



OCC Reference

Generated on: 2024-12-03 11:16:24 GMT+0000

SAP Commerce | 2205

Public

Original content: https://help.sap.com/docs/SAP_COMMERCE/e5d7cec9064f453b84235dc582b886da?locale=en-US&state=PRODUCTION&version=2205

Warning

This document has been generated from the SAP Help Portal and is an incomplete version of the official SAP product documentation. The information included in custom documentation may not reflect the arrangement of topics in the SAP Help Portal, and may be missing important aspects and/or correlations to other topics. For this reason, it is not for productive use.

For more information, please visit the <https://help.sap.com/docs/disclaimer>.

OCC Extension Architecture

OCC Extensions allow you to extend the functionality of OCC without the process of addon installation.

OCC extensions depend on `commerceservices`, and many existing OCC addons have corresponding OCC extensions with the same business logic and functionality. OCC extension names end with `occ`, such as `b2bocc`, for example.

Additional features of OCC extensions include the following:

- OCC extensions don't require any additional commands for installation. They are imported into the Spring web context of `commerceservices`. There's no need to copy files there.
- OCC extensions have a different directory structure to OCC addons, despite having the same logic as an OCC addon.

Directory Structure of an OCC Extension

An OCC extension has a standard extension directory structure, but without the web part, as it's not a web extension. It contains the same files as an OCC addon, but they are positioned differently in the directory structure. The main changes are listed in the following table:

OCC addon directory structure (old)	OCC extension directory structure (new)	Description
<code>/acceleratoraddon</code>		There is no <code>acceleratoraddon</code> directory mainly migrates to the new <code>/resources</code> directory of OCC extension with new OCC extension name. For example, <code>/resources/xyzocc</code> .
<code>/acceleratoraddon/web/src</code>	<code>/src</code>	REST Controller classes and other related classes of OCC extension.
<code>/acceleratoraddon/web/webroot/WEB-INF/messages</code>	<code>/resources/occ/v2/<extension_name>/messages</code>	Localized messages of an OCC extension.
<code>/acceleratoraddon/web/webroot/WEB-INF/lib</code>	<code>/lib</code>	There is no <code>WEB-INF/lib</code> directory in <code>C:\main\lib</code> directory. The <code>/lib</code> directory is located in <code>/acceleratoraddon/web/webroot</code> .
<code>/resources/<addon_name>/import</code>	<code>/resources/impex</code>	ImpEx files in OCC extension aren't loaded by the configuration convention. For more information, see Project Data .
<code>/resources/<addon_name>/web/spring</code>	<code>/resources/occ/v2/<extension_name>/web/spring</code>	Spring bean definitions of OCC extension. The <code>/resources/occ/v2/<extension_name>/web-spring.xml</code> file are imported into the context.

OCC Extensions Are Not Web Extensions

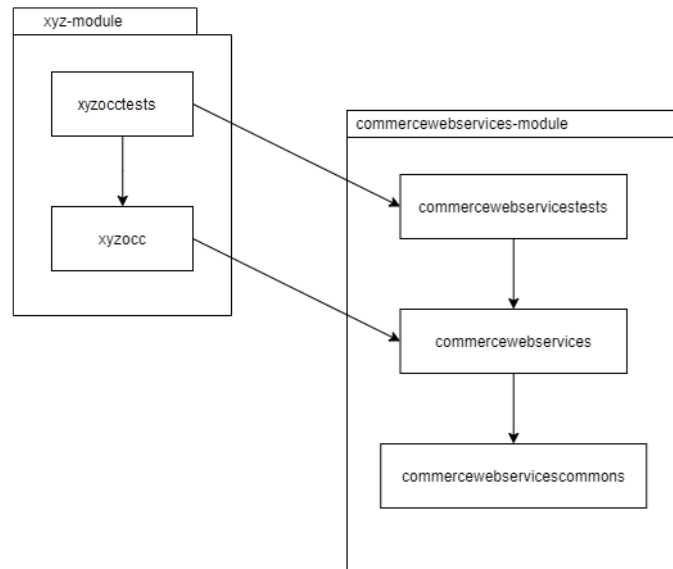
An OCC extension isn't a web extension and has no web context. It extends the functionality of OCC by adding new REST controllers, Spring beans, and localized messages. REST controller classes and other OCC extension classes are located in the main `/src` directory. In the Spring XML configuration of the `commerceservices` extension, there is an import statement that loads beans definitions from all OCC extensions.

```
<import resource="classpath*:occ/v2/*occ/web/spring/*-web-spring.xml"/>
```

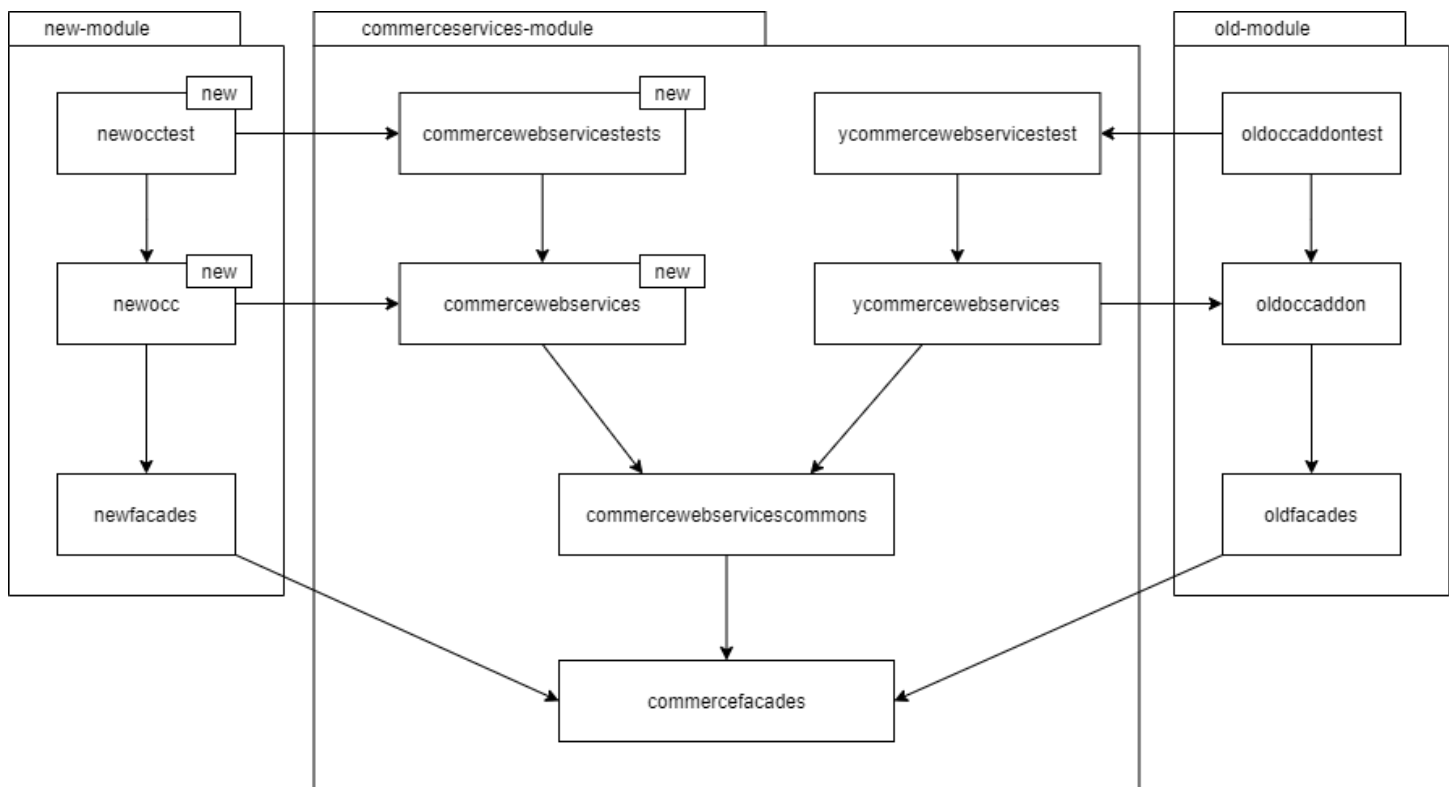
The XML configuration file import statement containing bean definitions must meet the following requirements:

- The configuration file must be located in the `/resources/occ/v2/<*occ>/web/spring` directory, where `<*occ>` points to a directory with the same name as the OCC extension that ends in `occ`, for example `xyzocc`.
- The name of the configuration file must end in `-web-spring.xml`.

Dependencies Between Extensions



The example xyzocc OCC extension depends on commercewebservices. OCC extension classes are located in the `/src` directory and can't inherit from classes in the `/web/src` directory of commercewebservices. Compared to an OCC addon the dependency is reversed. An OCC addon, by contrast, depends on ycommercewebservices because the OCC addon web source files are copied into the ycommercewebservices web source directory.



Overlapping Paths

When an OCC addon installation is used, it's possible to override controller endpoints. By using the `@RequestMappingOverride`, a controller endpoint of an addon can override the endpoints of other extensions. For more information, see [Extending the REST API](#).

For an OCC extension, however, all controller endpoints must have unique paths. Any endpoints that are in conflict with existing endpoints must be changed to have a unique path. The configuration property `occ.rewrite.overlapping.paths.enabled` can be used as a solution. When this property is set to `true`, the specified extension has to provide a unique path for its controller endpoints, that can be in conflict with endpoint in other extensions. For example: `/<baseSiteId>/users/<userId>/orders` becomes `/<baseSiteId>/orgUsers/<userId>/orders`

Site Channel Restrictions

There is a restriction that prevents inappropriate access from one site to the endpoints of another site. For example, it can prevent calling B2B endpoints in context of B2C sites. You can apply the restriction on controller endpoints by using the annotation. This annotation contains a configuration property used to read a set of allowed site channels for this endpoint.

OCC AddOn Converter

OCC AddOn Converter is a command-line interface tool that allows you to convert OCC AddOns to OCC Extensions.

The OCC Extensions Architecture allows you to extend the functionality of OCC without installing any OCC AddOns. As a result, OCC AddOns need to be converted to OCC Extensions. For more information, see [OCC Extension Architecture](#).

Using OCC AddOn Converter

Before running the script, make sure to install Java. For further information on Java version, see [Before You Start](#).

To use OCC AddOn Converter, navigate to the <HYBRIS_HOME>/build-tools/occ-addon-converter directory. You can use the script directly from the source code or run it as a packed JAR file. See the following instructions on how to run the script:

Using OCC AddOn Converter as a JAR File

Run the following commands:

- For Microsoft Windows systems:

```
gradlew.bat jar;

java -jar build\libs\occ-addon-converter.jar ^
--addondir "C:\Projects\xyoccaddon" ^
--extdir "C:\Projects\xyzocc" ^
--stepsdir "C:\Projects\customSteps" & :: stepsdir is optional
```

- For Unix-related systems (such as Linux or Mac OS):

```
./gradlew jar;

java -jar build/libs/occ-addon-converter.jar \
--addondir "/Projects/xyoccaddon" \
--extdir "/Projects/xyzocc" \
--stepsdir "/Projects/customSteps" # stepsdir is optional
```

Using AddOn Converter from the Source Code

Run the following commands:

- For Microsoft Windows systems:

```
gradlew.bat run --args="--addondir ""C:\Projects\xyoccaddon"" --extdir ""C:\Projects\xyzocc"" --stepsdir "
```

- For Unix-related systems (such as Linux or Mac OS):

```
./gradlew run --args="--addondir \" /Projects/xyoccaddon\" --extdir \" /Projects/xyzocc\" --stepsdir \" /Proj
```

Following is an explanation of the parameters:

- xyoccaddon specifies the name of the OCC AddOn.
- xyzocc specifies the name of the new OCC Extension. The names of OCC Extensions need to end with "occ".
- addondir is a parameter that specifies the directory of the OCC AddOn.
- extdir is a parameter that specifies the directory where the new OCC Extension is located after the conversion. The directory is erased every time the converter is used.

- `stepsdir` is an optional parameter that specifies the directory with custom steps that you can provide to apply additional changes.

Script Architecture


OCC AddOn Converter loads and runs Groovy scripts that carry out the conversion. The conversion steps include moving, renaming, and editing files. The steps are parsed without the need to stop the server. If a conversion of a given AddOn requires additional steps, use the `--stepsdir` flag to specify the directory with your custom steps. When the script is run, the filenames of the default steps and custom steps are logged.

Filename Convention

OCC AddOn Converter expects step filenames to start with `Step_`. Number your steps according to the order in which you want them to run. Regardless of whether the step is in the main script or your custom steps, the script orders them by their filenames so that `Step_15_Foo.groovy` is run between the steps `Step_10*` and `Step_20*`.

Writing a Custom Step

The Groovy binding mechanism in OCC AddOn Converter injects the following two implicitly defined variables at run time:

- `<ant>` is an instance of the `AntBuilder` object. For more information on using `AntBuilder`, see [Scripting Ant tasks](#) .
- `<ctx>` is a context object with information about a given operation.

The `<ctx>` variable consists of the following elements:

- `<ctx.ADDON_NAME>` is a String with the OCC AddOn name that is being converted.
- `<ctx.EXTENSION_NAME>` is a String with the target OCC Extension name.
- `<ctx.ADDON_DIRECTORY>` is a String with the directory of the AddOn.
- `<ctx.EXTENSION_DIRECTORY>` is a String with the target directory of the new OCC Extension.

For example, if you want your new OCC Extension to reference packages from a different module, use the following step to change the import settings:

```
ant.echo("Converting: $ctx.ADDON_NAME to $ctx.EXTENSION_NAME")
ant.replace(dir: ctx.EXTENSION_DIRECTORY, token: 'import com.example.old.occ.module', value: 'import com.example.new.occ.module')
```

If you add this code to a file that is located in the directory referenced by the `--stepsdir` flag, OCC AddOn Converter replaces `import com.example.old.occ.module` with `import com.example.new.occ.module` in all the relevant files of the module.

After Conversion

When you generate a new OCC Extension, it's recommended to review the output and run tests against the new extension. The extension has to be used with the Commerce Web Services module. You can't use the newly generated OCC Extension with either the `ycommercewebservices` extension or OCC AddOn that the extension was generated from.

OCC AddOns Architecture

OCC is a set of extensions providing commerce-driven RESTful web services. OCC addons extend OCC with new functionality.

Caution

This page refers to software that has been deprecated as part of the Accelerator UI and older OCC template extensions deprecation. For more information, see [Deprecation of Accelerator UIs and Older OCC Template Extensions](#).

