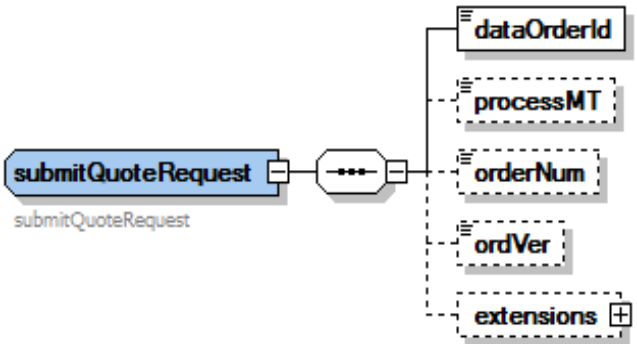
stage – the validation stage of the quote/order, it meant to take the enumeration values: 0 – initial stage, 100 – present stage, 200 – checkout stage, 300 – credit check stage, 400 – payment hold stage and 500 – submit stage. However it's not utilized on ON'S deployment, and user can freely re-define each stage and it's meaning in cwt_on:ClsValidation script

contextVariables – it sets the catalog context while adding the offer to the quote/order, it contributes to the result of catalog rules, it could be null.



There is no additional information with this call besides what's commonly included in genericResponse.

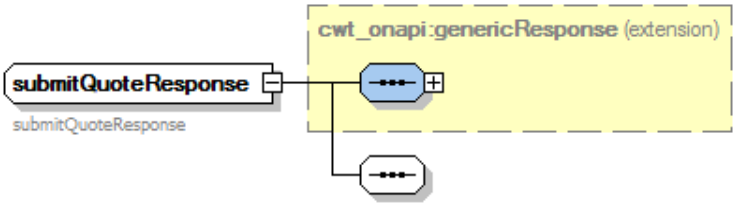
9.2.10 submitQuote Operation



This operation submits the quote to a user process, which decomposes the quote and or provisions it.

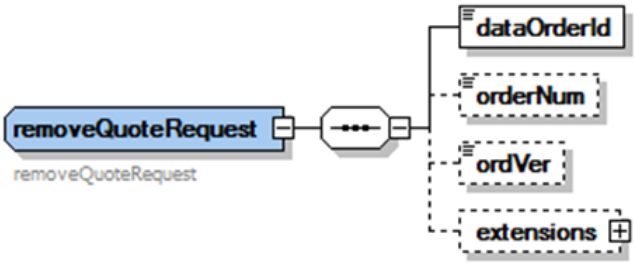
dataOrderId – the quote/order's internal ID.

processMT – the process's fully qualified metadata type string, for example, "orderHandling:newOrderProcess".



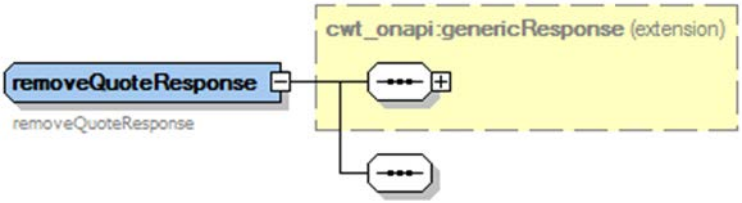
There's no additional information with this call besides what's commonly included in genericResponse.

9.2.11 removeQuote Operation



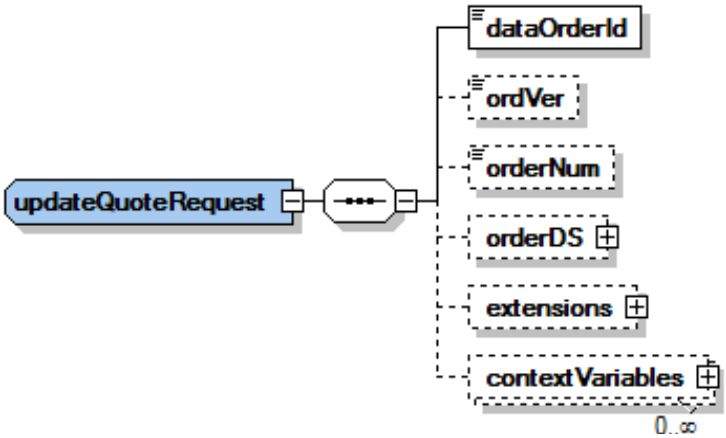
This operation removes the quote/order from the ON Order database. This should only be used when a quote/order has been corrupted.

dataOrderId – the quote/order's internal ID.