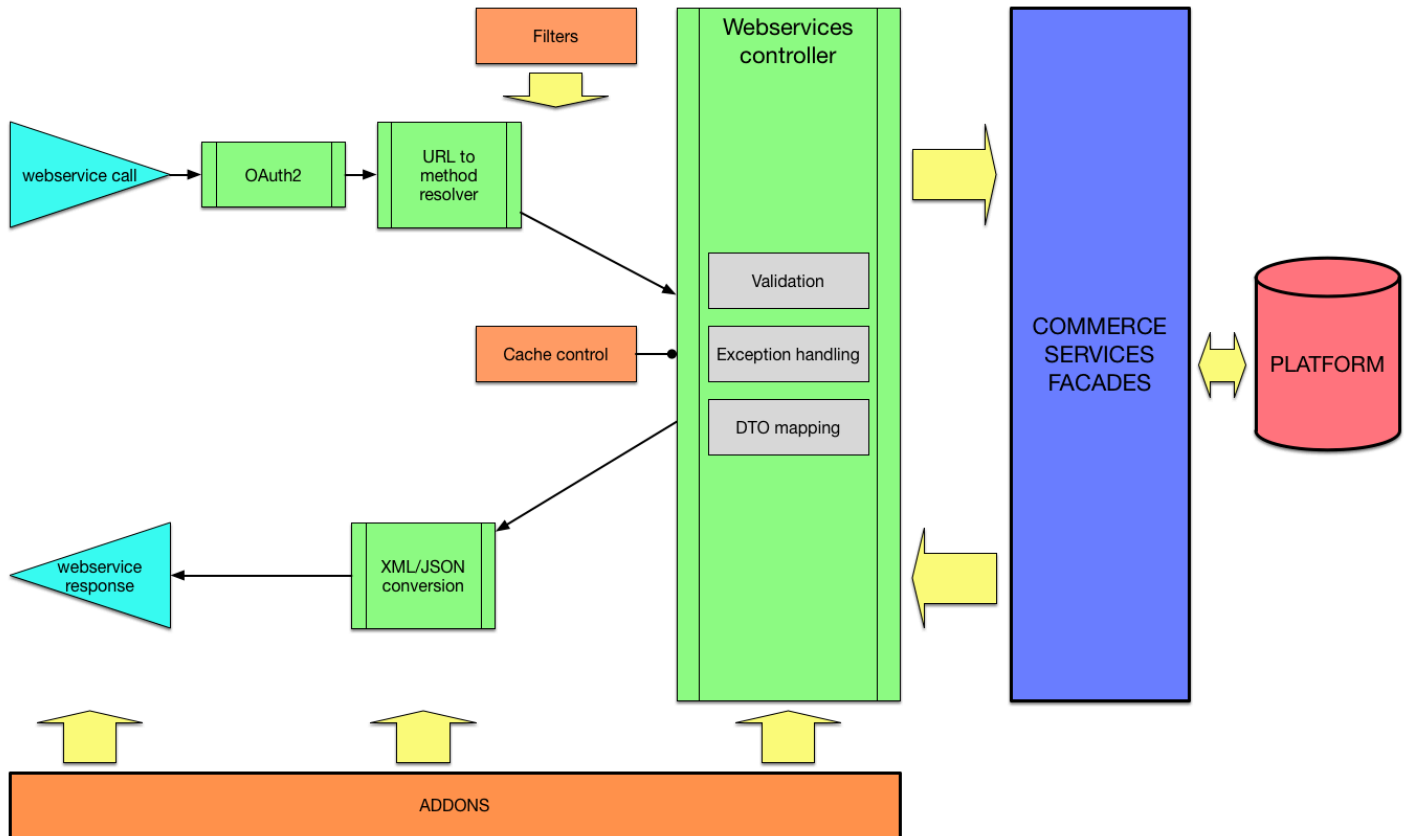
OCC provides a consistent way for web services to communicate with the platform commerce layer. Its main purpose is to expose existing commerce layer functionality to the world of web services. OCC web services are as transparent as possible, but contain some additional elements such as a separate cache, exception handling mechanism, authentication framework, and the attribute mapper.

Commerce Services Extensions

The Commerce Services module contains following extensions:

- `commerceservicescommons` extension
- `ycommerceservices` extension
- `ycommerceservicetest` extension

Some of these extensions are templates which can be adjusted to specific requirements. The `ycommerceservicesaddons` folder contains the `acceleratorwebservicesaddon` AddOn. This specific AddOn depends on the `accelatorservices` extension and is required to ensure advanced payment functionality using CIS services.



ycommerceservices Extension

This is the main template extension of the OCC module. The most important part of this extension is advanced webservises application build on the Spring MVC framework. The calls to the specific resources are executed by a method using a request to the controller. A standard flow is as follows:

1. The request comes to commerce web services controller and is in most cases passed directly to commerce facades (sometimes additional validation is required). It supports the following types of requests:
 - **GET** - a request for data which triggers facade methods that look for and retrieve proper information.
 - **POST, PUT, PATCH** - requests for creating and updating items which can be send either as separate URL parameters or with the use of RequestBody approach. The create/update requests are usually additionally validated in the OCC module.
 - **HEAD** - a request that can be used to retrieve only the number of requested items information - this number is set in the response header.
2. Once the data object is retrieved from the commerce facades, it is converted to predefined DTOs (this way web services are isolated from data object changes in the commerce layer).
3. Data objects then send a response to the call in XML or JSON format (assuming there is no errors or exceptions).

The extension also has some additional features.

The local cache is build on the top of Ehcache. The cache is enabled on specific controller calls - the ones that are cached can be found in the `de.hybris.platform.ycommerceservices.v2.helper` package as helper classes and are marked with `@Cacheable` annotation. The cache configuration parameters can be found in `ehcache.xml` file. Where it was possible the cache is enabled directly on the controller

methods. Additionally many of controller calls has `@CacheControl` annotation. It is the directive in the response header that is consumed by the client or proxy server and controls how the returned data should be cached.

The `ycommercewebservices` extension uses authentication mechanism based on the OAuth2 framework solution implemented in the platform. For further information on OAuth2 see [OAuth 2.0](#).

Additionally there are several filters and interceptors that are used for different purposes like cache control, base site and customer verification, setting up request context and session attributes. Although the Version 2 of `commercewebservices` is stateless, in the scope of a single request there is still a session created and used underneath.

`commercewebservicescommons` Extension

The `commercewebservicescommons` extension is the only extension in the OCC module that **is not a template**. This extension contains web services cache control, data mapping, errors definitions and some other elements that do not fit into template extensions. Although this extension is not a template, it is still distributed along with sources. It can be then easily extended or modified using a customer AddOn. A few new platform types have been defined in the commons extension and all of them are related to OAuth2 persistent token store that has been developed to enable flawless authentication in a clustered environment. Additionally, all DTO bean definitions and mappers between platform model types and DTO objects exposed in the OCC webservices are placed in this extension.

`ycommercewebservicetest` Extension

Although the junit and integration tests are stored in the `commercewebservices` extension, the extended tests suite of `commercewebservices` module is kept in a separate extension called the `ycommercewebservicetest` extension. This extension contains its own test data set used during test execution. Tests for `ycommercewebservices` extension are written in groovy. These are modular tests which perform different operations on tested URL resources and verify the results. Using groovy tests is convenient for web services testing as not only single calls are tested, but also complex flows such as the checkout process.

There is a separate test suite for version 1 and version 2 of the web services. The latter contains a test suite written in spock, a testing framework for java and groovy applications. Spock tests have a little bit higher level of abstraction than pure groovy tests, but also are easier to maintain and eventually provide a quicker way of testing the webservices application.

The configurations, actions and wizards are held in the `backoffice` extension.

`ycommercewebservices` AddOns

The `commercewebservices` module is compliant with AddOns. The idea behind is the same as for the SAP Commerce Accelerator - by using an AddOn you can add a new/modify the existing functionality of the `commercewebservices` extension. The advantage of creating an AddOn (instead of modifying an extension) is that the upgrading to a new version is more convenient, because as no changes are made to the `commercewebservices` extensions, no customizations will have to be migrated. Currently, the OCC distribution features only the `acceleratorwebservicesaddon` AddOn.

REST-based Communication

Communication follows the model for REST-based web services. The `ycommercewebservices` template has been developed in two versions: v1 and v2. OCC v2 is the default.

i Note

For more information on the difference between these versions, see [v1 and v2 in ycommercewebservices](#).

Creating an AddOn for OCC Web Services

Learn how to create an AddOn for the OCC Web Services.

Caution

This page refers to software that has been deprecated as part of the Accelerator UI and older OCC template extensions deprecation. For more information, see [Deprecation of Accelerator UIs and Older OCC Template Extensions](#).

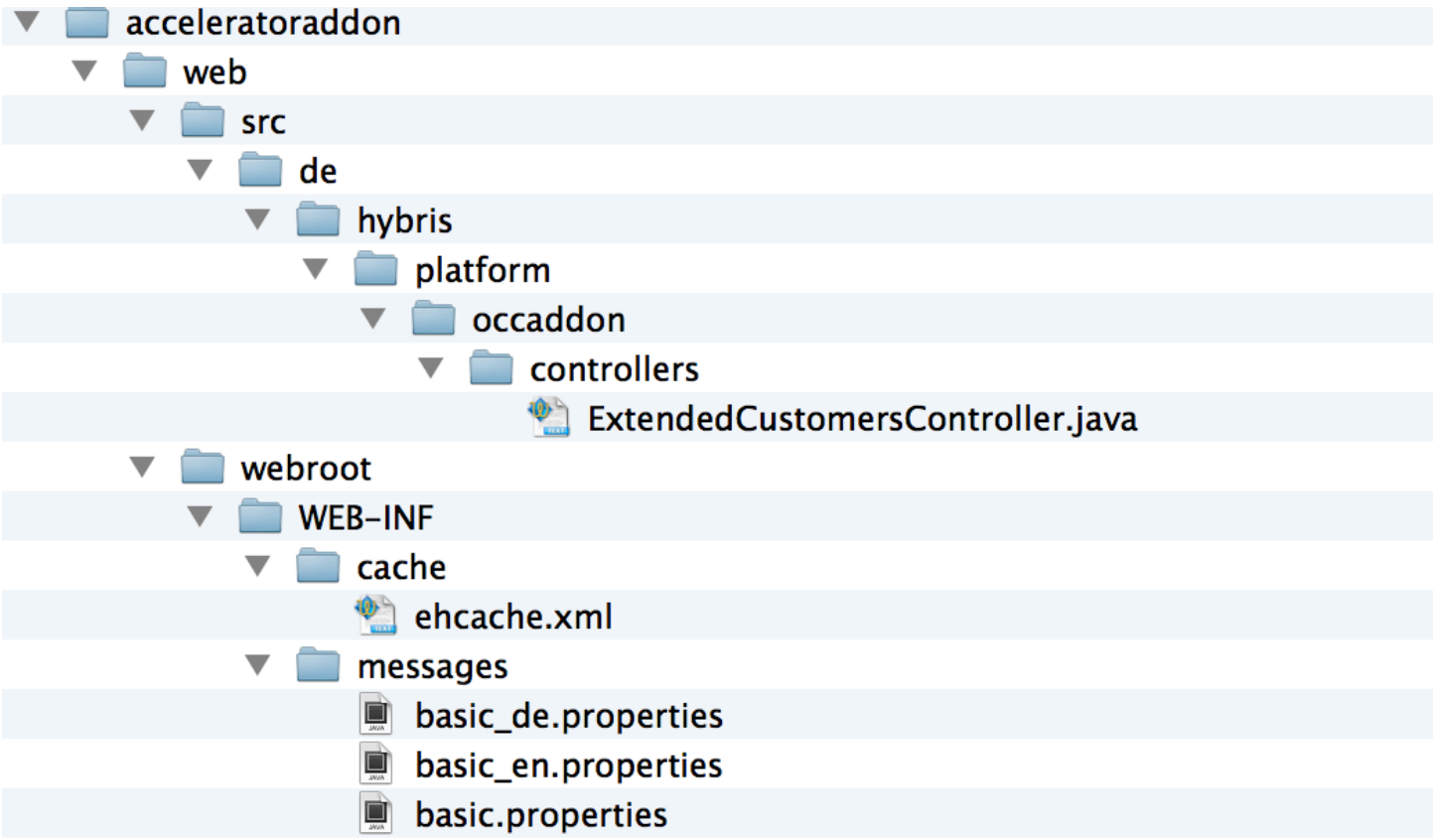
The `ycommercewebservices` extension exposes a part of the Commerce Facades as REST-based web services. As the Commerce Web Services are based on the standard Spring MVC, you can easily customize or extend them by creating new AddOns.

AddOn Structure

The structure of the AddOn is almost the same as of any other regular extension. There are certain directories and files you should be aware of:

Directory/File name	Remarks
<code><addonname>-web-spring.xml</code>	This file should be created in the resource directory, for example <code>resource\occaddon\web\spring\occaddon-web-spring.xml</code> . Also, it should contain the AddOn web context which is added to the OCC web services web context.
<code>project.properties.template</code>	<p>This file should be created so you can use the AddOn installation script. It is a template for the <code>project.properties</code> file that holds configuration properties. These properties are created during the installation process.</p> <p>i Note</p> <p>For details on installing an AddOn for a storefront refer to Installing an AddOn for a Specific Storefront</p>
<code>\acceleratoraddon</code> directory	The folder's structure mirrors the structure of folders that contain the web components of a regular extension.

The structure of an AddOn looks as follows:

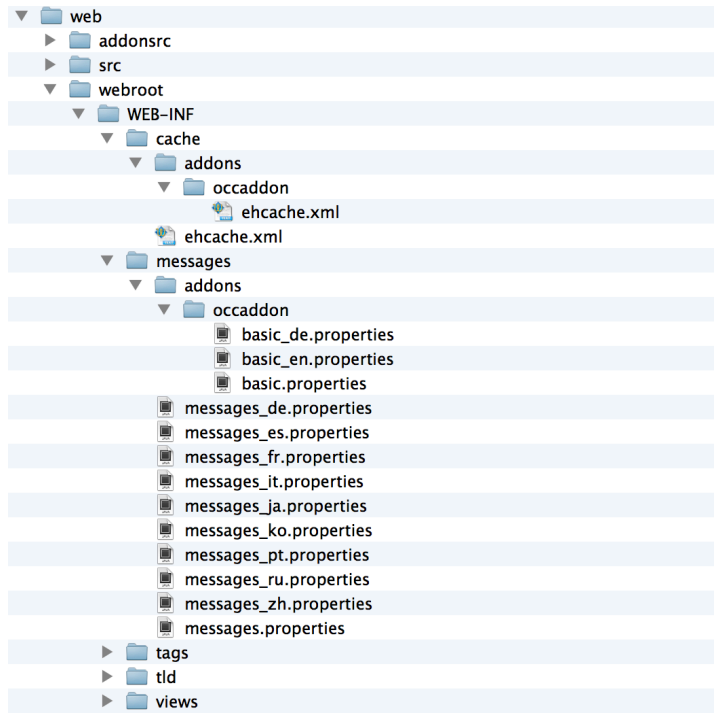


During the build phase, the system automatically copies the files from the `\acceleratoraddon` directory to the target extension, in our case, the `ycommercewebservices` extension. Files are copied into the following two directories:

Directory Name	Description
<code>\web\addonsrc</code>	Contains the source code.

Directory Name	Description
\web\webroot\WEB-INF\<resourceDir>\addons	Contains different web resources such as the message bundle (when <resourceDir> = messages) or cache configuration (<resourceDir> = cache). Similar directories can also be created for different frontend components like JSPs, HTML, images, *.tag files - but in the case of AddOns for the OCC, this part may not be required.

The structure of ycommercewebservices with addons look as follows:



Creating an AddOn

The section below instructs you on how to create an AddOn.

Generating an AddOn from a Template

In order to generate an AddOn from a Template perform the following steps:

1. Add the following extensions to your `localextensions.xml` file:
 - a.
 - `addonsupport`: Extension has an ant script required for AddOns to build correctly.
 - `yoccaddon`: Base template used to create AddOns for Commerce Web Services.
2. Generate the AddOn by using `extgen` and the `yoccaddon` extension template.

i Note

For details see [Creating a New Extension](#) and [yoccaddon Extension](#).

3. Append the new extension to the `config\localextensions.xml` file:

```
<extensions>
  ...
  <extension name="occaddon"/>
</extensions>
```

Extending the REST API

To expose new calls, you need to define a Controller class with the appropriate methods. A Controller should be created in the `\acceleratoraddon\web\src` directory. If it is created in package: `<addonPackage>.controllers` (for example `de.hybris.platform.occaddon.controllers` then it is automatically added to Commerce Web Services Spring web context.

i Note

If the controller is not created in the `<addonPackage>.controllers` package, the spring web context has to be adjusted as described in the [Creating the Web Spring Context](#) section.

```
@Controller
@RequestMapping(value = "/{baseSiteId}/newResource")
public class NewController
{
    @RequestMapping(method = RequestMethod.GET)
    @ResponseBody
    public NewResourceWsDTO getNewResource()
    {
        return new NewResourceWsDTO("newSampleResource");
    }
}
```

After installing addOn with such defined controller, it should be possible to call the following requests:

- `https://localhost:9002/rest/v1/{baseSiteId}/newResource` for version 1
- `https://localhost:9002/rest/v2/{baseSiteId}/newResource` for version 2

i Note

For details about more complex extending scenario see [Extending Commerce Web Services](#).

Installing an AddOn for Commerce Web Services

The `addoninstall` script adds the AddOn entry to the `extensioninfo.xml` of the Commerce Web Services and generates the `project.properties` file.

i Note

For details on installing an AddOn for a storefront, refer to [Installing an AddOn for a Specific Storefront](#).