There's no additional information with this call besides what's commonly included in genericResponse.

9.2.12 updateQuote Operation

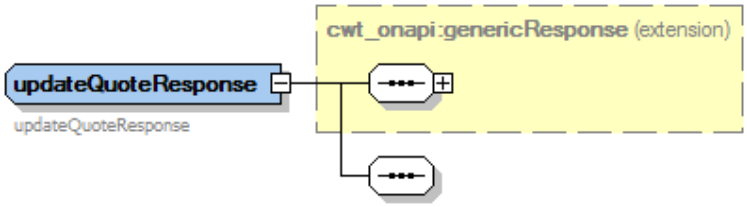


This operation updates the quote/order, for example changing some attribute values in orderInstance or a component.

dataOrderId – the quote/order's internal ID.

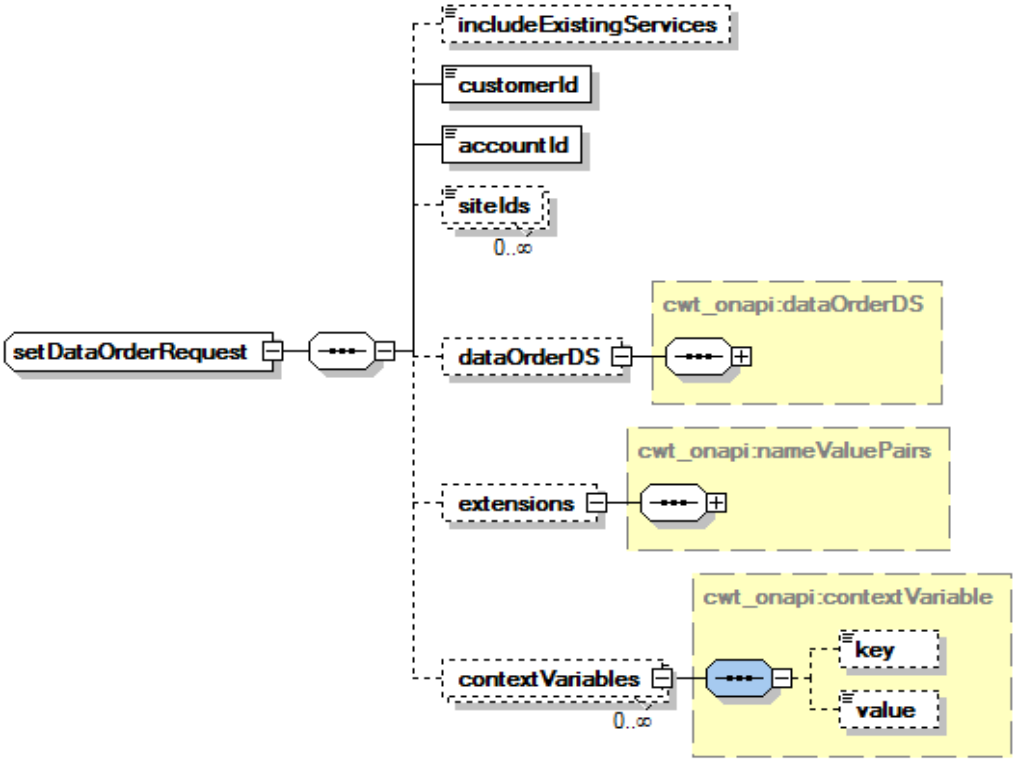
orderDS – the UI order representation explained in section 1 of Data Model Specification

contextVariables – it sets the catalog context while adding the offer to the quote/order. It contributes to the result of catalog rules and it could have a null value.



There's no additional information with this call besides what is commonly included in genericResponse.

9.2.13 setDataOrder (or rather *create*DataOrder) Operation



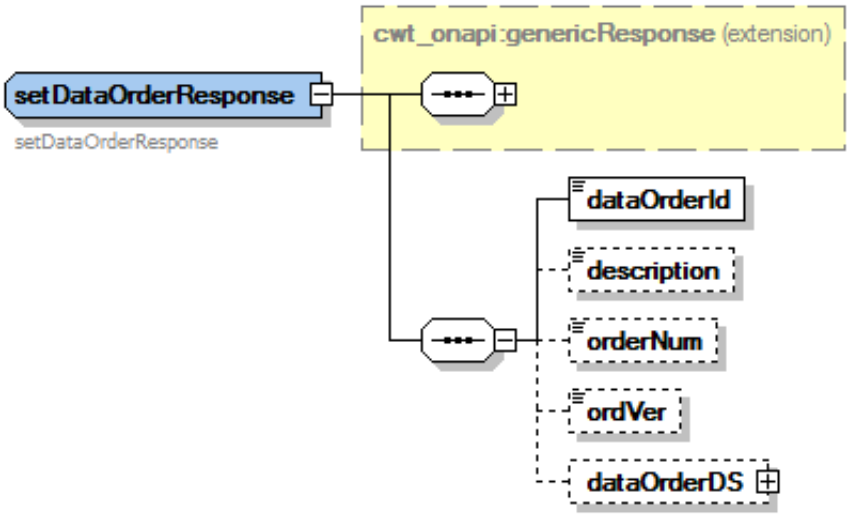
Similar to createQuote operation it takes **customerId, accountId** and an array of **siteIds**. However, this operation directly writes to the internal data order structure. ON won't be able to validate orders created using this method, so user needs to be very careful while populating the values of the data models inside this structure.

includeExistingServices – a Boolean flag takes 1 or 0, if it is set to 1, creating the data order does not load services from the SR it will assume those SR items already exist and populated in the dataOrderDS. If you want to add new services to the order and let ON automatically populates the SR items into the order, then don't set this flag.

dataOrderDS – if it is not null then ON will try to create and map the entire structure to an internal data order. If it is null then an empty data order will be created. Refer to the Data Model Specification section of this document for more information.

extensions - a list of name value pair capturing any user extension. It has been used in many places.

contextVariables – it sets the catalog context while adding the offer to the quote/order and it contributes to the result of catalog rules. Its value can be null.



setDataOrder operation returns the data order that's been created. ON also maps the order back to a **dataOrderDS** structure for user to verify. Check the i**sCompleted** field (included in genericResponse) to make sure it's completed successfully. Error or warning messages may appear in **messages**.

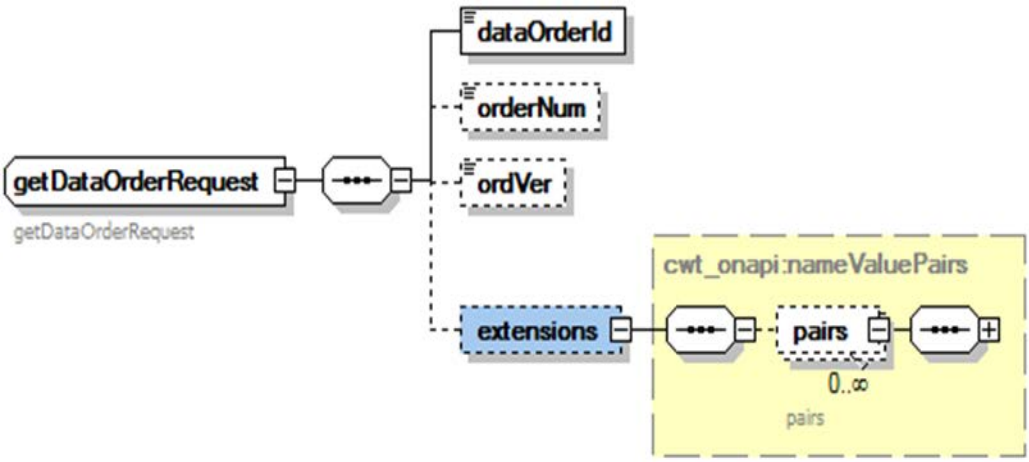
dataOrderId – the created data order's ID