Before you run your installation make sure that:

1. The `addonsupport` extension is listed in the `localextensions.xml` file or is located in the directory which is loaded automatically. These are defined by the following entry: `<path autoload="true" dir= ... />`
2. Your AddOn and `ycommercewebservices` extension are listed in the `localextensions.xml` file or are located in the directory which is loaded automatically.
3. You have properly defined the `project.properties.template` file for the AddOn.

Installing an AddOn for a Specific Extension

1. Start the ant task called `addoninstall`.
 - a. For the `occaddon` and `ycommercewebservices` the command line looks like the following:


```
ant addoninstall -Daddonnames="occaddon" -DaddonStorefront.ycommercewebservices="ycommercewebservice"
```
 - b. For the `occaddon` and `mycommercewebservices` the command looks like the following:


```
ant addoninstall -Daddonnames="occaddon" -DaddonStorefront.ycommercewebservices="mycommercewebservice"
```
2. Once the script has completed its work successfully, rebuild the system.

Related Information

ycommercewebservices Local Media Serving

The Platform comes with two filters for serving local media:

- **MediaFilter**: configured in the `web.xml` file of the mediaweb web application; any requests for non-secured media (`/media` endpoint) are handled by this filter,
- **SecureMediaFilter** configured separately for each web application (it should be added to `PlatformFilterChain`).

This approach leads to inconsistency, because the non-secured media are handled by the mediaweb application, whereas the secured ones are handled separately by each web application that needs them. A solution to that is the **WebAppMediaFilter**, which removes the inconsistency by serving both non-secured and secured media, however provided that each and every web application is configured separately.

Updating Image url

The `ycommercewebservices rest services /products` endpoint request provides `ImageData` with an `url` attribute that has been generated by the `LocalMediaWebURLStrategy` (begins with `/medias`). To ensure compatibility with our `WebAppMediaFilters`, we needed to modify the url by adding separate solutions for V1 and V2.

V1 Data Object Converter

For V1 we've added our implementation of `XStream SingleValueConverter`, which simply adds `/rest/v1` prefix for the existing url attribute value of `ImageData`.

V2 WsDTO Object Mapper

V2 DTO mapping has been implemented with Orika mappers. To achieve our goal we've added the implementation of `AbstractCustomMapper`, which is adding `/rest/v2` prefix for the url attribute value.

Sample response body GET `/products/123?fields=code,images(url)`

```
{
  "code" : "123",
  "images" : [ {
    "url" : "/rest/v2/medias/?context=..."
  } ]
}
```

Media Serving

Having updated the url in the response body, we had to set up a `WebAppMediaFilter` for `ycommercewebservices` to handle V1 or V2 media requests. To deal with that we've registered two separate `WebAppMediaFilter` spring beans and added them to the filter chains for both `ycommercewebservices` servlets (V1 and V2).

OCC Calls Security

The OCC calls security is ensured by highly configurable Spring security mechanisms.

General Information

OCC calls are secured by standard, highly configurable Spring security mechanisms. A user who gains access to the application is called a **principal**. It does not have to be a real user, it can be an external system like a backend or frontend application, or a mobile application. It is important to distinguish between authentication and authorization:

- **Authentication** means checking provided credentials. If credentials are valid, then the proper roles are assigned to a principal.
- **Authorization** means deciding if a principal can perform a given action. This is determined based on the assigned roles of the principal and also on other constraints, for example secure communication channel.

In order to simplify authentication and authorization, OCC uses a standard **OAuth2** protocol. The main purpose is to enable long-term access to the principal and differentiate security rules depending on the type of client application.

The authorization process takes place separately in two layers:

- HTTP layer
- Service (business) layer

Each layer applies its own set of rules and constraints.

OCC User Roles

The security of OCC calls is based mainly on user roles. These roles are assigned to the principal depending on the authentication type:

Authentication Type	Possible Roles
Anonymous	A non-authenticated principal is assigned a built-in ANONYMOUS role by default.
Clients	Every client application that was authenticated using an OAuth2 token in the client credentials flow is assigned a specific role depending on the client definition. When defining the clients remember to assign either the ROLE_CLIENT or ROLE_TRUSTED_CLIENT to them, because these roles allow client access to the ycommercewebservices extension. For details see: Configuring OAuth Clients .
Customers	Users who were authenticated using the OAuth2 token in the password flow are assigned a list of roles that are received from a service layer in the same way as it works in the whole application. By default, CUSTOMERGROUP and CUSTOMERMANAGERGROUP roles are used.
Guests	Anonymous users who provided their own email address. It can be done by calling /customers/current/guestlogin in v1 or /users/anonymous/carts/{guid}/email in v2. For such users, a built-in GUEST role is assigned.

Security Spring Configuration

The OAuth2 Resource Server configuration and other security aspects are defined in the **/ycommercewebservices/web/webroot/WEB-INF/config/common/security-spring.xml** file and in configuration files specific for webservices version :
/ycommercewebservices/web/webroot/WEB-INF/config/v2/security-v2-spring.xml,
/ycommercewebservices/web/webroot/WEB-INF/config/v1/security-v1-spring.xml.

OAuth configuration is stored in the Platform. You can configure the server settings in the project. properties file of the **oauth2** extension. For details see [OAuth2](#).

To make Spring Security work, you have to add the **springSecurityFilterChain** to the web services filter chain as shown below in the configuration for v1 and v2 webservices.

```
v1(/ycommercewebservices/web/webroot/WEB-INF/config/v1/filter-config-v1-spring.xml)

...
<bean id="commerceWebServicesFilterChainV1" class="de.hybris.platform.servicelayer.web.PlatformFilterChain
  <constructor-arg>
    <ref bean="commerceWebServicesFilterChainListV1" />
  </constructor-arg>
```

```

</bean>
<alias name="defaultCommerceWebServicesFilterChainListV1" alias="commerceWebServicesFilterChainListV1" />
<util:list id="defaultCommerceWebServicesFilterChainListV1">
  <!-- generic platform filters -->
  <ref bean="log4jFilter" />
  <ref bean="tenantActivationFilter" />
  <ref bean="sessionFilter" />
  <!-- commerce Webservices filters -->
  <ref bean="commerceWebServicesBaseSiteFilterV1" />
  <ref bean="commerceWebServicesSessionAttributesFilterV1" />
  <ref bean="baseSiteCheckFilterV1" />
  <!-- Security -->
  <ref bean="springSecurityFilterChain" />
  <ref bean="guestRoleFilterV1" />
</util:list>
...

```

v2(/ycommercewebservices/web/webroot/WEB-INF/config/v2/filter-config-v2-spring.xml)

```

...
<bean id="commerceWebServicesFilterChainV2" class="de.hybris.platform.servicelayer.web.PlatformFilterCha
  <constructor-arg>
    <ref bean="commerceWebServicesFilterChainListV2" />
  </constructor-arg>
</bean>
<alias name="defaultCommerceWebServicesFilterChainListV2" alias="commerceWebServicesFilterChainListV2" />
<util:list id="defaultCommerceWebServicesFilterChainListV2">
  <!-- filter that catches and resolves exceptions thrown from other filters -->
  <ref bean="exceptionTranslationFilter" />
  <!-- generic platform filters -->
  <ref bean="log4jFilter" />
  <ref bean="restSessionFilterV2" />
  <!-- commerce Webservices filters -->
  <ref bean="baseSiteMatchingFilter" />
  <ref bean="commerceWebServicesSessionAttributesFilterV2" />
  <!-- Security -->
  <ref bean="springSecurityFilterChain" />
  <ref bean="userMatchingFilter" />
  <!-- Matching filters -->
  <ref bean="cartMatchingFilter" />
  <ref bean="guestRoleFilterV2" />
</util:list>
...

```

Related Information

[Managing Users and User Groups](#)

[Managing and Checking Access Rights](#)

[OAuth 2.0](#)

OCC API Implementation

Get an overview of how the SAP Commerce OCC API is implemented, and learn what you need to know to extend it with your own custom API implementation.

[RESTful Implementation](#)

The RESTful implementation in OCC provides the user with an approach regarding the URLs and access control.

[Caching](#)

An overview of caching in OCC, along with guidelines for using it.

[Calls Reference](#)

The sample OCC calls and customer buying scenarios give you a set of examples for possible use of the OCC API.

[Save Cart in OCC](#)

The save cart functionality allows you to save and restore saved carts at a later date.

[DTO Mapping and Response Configuration](#)

The Mapping Data Mechanism facilitates mapping of the data between the source and destination objects.

[OCC Error Responses](#)

Guidelines on how you can use the OCC error responses.

[HTTP Message Converters](#)

The OCC API allows converting the DTO objects to or from their text representation (either JSON or XML) used in REST calls.

[WsDTO Concept](#)

WsDTO is a data layer used by the REST API in OCC.

[Payment in OCC](#)

The payment process flow calls description, including the payment details definition and card authorization.

[Enabling and Using the Order Status Queue](#)

The Order Status Update Queue is enabled and available by default in the `ycommercewebservices` extension. However it's not hooked to any business process responsible for order processing, because order processing is a complex mechanism and has many customizable touchpoints. Therefore it's up to the client to specify all of them and cover all possible order status changes.

[Enabling Language Fallback with OCC](#)

Ensure proper language fallback in cases where the defined language is not available for the requested data.

RESTful Implementation

The RESTful implementation in OCC provides the user with an approach regarding the URLs and access control.

Stateless

OCC does not use sessions. This means the `JSESSIONID` cookie can be (and should be) ignored. In order to access resources from a particular user, you can follow the URL convention described below (or implement your own).

Users

The user resources are available under the following path:

```
https://localhost:9002/rest/v2/{baseSiteID}/users/{userID}/...
```

Valid userID values

- **anonymous**

Anonymous user.

- **\${CustomerID}**

Customer identifier of the registered user.

Example:

```
GET https://localhost:9002/rest/v2/wsTest/users/2036bc69-d1ee-4cf4-9205-210c2f936970/addresses
```

You need to have the proper rights to see the resources of the specified user.

Carts

The cart resource is available under the following path:

`https://localhost:9002/rest/v2/{baseSiteID}/users/{userID}/carts/{cartID}`

Valid cartID values

- **current**

Represents last modified cart of the specified user.

- **\${guid}**

GUID of the anonymous cart. Works only with an **anonymous** user.

- **\${code}**

Code of the non-anonymous cart. Works only for registered users.

All calls related to the particular cart have the same structure that also contains the cart's owner:

`POST https://localhost:9002/rest/v2/wsTest/users/2036bc69-d1ee-4cf4-9205-210c2f936970/carts/0000012/entries`

This way, you can only access carts belonging to the specified user and you need to have proper rights to do so.

Orders

Orders can be accessed for a user or as a global resource for all users, but you need to have the proper rights to view the resources.

The user orders resource is available under the following path:

`https://localhost:9002/rest/v2/{baseSiteID}/users/{userID}/orders/{orderID}`

Valid orderID values

- **\${code}**

Order's code. Works only for registered users.

All calls related to the specific order have the same structure that also contains the order's owner:

`POST https://localhost:9002/rest/v2/wsTest/users/2036bc69-d1ee-4cf4-9205-210c2f936970/orders/testOrder1`

This way, you can only access orders belonging to the specified user. You must have the proper rights to do so.

The global orders resource is available under the following path:

`https://localhost:9002/rest/v2/{baseSiteID}/orders/{orderID}`

Valid code values

- **\${code}**

Order's code.

- **\${guid}**

Order's global identifier.

All calls related to the specific order have the same structure that also contains the order's owner:

`POST https://localhost:9002/rest/v2/wsTest/orders/1beb1e9f5043ef28aa5f821ada8aeee5a7a40ac4`

This way, you can access all of a user's orders, but you must have the proper rights to do so.

Access Control

Role	Description	Rights
ROLE_CLIENT	Client application (i.e. mobile app)	Can access only anonymous user resources
ROLE_CUSTOMERGROUP	User authenticated by client application	Can access only its own resources
ROLE_TRUSTEDCLIENT	Trusted client application (i.e. Adobe CQ5)	Can access all users and their resources
ROLE_CUSTOMERMANAGERGROUP	User manager authenticated by client app (i.e. POS terminal)	Can access all users and their resources

Verbs in RESTful API

In v2 of the OCC API, some of the HTTP methods have been redefined to match a more RESTful standard.

POST: Create a Resource

POST is used to create a subordinate resource which does not exist. As a result a created entity is returned. Examples: creating an order or address.

PUT: Update an Entire Resource

PUT is used to update a **complete** entity by sending an entire entity to a URL which points to a particular resource. All fields that are missing will be set to **NULL** or **default value**. Examples: updating a user or address.

PATCH: Update a Resource Partially

PATCH is used for a **partial** update. For instance, when you only need to update one field of the resource a PATCH is used. PUTting or POSTing a complete resource representation creates additional overhead and utilizes more bandwidth. Examples: updating a user's last name or updating the street name of an address.

DELETE: Remove a Resource

DELETE is used to remove the resource. Examples: removing a user or address.

Caching

An overview of caching in OCC, along with guidelines for using it.

Overview

A Cache sits between one or more Web servers (also known as origin servers) and a client or many clients, and watches requests come by, saving copies of the responses, such as HTML pages, images and files. Then, if another request for the same URL is placed, it can use one of the collected responses, instead of asking the origin server for it again. This document describes how to use caching in OCC. It provides the details on both client-side caching and server-side caching.

Client-Side Caching

The `webservicesscommons` extension defines the `@CacheControl` annotation which can be used to generate Cache-Control Header in response. If you want to enable the client-side caching for a particular method or the entire controller, simply annotate it with `@CacheControl` and specify appropriate directives. The annotation usage example can be noticed in the `ProductsController.java` class:

```
...
    @RequestMapping(value = "{productCode}", method = RequestMethod.GET)
    @CacheControl(directive = CacheControlDirective.PRIVATE, maxAge = 120)
```



```

@ResponseBody
public ProductWsDTO getProductByCode(@PathVariable final String productCode,
    @RequestParam(defaultValue = DEFAULT_FIELD_SET) final String fields)
{
    ...
}
...

```

i Note

Since Cache-Control annotation applies to GET and HEAD methods only, it will not affect any other request methods.

i Note

The @CacheControl annotation work only if CacheControlHandlerInterceptor is added to mvc interceptors.

```

...
<mvc:interceptors>
    <bean class="de.hybris.platform.webservicescommons.interceptors.CacheControlHandlerInterceptor"/>
</mvc:interceptors>
...

```

Server-Side Caching

Spring Cache Configuration

The Spring cache configuration can be found in the following file: ycommercewebservices/web/webroot/WEB-INF/config/cache-config-spring.xml.

The caching feature has to be enabled. It can be done using the <cache:annotation-driven> element. This element allows also to define the default key generator and cache manager which will be used for caching.