orderNum – the order's number more of a customer-facing ID

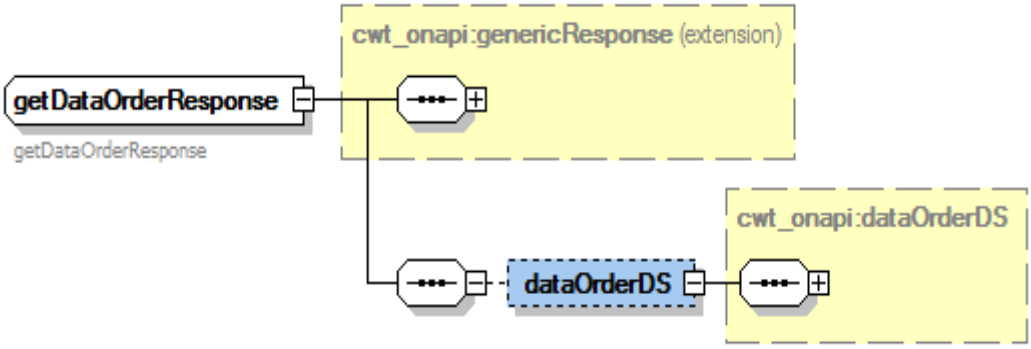
ordVer – the order's version number if there are more orders of the same order number

description – the visual key of the dataOrder

9.2.14 getDataOrder Operation



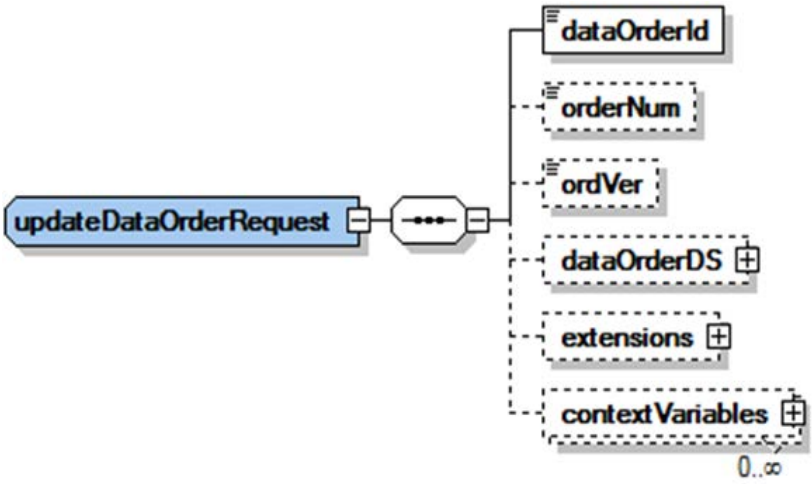
This operation returns the specified dataOrder identified by **dataOrderId**.



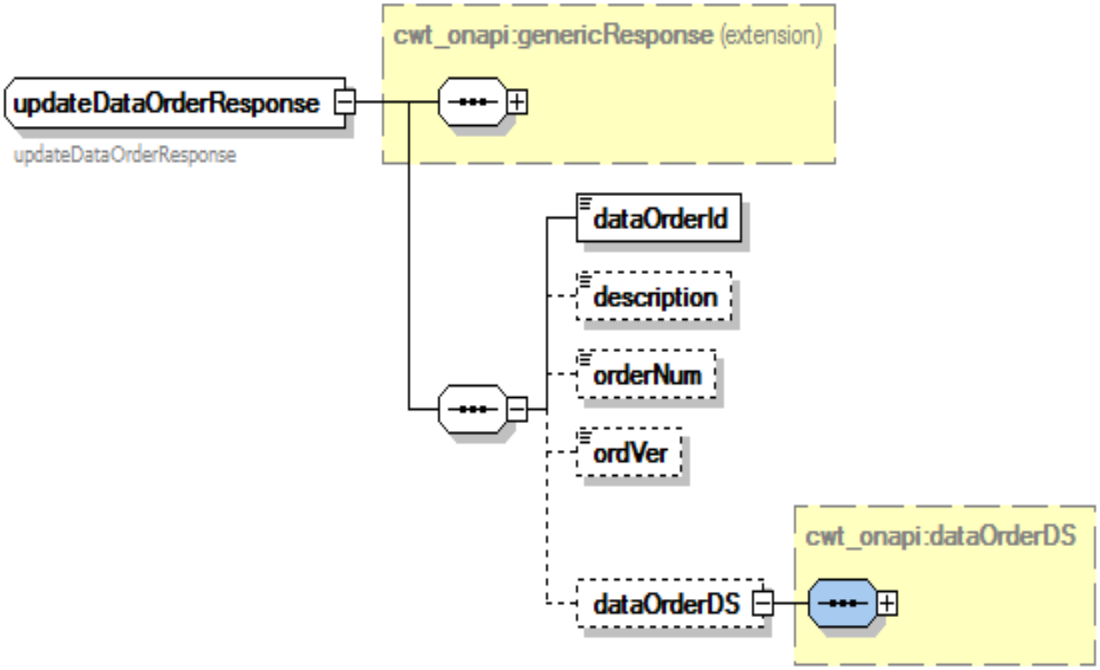
Check the **isCompleted** field (included in genericResponse) to make sure it is completed successfully. Error or warning messages appear in **messages**.