Configuration below uses the commerceCacheKeyGenerator described in section below : <Cache Key Generator>. The cache manager is defined with use of Spring CompositeCacheManager to make cache mechanism available also for addons. The single cache manager for OCC uses TenantAwareEhCacheManagerFactoryBean class, which is defined in the webservicescommons extension.

```

<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:cache="http://www.springframework.org/schema/cache"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/cache
        http://www.springframework.org/schema/cache/spring-cache.xsd">

    <cache:annotation-driven cache-manager="compositeWsCacheManager" key-generator="commerceCacheKeyGenerator"/>

    <alias name="defaultWsCacheManagerList" alias="wsCacheManagerList"/>
    <utils:list id="defaultWsCacheManagerList">
        <ref bean="defaultWSCacheManager"/>
    </utils:list>

    <!-- Composite cache manager is used to allow addons to add their own cache managers by modifying wsCacheMa

    <alias name="defaultCompositeWSCacheManager" alias="compositeWsCacheManager"/>
    <bean id="defaultCompositeWSCacheManager" class="org.springframework.cache.support.CompositeCacheManager">
        <property name="cacheManagers">
            <ref bean="wsCacheManagerList"/>

```

```

    </property>
  </bean>

  <!-- Default cache manager for OCC: -->

  <alias name="defaultWSCacheManager" alias="wsCacheManager"/>
  <bean id="defaultWSCacheManager" class="org.springframework.cache.ehcache.EhCacheCacheManager">
    <property name="cacheManager" ref="wsEhcache"/>
  </bean>

  <alias name="defaultWSEhcache" alias="wsEhcache"/>
  <bean id="defaultWSEhcache" class="de.hybris.platform.commerceservicescommons.cache.TenantAwareEhCacheMa
    <property name="configLocation" value="/WEB-INF/cache/ehcache.xml"/>
  </bean>
</beans>

```

Ehcache Configuration

Ehcache is an open-source, standards-based cache used to improve performance, offload the database, and simplify the scalability. The detail cache configuration can be found in the following file: `ycommercewebservices/web/webroot/WEB-INF/ehcache.xml`.

```

<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="../config/ehcache.xsd" updateCheck="false" monitoring="autodetect"
  dynamicConfig="true">

  <!--
  see ehcache-core-*.jar/ehcache-failsafe.xml for description of elements
  -->

  <diskStore path="java.io.tmpdir/occ_cache"/>
  <defaultCache
    maxElementsInMemory="100000"
    eternal="false"
    timeToIdleSeconds="360"
    timeToLiveSeconds="360"
    overflowToDisk="true"
    diskPersistent="false"
    maxEntriesLocalDisk="10"
    diskExpiryThreadIntervalSeconds="360"
    memoryStoreEvictionPolicy="FIFO"
  />

  <cache name="fieldSetCache"
    maxElementsInMemory="1000"
    eternal="true"
    overflowToDisk="true"
    diskPersistent="false"
    maxEntriesLocalDisk="2000"
    memoryStoreEvictionPolicy="LRU"/>

  <cache name="productSearchCache"
    maxElementsInMemory="1000"
    eternal="false"
    overflowToDisk="true"
    timeToLiveSeconds="150"
    diskPersistent="false"
    maxEntriesLocalDisk="2000"
    memoryStoreEvictionPolicy="LRU"/>

  ...
  ...

```

```
...
</ehcache>
```

Cached Methods

Methods which are to be cached must be annotated with the `@Cacheable` annotation. In the simplest format, the annotation requires only the name of the cache associated with this method. For example, `@Cacheable(defaultCache)`. In such a case, the key for the cache is generated using the **key-generator** defined in the `< cache:annotation-drive >` tag. For the `ycommercewebservices` extension it is the `commerceCacheKeyGenerator`. If you want to use a different key generator or define a key using SpEL, then use the `key` attribute of the annotation. For example, `@Cacheable(value = defaultCache, key = <"#{#param1,#param2,#param3}>")`.

Cache Key Generator

The default key generator configured for the `ycommercewebservices` extension is `commerceCacheKeyGenerator`. It is defined in the `commercewebservicescommons-spring.xml` file.

```
...
<bean id="commerceCacheKeyGenerator" class="de.hybris.platform.commercewebservicescommons.cache.CommerceCa
  <property name="baseSiteService" ref="baseSiteService"/>
</bean>
...
```

This key generator creates a key based on the method parameters and some additional attributes like: `<base site>`, `<language>`, `<user>`, `<currency>`. The base site identifier and language ISO codes are always added to the generated key. The user identifier and currency ISO code are not added by default, but can be easily added as presented by the following example:

```
@Cacheable(value = "productCache", key =
  "T(de.hybris.platform.commercewebservicescommons.cache.CommerceCacheKeyGenerator).generateKey(true,true,#produc
```

The first parameter of the `generateKey` method defines if a user identifier should be added to the cache key. The second parameter defines if a currency isocode should be added to the cache key.

Calls Reference

The sample OCC calls and customer buying scenarios give you a set of examples for possible use of the OCC API.

Before working with the sample calls, review the following key features of the OCC API:

Stateless

OCC does not use sessions. In order to access resources for a particular user you can use the following URL conventions:

- User resources: `https://localhost:9002/rest/v2/{baseSiteID}/users/{userID}/...`
- Cart resources: `https://localhost:9002/rest/v2/{baseSiteID}/users/{userID}/carts/{cartID}...`
- Order resources: `https://localhost:9002/rest/v2/{baseSiteID}/users/{userID}/orders/{orderID}...`

where `{userID}` can have the following values:

- `${customerId}` - Unique customerId of the registered user.
- `anonymous` - Anonymous user.
- `current` - User represented by the OAuth token.

where `{cartID}` can have the following values:

- `${guid}` - Globally Unique Identifier (GUID) of the anonymous cart. Works only for an **anonymous** user.
- `${code}` - Code of the non-anonymous cart. Works only for registered users.
- `current` - Represents last modified cart of the specified user.

Localization Request Parameters

Each of the calls can contain additional URL parameters that change the localization of the returned objects. The common parameters are:

- **lang**: Changes the language of the localized values in the response. Provide the language ISO code as a value. If no lang parameter is provided, then the response is localized with the default language of your base store.
- **curr**: Changes the currency of your web service call. This means that all the calculations are performed in the requested currency and all the price values are presented using the requested currency. Provide currency ISO code as a value. If no *<curr>* parameter is provided, then the default currency of your base store is used.

You can use these parameters with every requested resource. The parameters handling is isolated in a specialized HTTP request filter. Check the following examples:

- `https://localhost:9002/rest/v2/mysite/users/{userId}/carts?curr=USD&lang=en`: Use English language and US Dollar for the request.
- `https://localhost:9002/rest/v2/mysite/products/{productCode}?curr=USD`: Use US Dollar and default language of mysite's store.
- `https://localhost:9002/rest/v2/mysite/stores/{storeId}?lang=de`: Use German language and default currency of mysite's store.

Fields Parameter

For most resource requests, there is a **fields** parameter, which can be used to configure a response. It allows you to select fields for every object returned in a response. It can be composed of field names and levels separated by a comma, for example: BASIC_FIELD_LEVEL, field1,field2. It can also have a nested configuration for selected fields, such as: `field1(field11,field12)`.

Cart Recalculation Options in the CartMatchingFilter

V2 allows you to modify cart recalculation options in **CartMatchingFilter**.

In v2, the customer's cart is persisted in the database and loaded by **CartMatchingFilter** for every incoming cart resource request. When loading the cart, the filter checks if the cart has already expired. If so, the cart is rebuilt but no recalculation is performed. The new rebuilt cart contains all the entries rewritten from the expired cart. If you want to specify the validity time of the cart counted from the last modification of the cart, add the following property to your `local.properties` file:

```
commerceservices.cartValidityPeriod=<seconds>
```

If the cart has not expired, the loaded cart is recalculated only if its currency is different from the one already set by the currency filter. You can customize the cart recalculation behavior by changing the values of the `refreshCart` request parameter and the `ycommercewebservices.cart.refreshed.by.default` property. The `refreshCart` parameter is optional and can be used only with cart resource requests.

If the request contains the `refreshCart` parameter with the `true` value, the **CartMatchingFilter** forces the cart recalculation. See the following example of a "getting cart delivery modes" request with the `refreshCart` parameter forcing the cart recalculation:

```
https://localhost:9002/rest/v2/{baseSiteID}/users/{userID}/carts/{cartID}/deliverymodes?refreshCart=true
```

If the `refreshCart` parameter is not set, the behavior of the cart recalculation depends on the `ycommercewebservices.cart.refreshed.by.default` property. By default, the value of the property is set to `false`:

```
ycommercewebservices.cart.refreshed.by.default=false
```

The default value of the property ensures that the cart is recalculated only if its currency is different from the one already set by the currency filter. If you want to recalculate the cart for every cart resource request, add the property with the `true` value to your `local.properties` file.

i Note

Changing the default value of the `ycommercewebservices.cart.refreshed.by.default` property to recalculate carts for every cart resource request could slow your system down.

To disable the cart recalculation explicitly, use the `refreshCart` parameter with the value `false`. See the following example of a “getting cart delivery modes” request with the cart recalculation disabled for every request:

```
https://localhost:9002/rest/v2/{baseSiteID}/users/{userID}/carts/{cartID}/deliverymodes?refreshCart=false
```

DTO in Request Body

The Version 2 features some new requests that accept DTOs in a request body. In such requests, parameters can be passed in JSON or XML format. An example of this can be seen in the Body Parameters - Content-Type: application/json column.

Single Calls and Scenarios

Use the following links to learn more about executing a complete purchase scenario using the OCC calls.

- [Customer Buying Process Scenarios](#)
- [Single Calls for Customer Buying Scenarios](#)

Customer Buying Process Scenarios

Step-by-step guidance on the sequences of calls that should be used in order to complete a given scenario (for example registering a customer).

Overview

The scenarios provided below represent a complete step-by-step guidance on using a given sequence of calls.

i Note

The URL examples are used for presentation purposes only. In a customized environment, you must replace the server address with your own.

Registered Customer

Prerequisites

A customer is already logged in the store and has defined all information required for the checkout process (e.g. the address and payment details).

Scenario Steps

- Search for a product
- Get product details
- Add product to a cart (e.g.basket)
- Get access token for a customer
- Perform Checkout
- Display the placed order for a customer
- Log out

Request Flow

i Note

The calls should be executed in the order defined in the request flow table. Scroll to the right to see the full table content.

Method	URL
GET	https://localhost:9002/rest/v2/electronics/products/search
GET	https://localhost:9002/rest/v2/electronics/products/489702
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efaf
POST	https://localhost:9002/authorizationserver/oauth/token
POST	https://localhost:9002/rest/v2/electronics/users/current/carts
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001
GET	https://localhost:9002/rest/v2/electronics/users/current/addresses
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/addresses/delivery
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymodes
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymode
GET	https://localhost:9002/rest/v2/electronics/users/current/paymentdetails
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/paymentdetails
POST	https://localhost:9002/rest/v2/electronics/users/current/orders
GET	https://localhost:9002/rest/v2/electronics/users/current/orders
GET	https://localhost:9002/rest/v2/electronics/users/current/orders/00012002

New Customer

Prerequisites

A customer is not registered in the store.

Scenario Steps

- Register a new customer and get the access token
- Create a new address for the customer

- Search for a specific product
- Get the product details
- Create a cart for the customer
- Add the products to the cart
- Checkout: set a delivery address, set delivery mode, define credit card information
- Place the order
- Display placed order for the customer

Request Flow

i Note

The calls should be executed in the order defined in the request flow table. **Scroll to the right to see the full table content.**

Method	URL	Query Parameters	E
POST	https://localhost:9002/authorizationserver/oauth/token		C
POST	https://localhost:9002/rest/v2/electronics/users		c
POST	https://localhost:9002/authorizationserver/oauth/token		lc &
POST	https://localhost:9002/rest/v2/electronics/users/current/addresses	fields=id	c & fi
GET	https://localhost:9002/rest/v2/electronics/products/search	query=tripod	
GET	https://localhost:9002/rest/v2/electronics/products/3429337		

Method	URL	Query Parameters	E
POST	https://localhost:9002/rest/v2/electronics/users/current/carts		C
POST	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/entries		c
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/addresses/delivery		a
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymodes		
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymode		d
POST	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/paymentdetails		a 1
POST	https://localhost:9002/rest/v2/electronics/users/current/orders		c

Method	URL	Query Parameters	E
GET	https://localhost:9002/rest/v2/electronics/users/current/orders		C
GET	https://localhost:9002/rest/v2/electronics/users/current/orders/00012002		

Pick Up In Store

Prerequisites

A customer is already registered in the store and has defined all information needed for the checkout process (like payment information).

Scenario Steps

- Get an access token for the customer
- Create a cart for customer
- Add products that will be picked up in the store
- Perform checkout and set 'pickup' as the delivery mode
- Display the placed order to the customer

Request Flow

i Note

The calls should be executed in the order defined in the request flow table. Scroll to the right to see the full table content.

Method	URL	Query Parameters
POST	https://localhost:9002/authorizationserver/oauth/token	
POST	https://localhost:9002/rest/v2/electronics/users/current/carts	
POST	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/entries	fields=statusCode,qua
PATCH	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/entries/0	

Method	URL	Query Parameters
POST	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/entries	
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001	fields=code,pickuplten pickupOrderGroups(er
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymodes	fields=BASIC
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymode	
GET	https://localhost:9002/rest/v2/electronics/users/current/paymentdetails	saved=true
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/paymentdetails	
POST	https://localhost:9002/rest/v2/electronics/users/current/orders	
GET	https://localhost:9002/rest/v2/electronics/users/current/orders	
GET	https://localhost:9002/rest/v2/electronics/users/current/orders/00012002	

Pick Up in Store with Store Consolidation

Prerequisites

A customer is already registered in the store and has defined all information needed for the checkout process (such as address and payment information)

Scenario Steps

i Note

The [acceleratorwebservicesaddon AddOn](#) has to be installed.

- Get an access token for customer

- Create a cart for customer
- Add a product that will be picked up in the store
- Add a product that will be picked up in a different store
- Use the consolidate store functionality
- Perform checkout

i Note

The calls should be executed in the order defined in the request flow table. Scroll to the right to see the full table content.

Method	URL	Query Parameters
POST	https://localhost:9002/authorizationserver/oauth/token	
POST	https://localhost:9002/rest/v2/electronics/users/current/carts	
POST	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/entries	
POST	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/entries	
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001	fields=code,pickup pickupOrderGroups
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/consolidate	
POST	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/consolidate	
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymodes	

Method	URL	Query Parameters
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymode	
GET	https://localhost:9002/rest/v2/electronics/users/current/paymentdetails	saved=true
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/paymentdetails	
POST	https://localhost:9002/rest/v2/electronics/users/current/orders	

Combining Delivery Mode With Pick Up in Store

Prerequisites

A customer is already registered in the store and has defined all information needed for checkout process (such as address and payment information).

Scenario Steps

- Get an access token for the customer
- Create a cart for the customer
- Add a product that will be delivered
- Add a product that will be picked up in a store
- Perform checkout
- Display the placed order for the customer

Note

The calls should be executed in the order defined in the request flow table. Scroll to the right to see the full table content.

Request Flow

Method	URL	Query Parameters
POST	https://localhost:9002/authorizationserver/oauth/token	
POST	https://localhost:9002/rest/v2/electronics/users/current/carts	
POST	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/entries	

Method	URL	Query Parameters
POST	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/entries	
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001	fields=code,pickupOrderGroup,deliveryItemsQuery
GET	https://localhost:9002/rest/v2/electronics/users/current/addresses	
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/addresses/delivery	
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymodes	
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymode	
GET	https://localhost:9002/rest/v2/electronics/users/current/paymentdetails	saved=true
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/paymentdetails	
POST	https://localhost:9002/rest/v2/electronics/users/current/orders	
GET	https://localhost:9002/rest/v2/electronics/users/current/orders	
GET	https://localhost:9002/rest/v2/electronics/users/current/orders/00012002	

Checkout Process

Checkout with Creating New Address and Payment Information for Customer

Prerequisites

A customer is registered and has a cart with code 00012001.

Scenario Steps

- Create address and set it as delivery address
- Set delivery mode
- Create the payment information
- Authorize the cart
- Place the order

Request Flow

i Note

The calls should be executed in the order defined in the request flow table. Scroll to the right to see the full table content.

Method	URL	Body Parameters
		Content-Type: application/json
POST	https://localhost:9002/rest/v2/electronics/users/current/addresses	firstName=John&lastName=Smith&addressId=87961
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/addresses/delivery	addressId=87961
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymodes	
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymode	deliveryModeId=1
POST	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/paymentdetails	accountHolderName=John Smith 16-19&billingAddressId=87961

Method	URL	Body Parameters
		Content-Type: application/json
POST	https://localhost:9002/rest/v2/electronics/users/current/orders	cartId=00012001

Checkout with Setting Existing Address and Payment Information for Customer

Prerequisites

A customer is registered and has a cart with code 00012001.

Scenario Steps

- Try to get the delivery modes
- Set the delivery address
- Set the delivery mode
- Set the payment information
- Authorize the cart
- Place the order

Request Flow

i Note

The calls should be executed in the order defined in the request flow table. Scroll to the right to see the full table content.

Method	URL	Query Parameters
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymodes	
GET	https://localhost:9002/rest/v2/electronics/users/current/addresses	
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/addresses/delivery	

Method	URL	Query Param
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymodes	fields=BASIC
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymode	
GET	https://localhost:9002/rest/v2/electronics/users/current/paymentdetails	saved=true&f
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/paymentdetails	
POST	https://localhost:9002/rest/v2/electronics/users/current/orders	

Checkout Process with SOP

Prerequisites

A customer is registered and has a cart with code 00012001.

Scenario Steps

- Set the delivery address
- Set the delivery mode
- Get information needed to create the payment subscription
- Create a subscription contacting directly with payment provider
- Handle the response from payment provider and create payment details
- Authorize the card and place order

Request Flow

i Note

The calls should be executed in the order defined in the request flow table. Scroll to the right to see the full table content.

Method	URL	Query Parameters
GET	https://localhost:9002/rest/v2/electronics/users/current/addresses	
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymodes	fields=BASIC

Method	URL	Query Parameters
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/payment/sop/request	
POST	<i>postUrl</i> got in previous request	
POST	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/payment/sop/response	
POST	https://localhost:9002/rest/v2/electronics/users/current/orders	
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/addresses/delivery	
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymode	

Checkout Process with SOP and Extended Merchant Callback

Prerequisites

A customer is registered and has a cart with code 00012001.

Scenario Steps

- Set the delivery address
- Set the delivery mode
- Get information needed to create payment subscription
- Create subscription contacting directly with payment provider
- Ask OCC about payment provider response - negative response
- Remove the payment response information
- Create a subscription contacting directly with payment provider
- Ask OCC about payment provider response - positive response
- Authorize the card and place the order

Request Flow

i Note

This is custom documentation. For more information, please visit the [SAP Help Portal](#).

The calls should be executed in the order defined in the request flow table. Scroll to the right to see the full table content.

Method	URL	Query Parameter
GET	https://localhost:9002/rest/v2/electronics/users/current/addresses	
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/addresses/delivery	
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymodes	fields=BAS
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymode	
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/payment/sop/request	
POST	<i>payment provider Url</i> (postUrl got in previous request)	
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/payment/sop/response	
	i Note Assumption : reponse from previous call was negative (e.g. Customer gave wrong card number)	
DELETE	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/payment/sop/response	
POST	<i>payment provider Url</i>	
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/payment/sop/response	

Method	URL	Query Parameter
POST	https://localhost:9002/rest/v2/electronics/users/current/orders	

Guest Checkout Scenario with Creating Full Account

Prerequisites

None.

Scenario Steps

- Search for a product
- Get the product details
- Create an empty cart
- Add product to cart
- Get a client access token
- Introduce oneself as a guest
- Checkout
- Display placed order
- Convert guest account to full customer account (optional)
- Log in as a customer
- Get the order list

Request Flow

i Note

The calls should be executed in the order defined in the request flow table. Scroll to the right to see the full table content.

Method	URL
GET	https://localhost:9002/rest/v2/electronics/products/search
GET	https://localhost:9002/rest/v2/electronics/products/489702
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts

Method	URL
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4e
POST	https://localhost:9002/authorizationserver/oauth/token
PUT	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4e
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4e
GET	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4e
PUT	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4e
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4e

Method	URL
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/orders
POST	https://localhost:9002/rest/v2/electronics/users
POST	https://localhost:9002/authorizationserver/oauth/token
GET	https://localhost:9002/rest/v2/electronics/users/current/orders
GET	https://localhost:9002/rest/v2/electronics/users/current/orders/00002009

Guest Checkout Scenario with Checking Order Status

Prerequisites

None.

Scenario Steps

- Search for a product
- Get the product details
- Create an empty cart
- Add a product to the cart
- Get a client access token

- Identify yourself as a guest
- Perform checkout
- Get order information

Request Flow

i Note

The calls should be executed in the order defined in the request flow table. Scroll to the right to see the full table content.

Method	URL
GET	https://localhost:9002/rest/v2/electronics/products
GET	https://localhost:9002/rest/v2/electronics/products /489702
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efaf
POST	https://localhost:9002/authorizationserver/oauth/token
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efaf
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efaf

Method	URL
GET	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efaf
PUT	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efaf
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efaf
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/orders
GET	https://localhost:9002/rest/v2/electronics/users/anonymous/orders/51ecf334a103df146a85b486af01aad57df4efaf

Voucher Usage

Prerequisites

A voucher with code HY2008 is defined in the system..

Scenario Steps

- Create an empty cart
- Add products to the cart
- Apply the voucher
- Get the cart
- Register the new customer and get an access token for it
- Assign the cart to customer
- Create a new address for the customer
- Checkout: set delivery address, set delivery mode, set credit card information
- Place order
- Display placed orders for the customer

Request Flow

i Note

The calls should be executed in the order defined in the request flow table. Scroll to the right to see the full table content.

Method	URL
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efaf
GET	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efaf
GET	https://localhost:9002/authorizationserver/oauth/token
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efaf
GET	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efaf

Method	URL
POST	https://localhost:9002/rest/v2/electronics/users
POST	https://localhost:9002/authorizationserver/oauth/token
POST	https://localhost:9002/rest/v2/electronics/users/current/carts
POST	https://localhost:9002/rest/v2/electronics/users/current/addresses
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00002036/addresses/delivery
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00002036/deliverymodes
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00002036/deliverymode
POST	https://localhost:9002/rest/v2/electronics/users/current/carts/00002036/paymentdetails

Method	URL
POST	https://localhost:9002/rest/v2/electronics/users/current/orders
GET	https://localhost:9002/rest/v2/electronics/users/current/orders
GET	https://localhost:9002/rest/v2/electronics/users/current/orders/00002037

Cart Merging

Prerequisites

A customer is already registered in the store and has a cart with one product (product code = 1382080). Cart code= 00001002 and a cart guid=51ecf334a103df146a85b486af01aad57df4efa0.

Scenario Steps

- Search for products
- Get product details
- Create an empty cart
- Insert the product to the cart
- Get the token for customer
- Get customer carts
- Merge the last modified customer cart with anonymous cart
- Get cart

Request Flow

i Note

The calls should be executed in the order defined in the request flow table. Scroll to the right to see the full table content.

Method	URL
GET	https://localhost:9002/rest/v2/electronics/products/search
GET	https://localhost:9002/rest/v2/electronics/products/489702
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/02017b6dad8834ba6f3fd26eb9ab4f270e9b185i
GET	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/02017b6dad8834ba6f3fd26eb9ab4f270e9b185i
POST	https://localhost:9002/authorizationserver/oauth/token
GET	https://localhost:9002/rest/v2/electronics/users/current/carts
POST	https://localhost:9002/rest/v2/electronics/users/current/carts
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00001002

[RESTful Implementation in v2](#)
[Response Configuration in v2](#)
[Single Calls for Customer Buying Scenarios](#)

Examples of single steps of Buying Process Scenarios for the Omni Commerce Connect (OCC) in v2.

The tables below present sets of available calls that may be executed using the v2 API.

The calls have been grouped into topics reflecting their purpose, however they should be treated as separate examples rather than complete scenarios. For complete business scenarios, refer to [Customer Buying Process Scenarios](#).

The URL examples are used for presentation purposes only. In a customized environment, you must replace the server address with your own.

Main Topic	Method	URL	Query Parameters	Body Parameters
				Content-Type: application/x-www-form-urlencoded
New Customer Registration	POST	https://localhost:9002/authorizationserver/oauth/token		client_id=\$CLIENT_ID\$&client_secret=\$CLIENT_SECRET\$
	POST	https://localhost:9002/rest/v2/electronics/users		login=john.doe@mail.com&password=\$PASSWORD\$&firstName=John&lastName=Doe&title=Software Engineer
Get Customer Token	POST	https://localhost:9002/authorizationserver/oauth/token		client_id=\$CLIENT_ID\$&client_secret=\$CLIENT_SECRET\$&grant_type=password&username=john.doe@mail.com&password=\$PASSWORD\$

This is custom documentation. For more information, please visit the [SAP Help Portal](#).

Main Topic	Method	URL	Query f
Authorization	POST	https://localhost:9002/authorizationserver/oauth/token	
Address management	GET	https://localhost:9002/rest/v2/electronics/users/current/addresses	fields=
	POST	https://localhost:9002/rest/v2/electronics/users/current/addresses	
	PATCH	https://localhost:9002/rest/v2/electronics/users/current/addresses/8796158590999	
Payment Management	GET	https://localhost:9002/rest/v2/electronics/users/current/paymentdetails	saved=i
	GET	https://localhost:9002/rest/v2/electronics/users/current/paymentdetails/8796191359018	fields=i expiryY
	DELETE	https://localhost:9002/rest/v2/electronics/users/current/paymentdetails/8796191359018	
Customer Profile Management	PATCH	https://localhost:9002/rest/v2/electronics/users/current	

Main Topic	Method	URL	Query f
Password Management	POST	https://localhost:9002/occ/v2/electronics/users/current/password	None

Customer Profile Management (Using Customer Manager)

Main Topic	Method	URL	Query Parame
Authorization	POST	https://localhost:9002/authorizationserver/oauth/token	
Address Management	GET	https://localhost:9002/rest/v2/electronics/users/current/addresses	fields=[
	POST	https://localhost:9002/rest/v2/electronics/users/current/addresses	
	PATCH	https://localhost:9002/rest/v2/electronics/users/current/addresses/8796158590999	
Payment Management	GET	https://localhost:9002/rest/v2/electronics/users/current/paymentdetails	saved=i
	GET	https://localhost:9002/rest/v2/electronics/users/current/paymentdetails/8796191359018	
	DELETE	https://localhost:9002/rest/v2/electronics/users/current/paymentdetails/8796191359018	
Customer Profile Management	PATCH	https://localhost:9002/rest/v2/electronics/users/john.smith@mail.com	

Main Topic	Method	URL	Query Parame
Password Management	POST	https://localhost:9002/occ/v2/electronics/users/current/password	None

Product Search

Main Topic	Method	URL	Parameters
Product Search	GET	https://localhost:9002/rest/v2/electronics/products/search	query=:price-asc:category:575
	GET	https://localhost:9002/rest/v2/electronics/products/search	query=:name-desc:brand:brand_10
	GET	https://localhost:9002/rest/v2/electronics/products/search	query=camera&pageSize=40&fields=products(nar
	GET	https://localhost:9002/rest/v2/electronics/products/search	query=camera:relevance:category:575&fields=products(BASIC),pagination(DEFAULT)

Main Topic	Method	URL	Parameters
	GET	https://localhost:9002/rest/v2/electronics/products/search	query=camera:relevance:category:575:brand:branc
	GET	https://localhost:9002/rest/v2/electronics/products/search	query=camera:relevance:category:575:brand:branc

Future Stock Availability

Main Topic	Method	URL	Parameters
Future Stock Availability	GET	https://localhost:9002/rest/v2/electronics/users/current/futureStocks	productCodes=489702
	GET	https://localhost:9002/rest/v2/electronics/users/current/futureStocks/489702	

Search for Nearest Store Having Particular Product in Stock

Main Topic	Method	URL	Parameters
Store Search	GET	https://localhost:9002/rest/v2/electronics/products/search	query=EOS

Main Topic	Method	URL	Parameters
	GET	https://localhost:9002/rest/v2/electronics/products/1382080	
	GET	https://localhost:9002/rest/v2/electronics/products/1382080/stock	location=Tokio&fields=stores(name),pagir

Adding Product to Cart for Anonymous User

Main Topic	Method	URL
Creation of an Empty Cart	POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts
Cart Management	POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01
	POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01

Main Topic	Method	URL
	POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01
	PATCH	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01
	PATCH	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01
	PUT	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01
	DELETE	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01
	GET	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01
Cart Validation	POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01

Main Topic	Method	URL

Checkout Process

For detailed information on available calls refer to: [Customer Buying Process - Checkout Process Scenarios](#)

Checkout Process with SOP

For detailed information on available calls refer to: [Customer Buying Process - Checkout Process Scenarios](#)

Customer Order Management

Main Topic	Method	URL	Query Parameters
Order Management	POST	https://localhost:9002/authorizationserver/oauth/token	
	GET	https://localhost:9002/rest/v2/electronics/users/current/orders	fields=BASIC
	GET	https://localhost:9002/rest/v2/electronics/users/current/orders/00001002	fields=statusDisplay,creat

Store Search, Display Store Details

Main Topic	Method	URL	Parameters
Store Search	GET	https://localhost:9002/rest/v2/electronics/stores	query=Tokio&fields=stores(BASIC),pagination(BASIC)

Main Topic	Method	URL	Parameters
Display Store Details	GET	https://localhost:9002/rest/v2/electronics/stores/Nakano	fields=DEFAULT

Cart Merging

For detailed information on available calls refer to: [Customer Buying Process Scenarios - Cart Merging](#) .

Consent Management REST APIs

The Consent Management APIs provide self-service consent management for your decoupled storefront by allowing you to create and retrieve customer consent preferences.

The Consent Management APIs allow you to:

- Get a list of all of a user's consents
- Get a specific consent
- Record the giving of a consent
- Record consent withdrawal

i Note

All calls are documented in OpenAPI docs in SAP Commerce Cloud: <https://HOST:9002/rest/v2/swagger-ui.html>

Getting All Consents

The following example shows the REST API call to get all consents:

<https://{HOST}:9002/rest/v2/{STOREFRONT}/users/{USERID}/consenttemplates>

The following example shows the response that is generated from this call:

```
{
  "consentTemplates": [
    {
      "currentConsent": {
        "code": "000000RT",
        "consentGivenDate": "2018-06-12T19:00:55+0000",
        "consentWithdrawnDate": "2018-06-12T19:02:22+0000"
      },
      "description": "This is a sample consent description that will need to be updated or replaced, base",
      "id": "MARKETING_NEWSLETTER",
      "name": "I approve to this sample consent",
      "version": 0
    }
  ]
}
```

Countries REST API for Billing and Delivery Countries

The Countries REST API allows you to get a list of countries that are defined for either billing or delivery.

The Billing Countries REST API complements the `/deliverycountries` REST API. The functionality in the Billing Countries REST API provides the benefit of being able to distinguish between countries you deliver to as opposed to countries you can bill from. For example, if your storefront delivers only to European Community countries, you can still allow billing from countries outside of Europe. This call is meant to be used to

populate address forms. The final validation and acceptance of the country is handled through a payment provider. Using the billing countries functionality increases data accuracy and improves the user experience for your decoupled storefront.

i Note

All calls are documented in OpenAPI docs in SAP Commerce Cloud: <https://HOST:9002/rest/v2/swagger-ui.html>

Getting All Billing and Delivery Countries

The following example shows the REST API call to get all billing countries:

```
https://{HOST}:9002/rest/v2/{STOREFRONT}/countries?type=billing
```

The following example shows the response that is generated from this call:

```
{
  "countries": [
    {
      "isocode": "AL",
      "name": "Albania"
    },
    {
      "isocode": "AD",
      "name": "Andorra"
    },
    {
      "isocode": "AT",
      "name": "Austria"
    },
    ...
  ]
}
```

A list of all delivery countries can be retrieved by specifying `type=shipping`, as shown in the following example:

```
https://{HOST}:9002/rest/v2/{STOREFRONT}/countries?type=shipping
```

You can also get a list of all countries defined in SAP Commerce Cloud by not specifying a type, as shown in the following example:

```
https://{HOST}:9002/rest/v2/{STOREFRONT}/countries
```

i Note

The API call `GET/deliverycountries` has been deprecated.

Countries REST API for Regions

The Countries REST API for Regions allows you to get all regions (such as states or provinces) that are associated with a specific country. This call is meant to be used to populate address forms, such as for billing or delivery.

i Note

All calls are documented in OpenAPI docs in SAP Commerce Cloud: <https://HOST:9002/rest/v2/swagger-ui.html>

Getting All Regions

The following example shows the REST API call to get all regions for a specified country. The country specified in this example is the United States:

```
https://{HOST}:9002/rest/v2/{STOREFRONT}/countries/us/regions
```

The following example shows the response that is generated from this call:

```
{
  "regions": [
    {
      "isocode": "US-AL",
      "name": "Alabama"
    },
    {
      "isocode": "US-AK",
      "name": "Alaska"
    },
    {
      "isocode": "US-AS",
      "name": "American Samoa"
    },
    ...
  ]
}
```

Order Cancellation and Return Scenarios

Step-by-step guidance on the sequences of calls that should be used in order to complete a given scenario.