dataOrderDS – returns the full data order requested and mapped in dataOrderDS format.

9.2.15 updateDataOrder Operation



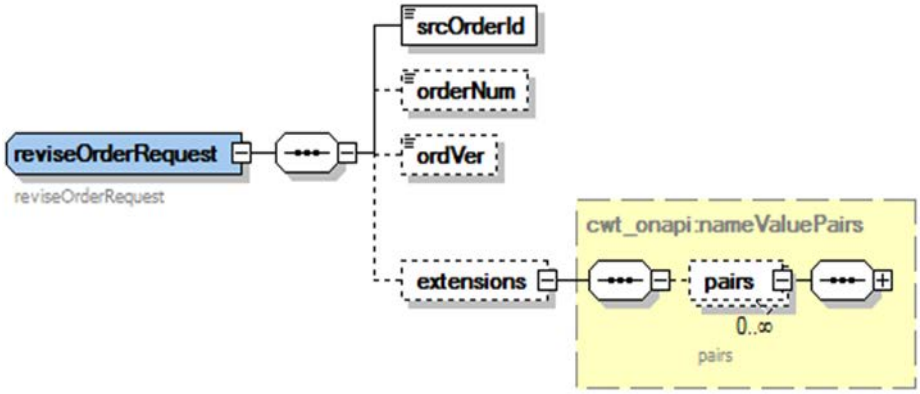
This operation works very similar to setDataOrder. However, it assumes the dataOrder identified by **dataOrderId** or **orderNum** + **ordVer** exists in ON. It does not load items from SR. No validation will be performed. It is recommended that you consult the Data Model Specification sections to learn the possible values and restrictions of each data model.



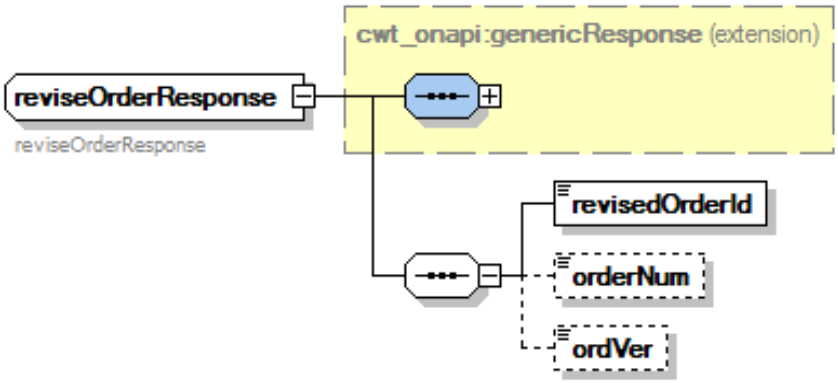
Check the **isCompleted** field (included in genericResponse) to make sure it's completed successfully. Error or warning messages should appear in **messages**.