Overview

The scenarios provided below represent a complete step-by-step guidance on using a given sequence of calls.

i Note

The URL examples are used for presentation purposes only. In a customized environment, you must replace the server address with your own.

Cancel an Order

Prerequisites

- A customer is registered.
- A customer has an order that is not shipped yet.
 - For detailed information on the checkout process calls, please refer to [Customer Buying Process Scenarios](#) and [Single Calls for Customer Buying Scenarios](#).

Scenario Steps

- Get the details of an order.
- Cancel an order.

Request Flow

Method	URL	Query Parameters	B
			C
GET	https://localhost:9002/rest/v2/electronics/users/current/orders/00000001	fields=DEFAULT	
POST	https://localhost:9002/rest/v2/electronics/users/current/orders/00000001/cancellation		

Method	URL	Query Parameters	Body Parameters

Return an Order

Prerequisites

- A customer is registered.
- A customer has an order that is partially shipped.
 - For detailed information on the checkout process calls, please refer to [Customer Buying Process Scenarios](#) and [Single Calls for Customer Buying Scenarios](#).
 - For more information on order shipment process with OMS, please refer to [Backoffice Customer Support Cockpit - Orders and Returns](#).

Scenario Steps

- Get the details of an order.
- Return an order.

Request Flow

Method	URL	Query Parameters	Body Parameters
			Content-Type: appli
GET	https://localhost:9002/rest/v2/electronics/users/current/orders/00000001	fields=DEFAULT	
POST	https://localhost:9002/rest/v2/electronics/users/current/orderReturns		<pre>{ "orderCode": "00000001", "returnReq": { "o": "00000001", "q": "00000001", "o": "00000001", "q": "00000001" } }</pre>

Cancel a Return Request

Prerequisites

- A customer is registered.

This is custom documentation. For more information, please visit the [SAP Help Portal](#).

- A customer has an order that is partially shipped.
 - For detailed information on the checkout process calls, please refer to [Customer Buying Process Scenarios](#) and [Single Calls for Customer Buying Scenarios](#).
 - For more information on order shipment process with OMS, please refer to [Backoffice Customer Support Cockpit - Orders and Returns](#).
- A customer has a cancellable return request.

Scenario Steps

- Get return requests.
- Get the details of a return request
- Cancel the return request.

Request Flow

Method	URL	Query Parameters	Body Param
			Content-Type
GET	https://localhost:9002/rest/v2/electronics/users/current/orderReturns	fields=DEFAULT	
GET	https://localhost:9002/rest/v2/electronics/users/current/orderReturns/00000000	fields=DEFAULT	
PATCH	https://localhost:9002/rest/v2/electronics/users/current/orderReturns/00000000		{"status"

Save Cart in OCC

The save cart functionality allows you to save and restore saved carts at a later date.

Overview

The save cart functionality is provided as a set of methods embedded within independent strategies that can be wired-up, relative to the different business requirements and the front-end implementations they are used for. These strategies can be easily extended with pre/post-hooks.

The following save cart operations are currently supported:

- Save a session cart as a saved cart
- Save specific cart IDs, for a back-end operations, as saved carts
- Display a list of saved carts
- Display the details of a saved cart
- Restore a saved cart to an active session cart

- Delete saved carts
- Clone saved carts
- Edit the name and description of a saved cart

Changes to the Web Services

In order to support the save cart functionality, the `ycommercewebservices` and `commercewebservicescommons` extensions have been extended. The following sections describe the save cart-related changes that have been made to these extensions.

SaveCartController

In the `ycommercewebservices` extension, the `SaveCartController` class has been added. This class delegates the incoming save cart calls to the implementation of the `SaveCartFacade` interface, which does the actual work.

Beans

In the `commercewebservicescommons/resources/commercewebservicescommons-spring.xml` file, you can configure the data transfer object that serves as an intermediary object between the models in the `ServiceLayer` and the web service client. The following is an example:

```
<bean>
...
<bean class="de.hybris.platform.commercewebservicescommons.dto.order.Sa
<property name="savedCartData" type="de.hybris.platform.commercewebserv
</bean>

</beans>
```

Save Cart Resource REST Calls

The following is a list of all the calls related to the save cart functionality:

- `/users/{userId}/carts/{cartId}/save`
- `/users/{userId}/carts/{cartId}/savedcart`
- `/users/{userId}/carts`
- `/users/{userId}/carts/{cartId}/flagfordeletion`
- `/users/{userId}/carts/{cartId}/restoresavedcart`
- `/users/{userId}/carts/{cartId}/clonesavedcart`

i Note

You can also use the `/users/{userId}/carts/{cartId}/save` method to update an existing saved cart by providing the name and/or description of the cart.

DTO Mapping and Response Configuration

The Mapping Data Mechanism facilitates mapping of the data between the source and destination objects.

Overview

The Mapping Data Mechanism supports fields configuration, which means you are able to provide a list of fields you want to be mapped. The Mapping Data Mechanism can be used in Web Services for mapping of the data between model objects and web services DTO or between the commerce layer data object and web services DTO. For details on Mapping Data Mechanism see: [Data Mapping Mechanism](#).

Configuring the Mapping Mechanism

The core functionality is exposed by the `DataMapper` interface which provides several methods for mapping between the source and destination objects. The `DefaultDataMapper` uses `FieldSetBuilder` and `GeneralFieldFilter` to map only specified fields of destination object. To use this Mapping Data Mechanism in web services extension you have to define the `DataMapper`, `GeneralFieldFilter` and `FieldSetBuilder` in the web context. The main configuration can be found in `/ycommercewebservices/web/webroot/WEB-INF/config/v2/dto-mappings-v2-spring.xml` file. The configuration related to `FieldSetBuilder` is described in the `Fields Configuration` section below.

```
...
<!-- DataMapper -->
<alias alias="dataMapper" name="defaultDataMapper"/>
<bean id="defaultDataMapper" class="de.hybris.platform.webservicescommons.mapping.impl.DefaultDataMapper">
    <property name="fieldSetBuilder" ref="fieldSetBuilder"/>
</bean>

<!-- Orika : Filters -->
<bean class="de.hybris.platform.webservicescommons.mapping.filters.GeneralFieldFilter">
    <property name="fieldSelectionStrategy" ref="fieldSelectionStrategy"/>
</bean>
...
```

Customizing the Mapping Mechanism

The mapping mechanism is based on Orika, a popular Java Bean mapper framework. You can use all the features that are available in supported versions of Orika by either creating custom classes or customizing the `DataMapper` implementation.

i Note

The `WsDTOMapping` annotation is something you will need during the customization process, because the `DefaultDataMapper` registers only the annotated beans in the Orika Mapper Factory. That is why you need to annotate your custom converters, mappers and filters with `WsDTOMapping`.

i Note

When you are creating a custom mapper you can also extend `AbstractCustomMapper` - this class is already annotated and additionally has support for fields selection mechanism.

Below you can find example of mappers and converters defined in `ycommercewebservices` extension in the `dto-mappings-v2-spring.xml` file.

```
...
<!-- Orika : Mappers -->
<bean class="de.hybris.platform.ycommercewebservices.mapping.mappers.AddressValidationDataMapper"
    parent="abstractCustomMapper"/>
<bean class="de.hybris.platform.ycommercewebservices.mapping.mappers.SpellingSuggestionMapper"
    parent="abstractCustomMapper"/>
<bean class="de.hybris.platform.ycommercewebservices.mapping.mappers.CCPaymentInfoDataMapper"
    parent="abstractCustomMapper"/>

<!-- Orika : Converters -->
<bean class="de.hybris.platform.ycommercewebservices.mapping.converters.StockLevelStatusConverter"/>
<bean class="de.hybris.platform.ycommercewebservices.mapping.converters.OrderStatusConverter"/>
<bean class="de.hybris.platform.ycommercewebservices.mapping.converters.ConsignmentStatusConverter"/>
<bean class="de.hybris.platform.ycommercewebservices.mapping.converters.DeliveryStatusConverter"/>
<bean class="de.hybris.platform.ycommercewebservices.mapping.converters.ProductReferenceTypeEnumConverter"/>
...
```

FieldMapper

In case you only need to map one single field to another with a different name you can use

`de.hybris.platform.webservicescommons.mapping.config.FieldMapper` class or the `fieldMapper` abstract bean. To learn how, see the configuration in the `dto-mappings-v2-spring.xml` file below:

```
...
<!-- Field Mappings : User -->
<bean parent="fieldMapper">
  <property name="sourceClass"
    value="de.hybris.platform.commerceservicescommons.dto.user.UserSignUpWsDTO"/>
  <property name="destClass"
    value="de.hybris.platform.commerceservicescommons.dto.user.RegisterData"/>
  <property name="fieldMapping">
    <map>
      <entry key="uid" value="login"/>
    </map>
  </property>
</bean>
<bean parent="fieldMapper">
  <property name="sourceClass"
    value="de.hybris.platform.commerceservicescommons.dto.user.CustomerData"/>
  <property name="destClass"
    value="de.hybris.platform.commerceservicescommons.dto.user.UserWsDTO"/>
  <property name="fieldMapping">
    <map>
      <entry key="defaultShippingAddress" value="defaultAddress"/>
    </map>
  </property>
</bean>
...
```

Fields Configuration

The `DataMapper` supports field configuration, which means you are able to provide a list of fields you want to have in the response. You can also create Field Levels, which are pre-configured sets (like BASIC, DEFAULT, FULL) to use later on while working with the mapper. The `DefaultDataMapper` uses a field set builder to parse the fields configuration. The default field set builder bean is defined in the `/ycommercewebservices/web/webroot/WEB-INF/config/v2/dto-level-mappings-v2-spring.xml`

```
...
<alias alias="fieldSetBuilder" name="defaultFieldSetBuilder"/>
<bean id="defaultFieldSetBuilder"
  class="de.hybris.platform.webservicescommons.mapping.impl.DefaultFieldSetBuilder">
  <property name="defaultRecurrencyLevel" value="4"/>
  <property name="defaultMaxFieldSetSize" value="50000"/>
  <property name="fieldSetLevelHelper" ref="fieldSetLevelHelper"/>
</bean>
<alias alias="fieldSetLevelHelper" name="defaultFieldSetLevelHelper"/>
<bean id="defaultFieldSetLevelHelper"
  class="de.hybris.platform.webservicescommons.mapping.impl.DefaultFieldSetLevelHelper">
</bean>
...
```

Field Level Definition

Field levels are a predefined set of fields, which can be defined in the Spring configuration. Default field set levels for DTO class can be found in `/ycommercewebservices/web/webroot/WEB-INF/config/v2/dto-level-mappings-v2-spring.xml` file

```
...
<bean parent="fieldSetLevelMapping">
  <property name="dtoClass"
    value="de.hybris.platform.commerceservicescommons.dto.user.UserGroupWsDTO"/>
  <property name="levelMapping">
    <map>
      <entry key="BASIC" value="membersCount,subGroups,members,uid,name"/>
    </map>
  </property>
</bean>
```

```

        <entry key="DEFAULT"
            value="membersCount,subGroups(DEFAULT),members(DEFAULT),uid,name"/>
        <entry key="FULL"
            value="membersCount,subGroups(FULL),members(FULL),uid,name"/>
    </map>
</property>
</bean>

<bean parent="fieldSetLevelMapping">
    <property name="dtoClass"
        value="de.hybris.platform.commerceservicescommons.dto.user.PrincipalWsDTO"/>
    <property name="levelMapping">
        <map>
            <entry key="BASIC" value="uid,name"/>
            <entry key="DEFAULT" value="uid,name"/>
            <entry key="FULL" value="uid,name"/>
        </map>
    </property>
</bean>
...

```

OCC Error Responses

Guidelines on how you can use the OCC error responses.

Overview

The OCC error handling mechanism is defined in the `webservicescommons` extension. The mechanism has several benefits:

- Providing common error response formats
- Mapping the response status for exceptions

The `RestExceptionHandlerResolver` class replaces the `RestHandlerExceptionHandlerResolver`. The `RestExceptionHandlerResolver` allows you to define generic error messages so that internal information is not leaked. An example of internal information is class structure.

RestHandlerExceptionHandlerResolver

i Note

The `RestHandlerExceptionHandlerResolver` is deprecated since version 2105.

The `RestHandlerExceptionHandlerResolver` class is the deprecated implementation of the `HandlerExceptionHandlerResolver` Spring interface. It is deprecated from version 2105 onward. The `RestHandlerExceptionHandlerResolver` class is located in the `webservicescommons` extension. For details see: [RestHandlerExceptionHandlerResolver](#).

Below is an example of the Spring configuration for the previous version of the `ycommercewebservices` extension.

```

<bean id="abstractRestHandlerExceptionHandlerResolverV2" abstract="true">
    <property name="webServiceErrorFactory" ref="webServiceErrorFactory" />
    <property name="messageConverters" ref="messageConvertersV2" />
</bean>

<bean id="restHandlerExceptionHandlerResolverV2" class="de.hybris.platform.webservicescommons.resolver.RestHandlerExce
    parent="abstractRestHandlerExceptionHandlerResolverV2">
    <property name="propertySpecificKey" value="ycommercewebservices"/>
    <property name="configurationService" ref="configurationService"/>
</bean>

<util:list id="exceptionResolversV2">

```

```
<ref bean="restHandlerExceptionResolverV2" />
</util:list>
```

Configuration

Configuration can be set in the properties in `ycommercewebservices/project.properties` file with the following format:

```
webservicescommons.resthandlerexceptionresolver.{extensionName}.{exceptionName}.logstack=true or false
webservicescommons.resthandlerexceptionresolver.{extensionName}.{exceptionName}.status=HTTP_STATUS_CODE
```

An example of the configuration is demonstrated here:

```
webservicescommons.resthandlerexceptionresolver.ycommercewebservices.CartAddressException.logstack=true
webservicescommons.resthandlerexceptionresolver.ycommercewebservices.CartAddressException.status=400
```

i Note

The simple class name determines the status code. The canonical class name does not determine the status code. The exception name must be unique.

RestExceptionHandlerResolver

The `RestExceptionHandlerResolver` class is the implementation of the `HandlerExceptionResolver` Spring interface. The `RestExceptionHandlerResolver` class is located in the `webservicescommons` extension. The class is used in the `ycommercewebservices` extension to handle exceptions. For further details view [RestExceptionHandlerResolver](#).

```
<bean id="restHandlerExceptionResolverV2" class="de.hybris.platform.webservicescommons.resolver.RestExceptionHandlerResolver"
      parent="wsBaseRestExceptionHandlerResolver">
  <property name="webServiceErrorFactory" ref="webServiceErrorFactory" />
  <property name="messageConverters" ref="messageConvertersV2" />
  <property name="extensionName" value="ycommercewebservices" />
</bean>

<util:list id="exceptionResolversV2">
  <ref bean="restHandlerExceptionResolverV2" />
</util:list>
```

Configuration

Configuration can be done through properties in `ycommercewebservices/project.properties` file. The newly implemented solution allows you to define properties such as `messageFormatterType` and `message`. Examples of the configuration are demonstrated below.

This example demonstrates the `CartAddressException` being forwarded as is:

```
webservicescommons.resthandlerexceptionresolver.ycommercewebservices.CartAddressException.logstack=true
webservicescommons.resthandlerexceptionresolver.ycommercewebservices.CartAddressException.status=400
webservicescommons.resthandlerexceptionresolver.ycommercewebservices.CartAddressException.messageFormatterType=
```

This example demonstrates the `CartAddressException` being covered with a `GENERIC` message:

```
webservicescommons.resthandlerexceptionresolver.ycommercewebservices.CartAddressException.logstack=true
webservicescommons.resthandlerexceptionresolver.ycommercewebservices.CartAddressException.status=400
webservicescommons.resthandlerexceptionresolver.ycommercewebservices.CartAddressException.message=Cart address
webservicescommons.resthandlerexceptionresolver.ycommercewebservices.CartAddressException.messageFormatterType=
```

WebserviceErrorFactory

The `WebserviceErrorFactory` is used in error handling mechanism to convert exceptions to `ErrorWsDTO` objects. Its default implementation `DefaultWebserviceErrorFactory` uses list of `AbstractErrorConverter` in conversion process. You can influence the way the exceptions will be converted to `ErrorWsDTO` object by adding your converter to that list in `/ycommercewebservices/web/webroot/WEB-INF/config/common/error-config-spring.xml`.

```
...
<alias alias="webServiceErrorFactory" name="defaultWebServiceErrorFactory" />
<bean id="defaultWebServiceErrorFactory" class="de.hybris.platform.webservicescommons.errors.factory.impl.D
  <property name="converters">
    <list>
      <ref bean="validationErrorConverter" />
      <ref bean="cartVoucherValidationListErrorConverter" />
      <ref bean="cartModificationDataListErrorConverter" />
      <ref bean="webServiceExceptionConverter" />
      <ref bean="exceptionConverter" />
    </list>
  </property>
</bean>
...
```

i Note

All converters are checked to verify if they support validation objects or not. It is therefore possible to have two different converters that support the same object but produce different errors.