dataOrderDS – returns the full updated data order mapped in dataOrderDS format

9.2.16 reviseOrder Operation

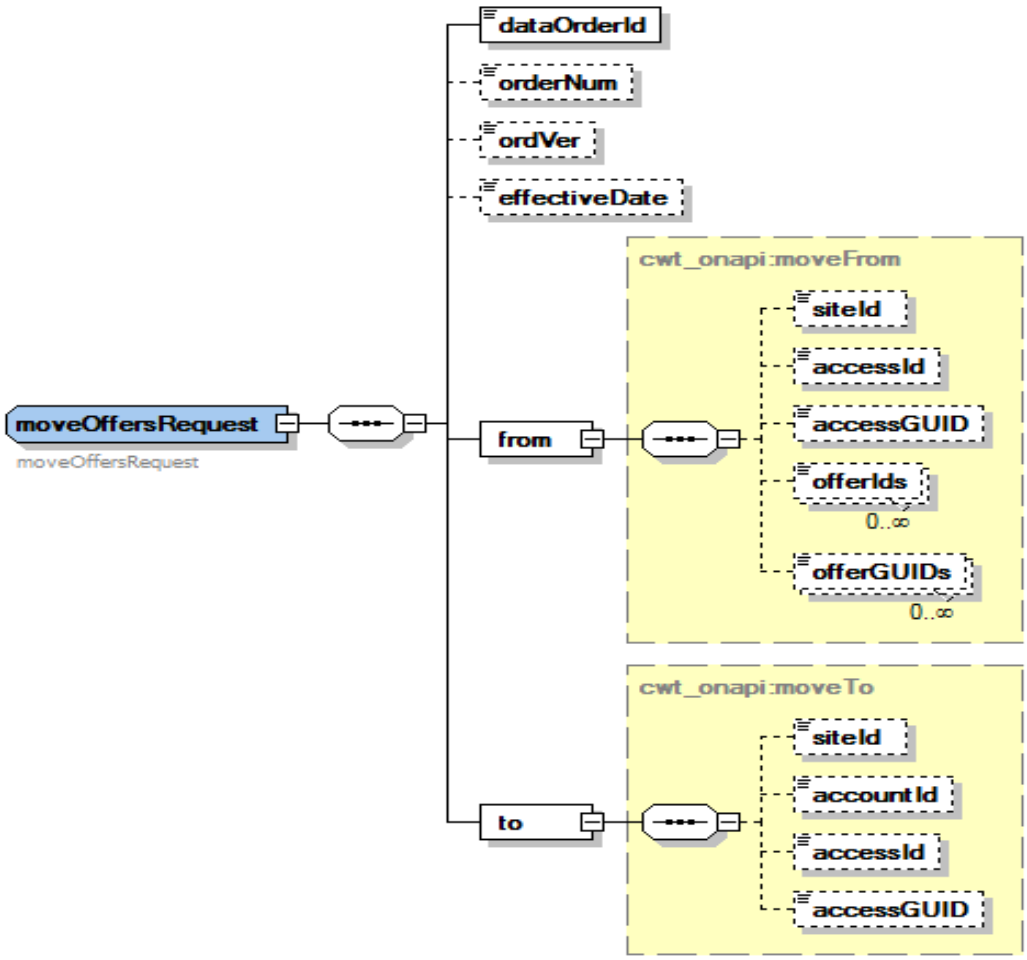


This operation "Revise-Cancel" the quote/order identified by **srcOrderId** and makes a "Draft" state copy so user can modify the content.



It returns the **revisedOrderId** with **orderNum** and **ordVer**, which identifies the "Draft" state copy of the original order. Note that the original order and the revised order should have the **same order number** but **different order version**. The data order ID for each order is always unique.

9.2.17 moveOffers Operation



This is a utility operation and it relocates the specified offers from one site defined in the order (identified by **dataOrderId** or **orderNum** with **ordVer**) to another.