Default Converters

By default, converters for the following types are provided:

Default Converters	
Error Converter	Supported Types
ValidationErrorConverter	org.springframework.validation.Errors
CartModificationDataListErrorConverter	de.hybris.platform.commercefacades.order.data.CartModificationDataList
CartModificationDataErrorConverter	de.hybris.platform.commercefacades.order.data.CartModificationData
WebServiceExceptionConverter	de.hybris.platform.webservicescommons.errors.converters.WebServiceExcept
ExceptionConverter	java.lang.Exception excluding descendants of de.hybris.platform.webservicescommons.errors.converters.WebServiceExcept

HTTP Message Converters

The OCC API allows converting the DTO objects to or from their text representation (either JSON or XML) used in REST calls.

OCC allows converting the DTO objects to/from their text representation (either JSON or XML) used in REST calls. For each REST call, the output data is returned as a standard DTO object. The DTO objects are regular Java classes that are automatically generated by a standard platform mechanism based on the `commercewebservicescommons-beans.xml` file. These objects can also be used as parameters. To be used in the REST calls, the DTO objects are converted to/from the text (JSON/XML) using a conversion mechanism available in the `webservicescommons` extension. The `ycommercewebservices` extension is configured to use the HTTP message converters defined in it.

```
...
<alias name="jaxbMessageConverters" alias="messageConvertersV2" />
```

...

...

```
<bean id="abstractRestHandlerExceptionResolverV2" abstract="true">
  <property name="webserviceErrorFactory" ref="webserviceErrorFactory" />
  <property name="messageConverters" ref="messageConvertersV2" />
</bean>
```

..

...

```
<bean id="oAuth2ExceptionHandlerV2" parent="oAuth2ExceptionHandler">
  <property name="messageConverters" ref="messageConvertersV2"/>
</bean>
```

..

Selecting the Response Format

By default, only the `Accept` header is enabled and used in the content negotiation for incoming OCC API requests. Content negotiation using the format query parameter and path extension is disabled in OCC API. According to the Spring Framework documentation, using path extensions is discouraged and deprecated from the 5.3 version. As an alternative, it is possible in OCC to enable a fallback mode and use a format query parameter to express acceptable response format.

To enable fallback mode add the following property to your configuration:

```
ycommercewebservices.content.negotiation.legacy=true
```

Related Information

[HTTP Message Converters](#)

WsDTO Concept

WsDTO is a data layer used by the REST API in OCC.

Concept

The WsDTO model adds stability to the REST API by removing the dependency to the commerce services data model. As a result, changes in the commerce data model do not directly affect the REST API. It also improves flexibility by introducing a mechanism that allows control of returning fields. Therefore, you now have influence on what the returning WsDTO object will look like. You can explicitly request specific fields to be filled in an object or use a predefined set of fields.

Stable API

In v1 of the REST API, services return data objects which could lead to some unexpected behaviors when fields are added or changed in the commerce services data model. Every data model change in the commerce services could influence the REST API and as a result, could affect compatibility with the service consumers. The WsDTO layer is resilient to changes in the commerce services data model to ensure API stability.

Dynamic Field Configuration

The new WsDTO layer was introduced with a new mechanism that allows you to define fields to return by API. By default there are three configurations:

- BASIC - only a basic set of fields that identify an object are filled. For example, name, code, and ID.
- DEFAULT - medium set of fields defined on the most common use cases.
- FULL - all fields are returned.

All sets can be changed or new ones can be added on the server side by providing a specific configuration. Consumers have influence on returning fields by explicitly sending them as a parameter or implicitly by sending a set name in parameter.

WsDTO Structure

WsDTO objects have been created initially as closely as possible to commerce services data objects so mapping is simple and automatically done by Orika mapper. They are created in the `commercewebservicescommons-beans.xml` file as regular bean objects.

```
...
<bean class="de.hybris.platform.commercewebservicescommons.dto.order.CartWsDTO"
      extends="de.hybris.platform.commercewebservicescommons.dto.order.AbstractOrderWsDTO">
  <property name="totalUnitCount" type="Integer"/>
  <property name="potentialOrderPromotions"
            type="java.util.List<de.hybris.platform.commercewebservicescommons.dto.product.PromotionRe
  <property name="potentialProductPromotions"
            type="java.util.List<de.hybris.platform.commercewebservicescommons.dto.product.PromotionRe
</bean>
...
```

When adding a new WsDTO object, it is recommended to also add a level mapping configuration in the `/WEB-INF/config/v2/dto-level-mappings-v2-spring.xml` file.

Differences to Commerce Services Data Objects

Despite the fact that most of the objects were defined as closely as possible to data objects there are some exceptions.

No Generic Objects

There are no generic WsDTO types. All WsDTO types are concrete objects without parameter types to avoid ambiguity of returned objects by REST services.

Wrapping Root Level Collections

All methods in the OCC REST API returns WsDTO objects which means that all collection types that could be returned by services have been wrapped to the `ListWsDTO` type. Most of the `ListWsDTO` types are just wrapping classes that have a field that is a collection type.

Collections Inside WsDTO Objects

Fields in WsDTO objects are opposite to root objects, which do not have to be wrapped in the `ListWsDTO` type, so it is permitted to define fields as collection types. For example, Lists.

Changes in API

After introducing the new WsDTO layer, the OCC REST API had to be changed too. The most critical change is that any method does not return commerce services data objects anymore. Every Data object has its counterpart in WsDTO and appropriate mapping. All OCC REST API methods in v2 have changed their signatures to return WsDTO objects.

Related Information

[DTO Mapping and Response Configuration](#)

Payment in OCC

The payment process flow calls description, including the payment details definition and card authorization.

Payment Flows

The payment procedure is a crucial part of the customer checkout flow. Within the process, two main stages may be distinguished:

1. Payment details definition.
2. Card authorization.

Submitting Payment Details Flow

In this payment flow, the two main steps described above can be achieved using the following requests :

Method	URL	Description
POST	<code>https://localhost:9002/rest/v2/{baseSiteId}/users/{userId}/carts/{cartId}/paymentdetails</code>	Defines payment details for customer.
POST	<code>https://localhost:9002/rest/v2/{baseSiteId}/users/{userId}/orders</code>	Authorizes the card and places the order.

i Note

The following restrictions apply to the use of SOP payment flow:

- This payment flow is not recommended for most use cases as it requires PCI compliance to work with SAP Commerce servers.
- This payment flow is not supported for `cisCybersource` payment provider - available in `cispayment` AddOn. If `cispayment` addOn is installed and payment provider for base store is set to `cisCybersource` then `UnsupportedRequestException` is thrown for create payment details request.

SOP Payment Flows

These payment flows allow the user to utilize the Silent Order POST credit card payment gateway. The client creates payment subscription by sending the request directly to payment provider.

There are two possible SOP payment flows. The main difference between them is in the way of creating payment details based on payment provider response:

1. In the first flow payment details are defined when client forward payment provider response to OCC. In this case, the merchant callback sets a validation flag only.
2. In the second flow payment details are defined in a merchant callback.

Required Extensions

i Note

To be able to use the SOP payment you have to install the [acceleratorwebservicesaddon AddOn](#).

This AddOn extends the OCC API with the following calls:

- `/v2/{baseSiteId}/users/{userId}/carts/{cartId}/payment/sop/request`
- `/v2/{baseSiteId}/users/{userId}/carts/{cartId}/payment/sop/response`

The `acceleratorwebservicesaddon` AddOn depends on the [acceleratorfacades](#) and [acceleratorservices](#) extensions where SOP payment functionality is defined. The `acceleratorservices` extension provides also the Payment Gateway Mock which is helpful during the development process.

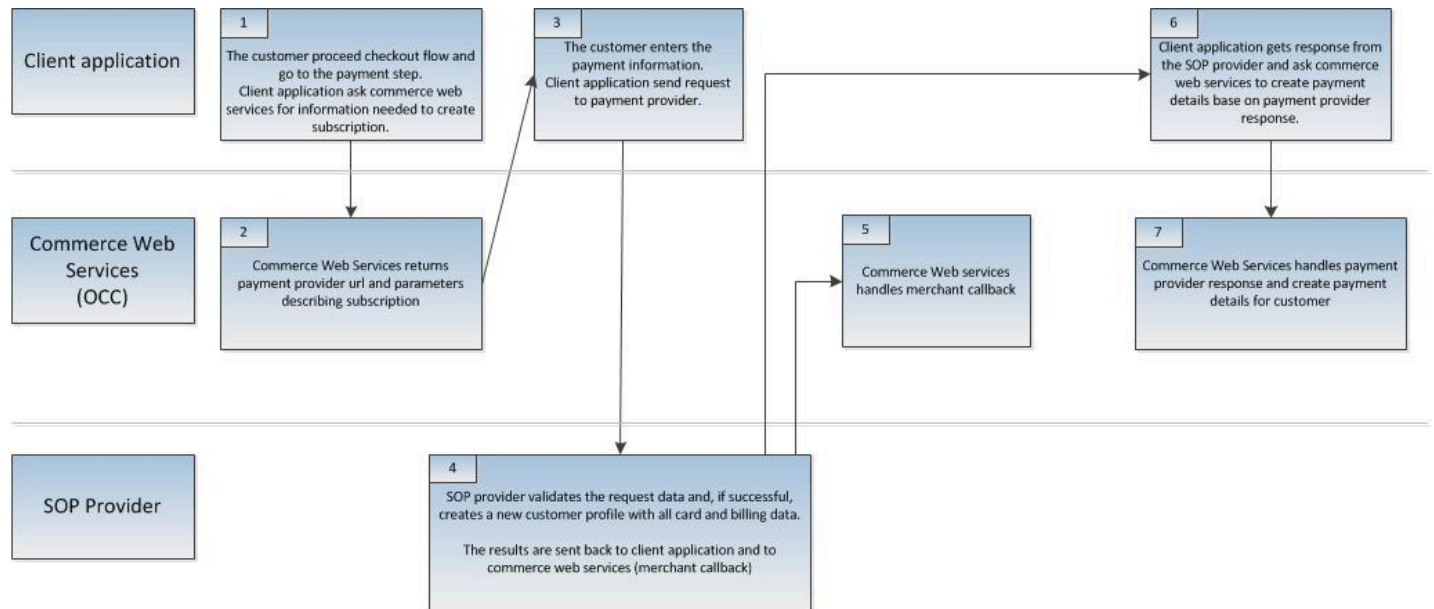
i Note

This is custom documentation. For more information, please visit the [SAP Help Portal](#).

SOP Payment Flow

The figure below provides an overview of the SOP payment flow. It presents two responses from the payment provider:

- the first response is sent to client application - data from this response is forwarded to OCC and payment details are defined.
- the second response is sent directly to OCC (merchant callback). It is then validated and subscription validation flag is set in payment details.



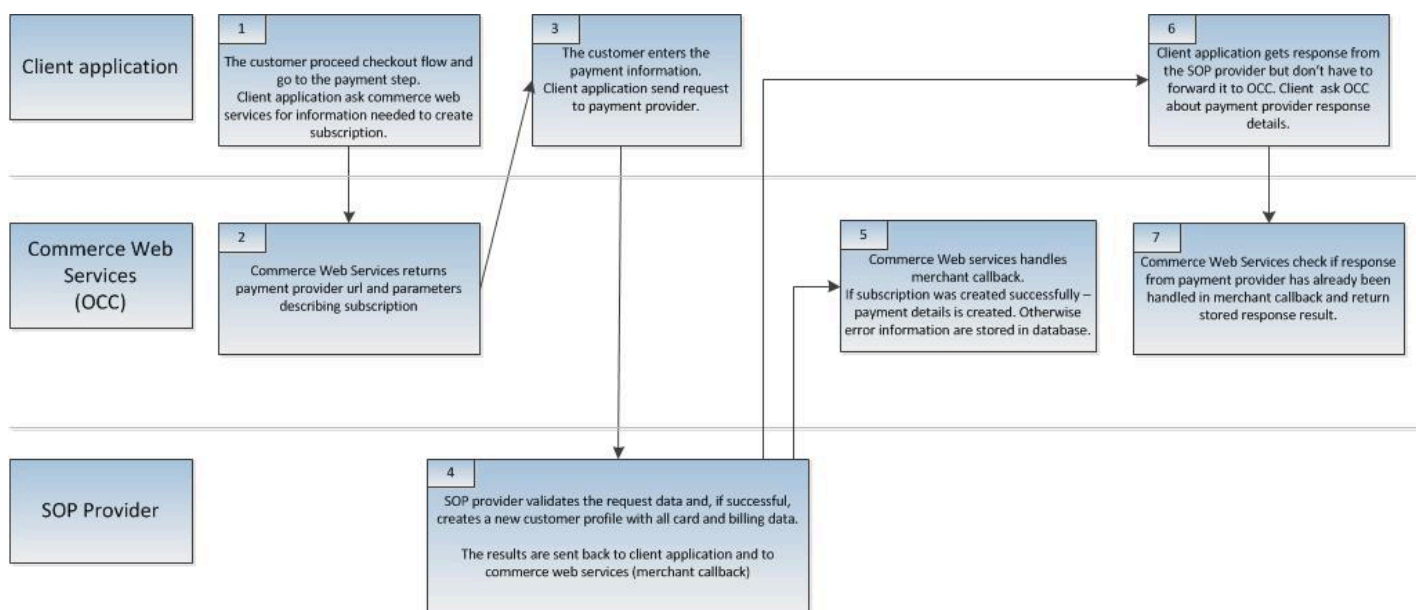
To complete create payment details step in this flow you have to send the requests listed below :

Method	URL	Description
GET	<code>https://localhost:9002/rest/v2/{baseSiteId}/users/{userId}/carts/{cartId}/payment/sop/request?responseUrl=sampleUrl</code>	Gets information needed for creating a subscription contacting directly with the payment provider
POST	postUrl got in previous request	Creates a subscription contacting directly with the payment provider.
POST	<code>https://localhost:9002/rest/v2/{baseSiteId}/users/{userId}/carts/{cartId}/payment/sop/response</code>	Handles response from payment provider and creates payment details.

Method	URL	Description
POST	https://localhost:9002/rest/v2/electronics/users/{userId}/orders	Authorizes card - this remains the same as in the case of a standard flow.

SOP Payment Flow with Extended Merchant Callback

The figure below provides an overview of an extended SOP payment flow. It shows that handling merchant callback is more complex for this flow. The OCC stores information from the payment provider to be able to inform the customer about the result of the create subscription process. If it is successful the merchant callback handler will also define the payment details for the customer.



To complete the create payment details step in this flow you have to send the requests listed below:

Method	URL	Description
GET	https://localhost:9002/rest/v2/{baseSiteId}/users/{userId}/carts/{cartId}/payment/sop/request?responseUrl=sampleUrl& extendedMerchantCallback=true	Gets information about subscription payment process. Note It is important to use the extendedMerchantCallback parameter because this URL will be used for the merchant callback.
POST	postUrl got in previous request	Creates a subscription with the payment details.
GET	https://localhost:9002/rest/v2/{baseSiteId}/users/{userId}/carts/{cartId}/payment/sop/response	Get information about the response from the merchant callback. The following values are returned: <ul style="list-style-type: none">200 OKreturn status

Method	URL	Description
		<ul style="list-style-type: none"> • 202 (receiv paym client • 400 (provi subsc infor retur
POST	<code>https://localhost:9002/rest/v2/electronics/users/{userId}/orders</code>	Authorizes ca in the case of

Removing Stored Payment Provider Responses

In this flow payment the provider responses are stored in database to allow client get it from OCC. They are automatically removed during place order process but you have to remember to remove it between sending multiple request to payment provider. For further information refer to: Checkout with SOP and extended merchant callback in [Calls Reference - v2](#). There is also a cronjob for removing old payment provider responses defined: `oldPaymentSubscriptionResultRemovalCronJob`.