For example, to move:

1 all the offers from siteA to siteB, the request needs to specify

from.siteId **=** siteA.siteId

to.siteId = siteB.siteId

access offer (access1) from siteA to siteB

from.accessId = access1.id or from.accessGUID = access1.guid

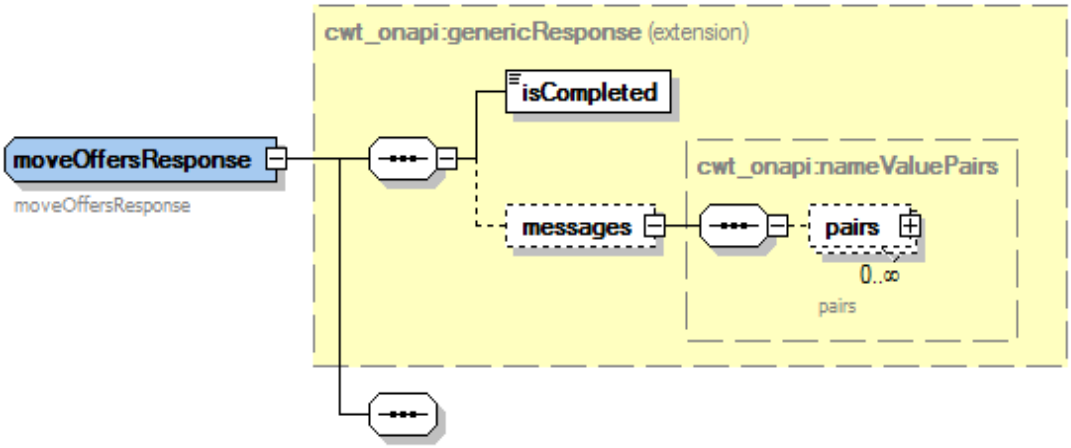
to.siteId = siteB.siteId

to.accountId = <another account>.accountId

a selected list of offers to another access (thus with the access site & account)

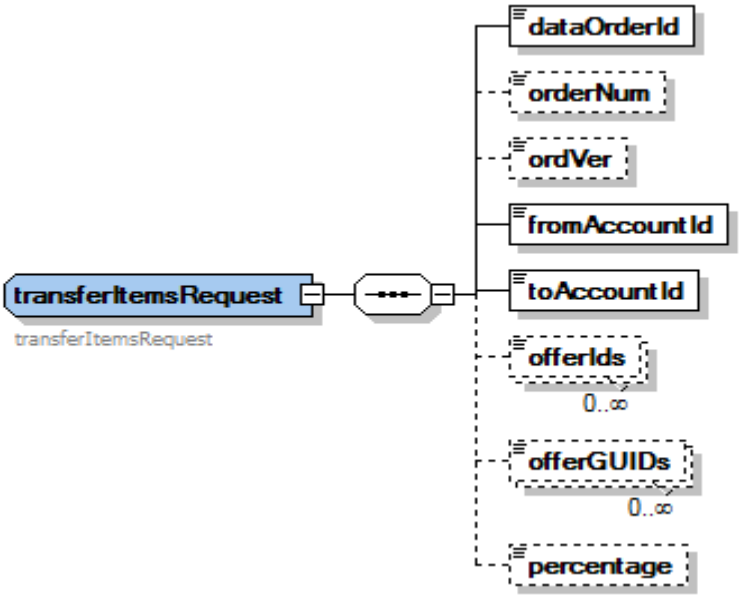
from.offerIds or from.offerGUIDs array with the offers' id/guid

to.accessId or to.accessGUID



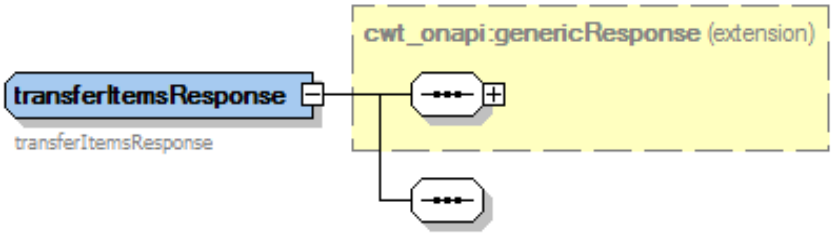
Note that sometimes it might not be eligible to remove or add offers due to rules and restrictions defined in the catalog. Check **isCompleted** to ensure that errors and warning **messages** are available.