Configuration Required for SOP

The table below provides description of the key properties used by `accelerator-services` extension.

Property Key	Description
<code>sop.post.url</code>	<p>This is the URL for the Silent Order Post. It is recommended that it uses a secure connection for data security.</p> <p>Example : <code>#sop.post.url=https://orderpagetest.ic3.com/hop/ProcessOrder.do</code> <code>sop.post.url=/accelerator-services/sop-mock/process</code></p>
<code>hop.cybersource.sharedSecret</code>	<p>It is key generated in your cybersource profile which signs the transaction data and is required for each transaction</p> <p>→ Tip For site-specific settings in a project that has multiple storefronts, append the name of your site.</p> <p>Example :</p> <p><code>hop.cybersource.sharedSecret.wsIntegrationTest=MIGfMA0GCSqGSIB3DQEBQUAA_Your_sha</code></p>
<code>hop.cybersource.merchantID</code>	<p>This is the CyberSource-assigned merchant ID issued when creating your account with them.</p> <p>→ Tip For site-specific settings in a project that has multiple storefronts, append the name of your site.</p> <p>Example :</p> <p><code>hop.cybersource.merchantID.wsIntegrationTest=your_merchant_id</code></p>

The table below provides a description of the key properties used by [accelerator-web-services-addon AddOn](#).

Property Key	Description
<code>webroot.commercewebservices.http</code>	This is the base URL for commerce web services. It is required to create the full merchant callback URL.
<code>webroot.commercewebservices.https</code>	<p>Example:</p> <p><code>webroot.commercewebservices.http=http://localhost:9001/rest</code> <code>webroot.commercewebservices.https=https://localhost:9002/rest</code></p>

Property Key	Description
	<p>→ Tip</p> <p>For site specific settings in project that has multiple storefronts, add the name of your site after the <code>webroot.commercewebservices</code> part.</p> <p>Example:</p> <pre>webroot.commercewebservices.wsIntegrationTest.http=http://localhost:9001/rest webroot.commercewebservices.wsIntegrationTest.https=https://localhost:9002/rest</pre>

Related Information

[ycommercewebservices Extension](#)

[acceleratorservices Extension](#)

[payment Extension](#)

Enabling and Using the Order Status Queue

The Order Status Update Queue is enabled and available by default in the `ycommercewebservices` extension. However it's not hooked to any business process responsible for order processing, because order processing is a complex mechanism and has many customizable touchpoints. Therefore it's up to the client to specify all of them and cover all possible order status changes.

If you are using the B2C Accelerator, the most interesting starting point will be to customize the order workflow (`yacceleratorfullfilmentprocess`). Another touchpoint could be the `cscockpit` and any other place that affects the order status. The sections below will guide you on how to quickly hook into order workflow to notify queue about order status change.

Hooking into Order Workflow

The Order Status Queue functionality uses the Spring Messaging System to communicate between various extensions without making them depend on each other. It uses a special message channel called `orderStatusUpdateChannel`. The first thing you need to do to use it in `yacceleratorfullfilmentprocess` is to declare this channel in the spring configuration as presented below. Modify the `yacceleratorfullfilmentprocess-spring.xml` file to introduce your changes.

```
<!--Order Status Update Queue functionality -->
<int:channel id="orderStatusUpdateChannel"/>
```

Next you have to take a look into your order process, which you will find in the `order-process.xml` file and find out where the actual change of order status takes place. When you discover the right place you can send a message from there:

```
/**
 * This action implements payment authorization using {@link CreditCardPaymentInfoModel}. Any other payment mod
 * be implemented here, or in a separate action, if the process flow differs.
 */
public class CheckAuthorizeOrderPaymentAction extends AbstractSimpleDecisionAction<OrderProcessModel>
{
    private MessageChannel productExpressUpdateChannel; // channel has to be injected here

    @Override
    public Transition executeAction(final OrderProcessModel process)
    {
        final OrderModel order = process.getOrder();
        if (order != null)
        {
            if (order.getPaymentInfo() instanceof InvoicePaymentInfoModel)
            {
```

```

        return Transition.OK;
    }
    else
    {
        for (final PaymentTransactionModel transaction : order.getPaymentTransactions())
        {
            for (final PaymentTransactionEntryModel entry : transaction.getEntries())
            {
                if (entry.getType().equals(PaymentTransactionType.AUTHORIZATION)
                    && TransactionStatus.ACCEPTED.name().equals(entry.getTransactionStatus()))
                {
                    order.setStatus(OrderStatus.PAYMENT_AUTHORIZED);
                    modelService.save(order);
                    sendOrderToChannel(order); // send a message to the order status update queue
                    return Transition.OK;
                }
            }
        }
    }
    return Transition.NOK;
}

/**
 * This method helps sending spring message to the orderStatusUpdateChannel
 */
protected boolean sendOrderToChannel(final OrderModel order)
{
    try
    {
        final Message<OrderModel> message = MessageBuilder.withPayload(order).build();
        return getOrderStatusUpdateChannel().send(message);
    }
    catch (final MessageDeliveryException exception)
    {
        LOG.warn("Probably there is no listener for orderStatusUpdateChannel", exception);
    }
    return false;
}

/* ... */
}

```

Inject the message channel into the proper Action in the `order-process-spring.xml` file.

```

<bean id="checkAuthorizeOrderPaymentAction" class="de.hybris.platform.yacceleratorfulfilmentprocess.actions.ord
    <property name="orderStatusUpdateChannel" ref="orderStatusUpdateChannel"/>
</bean>

```

Enabling Cron Jobs

The Order Status Update Queue is an in-memory structure, which has to be cleaned from time to time. The `OrderStatusUpdateCleanerJob` is predefined for you but not active by default.

i Note

The `orderStatusUpdateCleanerJob` settings are loaded together with the `ycommercewebseervices` extension project data. If the project data is not loaded, you need to define the cronjob based on the `orderStatusUpdateCleanerJob` job definition.

To make it running open theBackoffice Administration Cockpit and navigate to ►System► Background Processes ► CronJobs ► Select the OrderStatusUpdateCleanerJobfrom the list and perform the following steps:

- Set the Enabled flag to True for the OrderStatusUpdateCleanerJob.

order

Search

orderStatusUpdateCleanerJob : orderStatusUpdateCleanerCronJob - UNKNOWN - UNKNOWN

Log

Task

Run as

Time Schedule

System Recovery

Administration

Code

Current status

Job definition

orderStatusUpdateCleanerCronJob

NEW

orderStatusUpdateCleanerjob

Timetable

Last start time

Enabled ?

Daily at 03:00:00

True

False

You can specify one or multiple time slots to run this task in, or you can run the task immediately.

Trigger ?

Daily at 03:00:00 - Fri Oct 14 03:00:00 CEST 2016

Create new Trigger

- Set the Active flag to True for the trigger.

Edit item Daily at 03:00:00 - Fri Oct 14 03:00:00 CEST 2016



PK

8796093383158

Type

Trigger

Time created

Oct 13, 2016 10:05:56 AM



Time modified

Oct 13, 2016 10:09:05 AM



Owner



Last changes



Next Activation time

Oct 14, 2016 3:00:00 AM



Active

☐ True ☒ False

Rest Resource

The endpoint for accessing order status update can be found at the feeds controller:

GET v2/{site}/feeds/orders/statusfeed?timestamp=2013-12-12T12%3A00%3A00Z

the endpoint is available for the role TRUSTED_CLIENT and returns only the elements for the specified basesite updated after the provided timestamp.

Related Information

[Enabling the Product Express Update](#)

Enabling Language Fallback with OCC

Ensure proper language fallback in cases where the defined language is not available for the requested data.

Context

When requesting data from the OCC interface with a URL language parameter, if there is no data in the requested language, empty content may be returned instead of the expected fallback language data. For instance, if German has English as its fallback language, when requesting product data with the URL parameter `&lang=de`, and product has `{name[de]=null and name[en]=test}`, then empty may be returned for product name.

In the Accelerator storefront, when visiting pages, the following two session attributes should be set to TRUE to enable language fallback.


```
getSessionService().setAttribute(LocalizableItem.LANGUAGE_FALLBACK_ENABLED, Boolean.TRUE);
getSessionService().setAttribute(AbstractItemModel.LANGUAGE_FALLBACK_ENABLED_SERVICE_LAYER, Boolean.TRUE);
```

In OCC, these two attributes are not set by default. However, you can set them in custom implementation to enable language fallback based on your actual requirements. The setting can also be achieved by injecting `I18NService` and call `i18NService.setLocalizationFallbackEnabled(true)`.

Enabling Fallback with an occaddon

Follow this process for OCC addons based on `ycommercewebservices`.

Procedure

Set the required parameters to true in any preferred place before relevant data is retrieved. For instance, the following example shows `de.hybris.platform.ycommercewebservices.filter.SessionLanguageFilter.doFilterInternal(HttpServletRequest, HttpServletResponse, FilterChain)`:

```
@Autowired
private I18NService i18NService;

@Override
protected void doFilterInternal(final HttpServletRequest request, final HttpServletResponse response,
    final FilterChain filterChain) throws ServletException, IOException
{
    getContextInformationLoader().setLanguageFromRequest(request);
    i18NService.setLocalizationFallbackEnabled(true);

    filterChain.doFilter(request, response);
}
```

Enabling Fallback with an occ extension

Follow this process for OCC extensions based on `yocc`.

Context

OCC web services implemented using the `yocc` template depend on `commercewebservices`. In `commercewebservices` the `SessionLanguageFilter` filter definition is in the following file:

```
/commercewebservices/web/webroot/WEB-INF/config/v2/filter-config-v2-spring.xml
```

The spring bean of `SessionLanguageFilter` is `commerceWebServicesSessionLanguageFilterV2`, and it is assembled in `commerceWebServicesFilterChainListV2`, which is then used as the filter chain.

```
<bean id="commerceWebServicesSessionLanguageFilterV2" class="de.hybris.platform.commercewebservices.core.filter
  <property name="contextInformationLoader" ref="wsContextInformationLoaderV2" />
</bean>

<alias name="defaultCommerceWebServicesFilterChainListV2" alias="commerceWebServicesFilterChainListV2" />
<util:list id="defaultCommerceWebServicesFilterChainListV2">
  ...
  <!-- Matching filters -->
  <ref bean="commerceWebServicesEurope1AttributesFilterV2" />
  <ref bean="commerceWebServicesSessionLanguageFilterV2" />
  <ref bean="commerceWebServicesSessionCurrencyFilterV2" />
  <ref bean="cartMatchingFilter" />
```

```
</util:list>
```

Procedure

1. Override `WebServicesSessionLanguageFilterV2` in your customized occ extension by creating your own implementation to replace `SessionLanguageFilter`.

Both subclass and new class can be used, as this is a Spring bean redefinition, as in the following example.

```
<bean id="commerceWebServicesSessionLanguageFilterV2" class="customized_filter_implementation">
  <property name="contextInformationLoader" ref="wsContextInformationLoaderV2" />
</bean>
```

2. You can manipulate the filter list in `commerceWebServicesFilterChainListV2` using Spring. It can be referenced to customize the filter chain, as in the following example from `cmsocc`:

`/cmsocc/resources/occ/v2/cmsocc/web/spring/cmsocc-web-spring.xml`

```
<bean depends-on="commerceWebServicesFilterChainListV2" parent="listMergeDirective">
  <property name="add" ref="cmsPreviewTicketFilter" />
  <property name="afterBeanNames">
    <list value-type="java.lang.String">
      <value>commerceWebServicesSessionLanguageFilterV2</value>
    </list>
  </property>
  <property name="beforeBeanNames">
    <list value-type="java.lang.String">
      <value>0ccConsentLayerFilter</value>
      <value>cx0ccPersonalizationFilter</value>
      <value>cartMatchingFilter</value>
      <value>guestRoleFilterV2</value>
    </list>
  </property>
</bean>
```

Enabling Interactive OCC REST API Documentation

OAuth clients need to be defined and authorized to enable the interactive OCC REST API documentation.

Context

Interactive OCC REST API Documentation is currently supported in B2C Accelerator.

Procedure

1. Register a user on your storefront.
2. Open the ImpEx Import page in SAP Commerce Administration Console: <https://localhost:9002/console/impex/import>.
3. Copy the following ImpEx data:

```
INSERT_UPDATE OAuthClientDetails;clientId[unique=true] ;resourceIds ;scope ;authorizedGrar
;client-side ;hybris ;basic ;implicit,clier
;mobile_android ;hybris ;basic ;authorization
```

This ImpEx data provides you with a user called `mobile_android`, and a corresponding password, `secret`. This data is used to enable the interactive calls in the OCC REST API documentation.

4. Paste the ImpEx data into the **Import content** window, then click **Import content**.
5. Copy the following ImpEx data:

```
INSERT_UPDATE OAuthClientDetails;clientId[unique=true] ;resourceIds ;scope ;authorizedGrar
;trusted_client ;hybris ;extended ;authorization
```

This ImpEx data provides you with a user called `trusted_client` with a corresponding password. This data is used to enable the interactive calls in the OCC REST API documentation that require extended permissions.

6. Paste the ImpEx data into the **Import content** window, then click **Import content**.

7. Open the interactive OCC REST API documentation in your web browser by accessing the following link: `http://<hostname>:<port>/rest/v2/swagger-ui.html`.

For example, if you have a storefront installed on your local machine, you can access the OCC REST API documentation with the following link: `https://localhost:9002/rest/v2/swagger-ui.html`.

8. Click **Authorize** at the top of the OCC REST API documentation web page.
9. In the **Available authorizations** window that appears, enter the credentials of your storefront user in the **Username** and **Password** fields.
10. In the **Type** drop-down list, select **Basic auth**.
11. In the **ClientId** field that appears, enter `mobile_and_roid`, and in the **Secret** field, enter `secret`.
12. Select the **basic** checkbox and click **Authorize**.

The steps that follow are for authorizing the `trusted_client` user.

13. Click **Authorize** at the top of the OCC REST API documentation web page.
14. In the **Available authorizations** window that appears, scroll down to the second **OAuth2.0** section, then select **Basic auth** from the **Type** drop-down list.
15. In the **ClientId** field that appears, enter `trusted_client`, and in the **Secret** field, enter `secret`.
16. Select the **extended** checkbox and click **Authorize**.

Making REST calls in the Interactive OCC REST API Documentation

The interactive OCC REST API documentation for commerce web services allows you to try out REST API calls directly in the documentation page. If you make changes to your storefront, the changes are reflected in the response body of any REST API calls that you make.

Context

OAuth clients need to be defined and authorized to enable the interactive OCC REST API documentation. For more information, see [Enabling Interactive OCC REST API Documentation](#).

Procedure

1. Install B2C Accelerator.
2. Open the interactive OCC REST API documentation in your web browser by accessing the following link: `http://<hostname>:<port>/rest/v2/swagger-ui.html`.
For example, if you have a storefront installed on your local machine, you can access the OCC REST API documentation with the following link: `https://localhost:9002/rest/v2/swagger-ui.html`

3. Click on any controller. For example, click on the **Carts** controller.

All of the available methods for that controller appear.

4. Click on any method. For example, click on the **Get** method of the **Carts** controller.

All of the information related to that method appears.

5. In the **baseSiteId** field, enter the name of your storefront, such as `electronics` or `apparel-uk`.
6. In the **userId** field, enter the username of the user that you registered in your storefront.

This user is created as part of the procedure for [Enabling Interactive OCC REST API Documentation](#).

i Note

Not all methods require a `userId`. For example, the methods for the **Promotions** controller only require the `baseId`.

7. Click **Try it out!**

The REST call is made, and the response is displayed below the **Try it out!** button.