9.2.18 transferItems operation



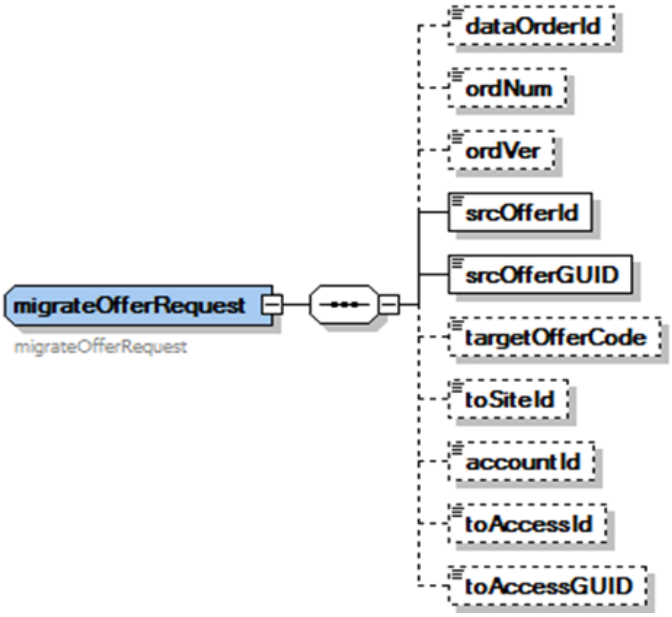
This is a utility operation to transfer ownership (billing/account info) of a list of offers in the order (identified by **dataOrderId** or **orderNum** with **ordVer**) to another.

fromAccountId specifies the source account and toAccountId specifies the destination account.

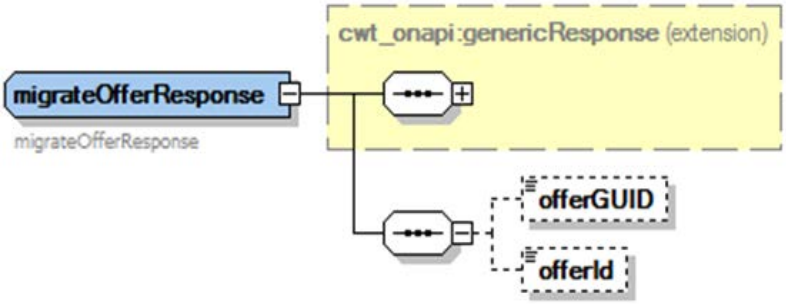


Check **isCompleted** to ensure that errors and warning **messages** are available.

9.2.19 migrateOffer



This is a utility operation to upgrade or downgrade a specific offer (identified by **srcOfferId** or **srcOfferGUID**) in the order (identified by **dataOrderId** or **orderNum** with **ordVer**) to another. A typical use case would be upgrading a plan offer from Basic to Advanced or something similar.



On success, it returns the newly added offer ID and GUID. Note that sometimes it might not be eligible to remove or add offers due to rules and restrictions defined in the catalog. Check **isCompleted** to ensure that errors and warning messages are available.

10 Permissions Control

CW 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 ON.

Privilege ID	Privilege Name	Functions
cwt_wzConfig	CWT - Wizard Application	Allow full access to wizard configuration application
cwt_onEntry	ON - Order Entry	Allow access to Order Negotiations application. Intended to be used by a CSR user.
cwt_onCorrection	ON - Order Correction	Allow access to Order Negotiations application. Intended to be used by a support user.
cwt_onCredit	ON - Credit Analyst	Allow access to see Order Negotiations application. Intended to be used by a credit analyst user.
CWDeveloper	CWDeveloper	Allow full access to all ON's menu objects. Intended to be used by developers to debug issues.

There are more fine grained permission control available by override scripts under CWT - ON UI Order / scripts starting with permission.

11 Reference List

The following is a list of documentation for reference:

- *Customer Information Management User Guide*
- *Ericsson Catalog Manager Installation Guide*
- *Service Registry Configuration Guide*
- *System Administration User Guide*
- *System Configuration User Guide*
- *User Profile Administration User Guide*
- *Velocity Studio User Guide*

12 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.

13 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.