OCC Reference

Generated on: 2024-12-03 10:44:44 GMT+0000

SAP Commerce | 2205

Public

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

Omni Commerce Connect

The Omni Commerce Connect (OCC) API exposes a broad set of commerce and data services. It enables you to integrate SAP Commerce functionality anywhere in your application landscape.

Features	Architecture	Implementation
		
Enabling Interactive OCC REST API Documentation OCC Calls Security	OCC Extension Architecture OCC AddOns Architecture	OCC API Implementation OCC API v1 Reference

OCC Extension Architecture

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

OCC AddOns Architecture

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

OCC Calls Security

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

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.

OCC API v1 Reference

Version one of the OCC API offers stateful interaction with SAP Commerce functionality.

Enabling Interactive OCC REST API Documentation

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

OCC Extension Architecture

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

OCC extensions depend on `commercewebservices`, 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 `commercewebservices`. 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
`/acceleratoraddon`		There is no `acceleratoraddon` directory in (directory mainly migrates to the new `/occ/v2`

OCC addon directory structure (old)	OCC extension directory structure (new)	Description
		`/resources` directory of OCC extension. As a new OCC extension name. For example, `xyzoc`
`/acceleratoraddon/web/src`	`/src`	REST Controller classes and other related class OCC extension.
`/acceleratoraddon/web/webroot/WEB-INF/messages`	`/resources/occ/v2/<extension_name>/messages`	Localized messages of an OCC extension are lo
`/acceleratoraddon/web/webroot/WEB-INF/lib`	`/lib`	There is no `WEB-INF/lib` directory in OCC ex main `/lib` directory. The `/lib` directory conta `/acceleratoraddon/web/webroot/WEB`
`/resources/<addon_name>/import`	`/resources/impex`	ImpEx files in OCC extension aren't loaded auto configuration convention. For more information Project Data.
`/resources/<addon_name>/web/spring`	`/resources/occ/v2/<extension_name>/web/spring`	Spring bean definitions of OCC extension locat `/resources/occ/v2/<extension_name` `web-spring.xml` file are imported into the w

OCC Extensions Are Not Web Extensions

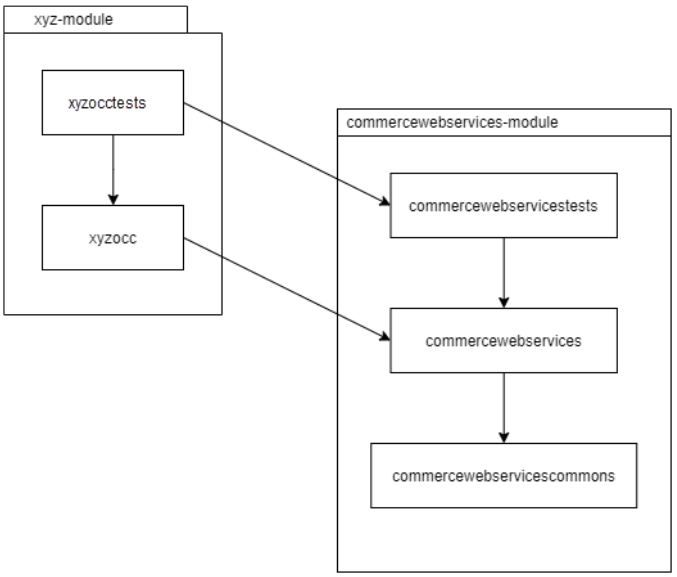
An OCC extension isn't a web extension and has no web context. It extends the functionality of OCC by adding new REST controllers, Spring beans, and localized messages. REST controller classes and other OCC extension classes are located in the main `/src` directory. In the Spring XML configuration of the `commercewebservices` 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"/>
```

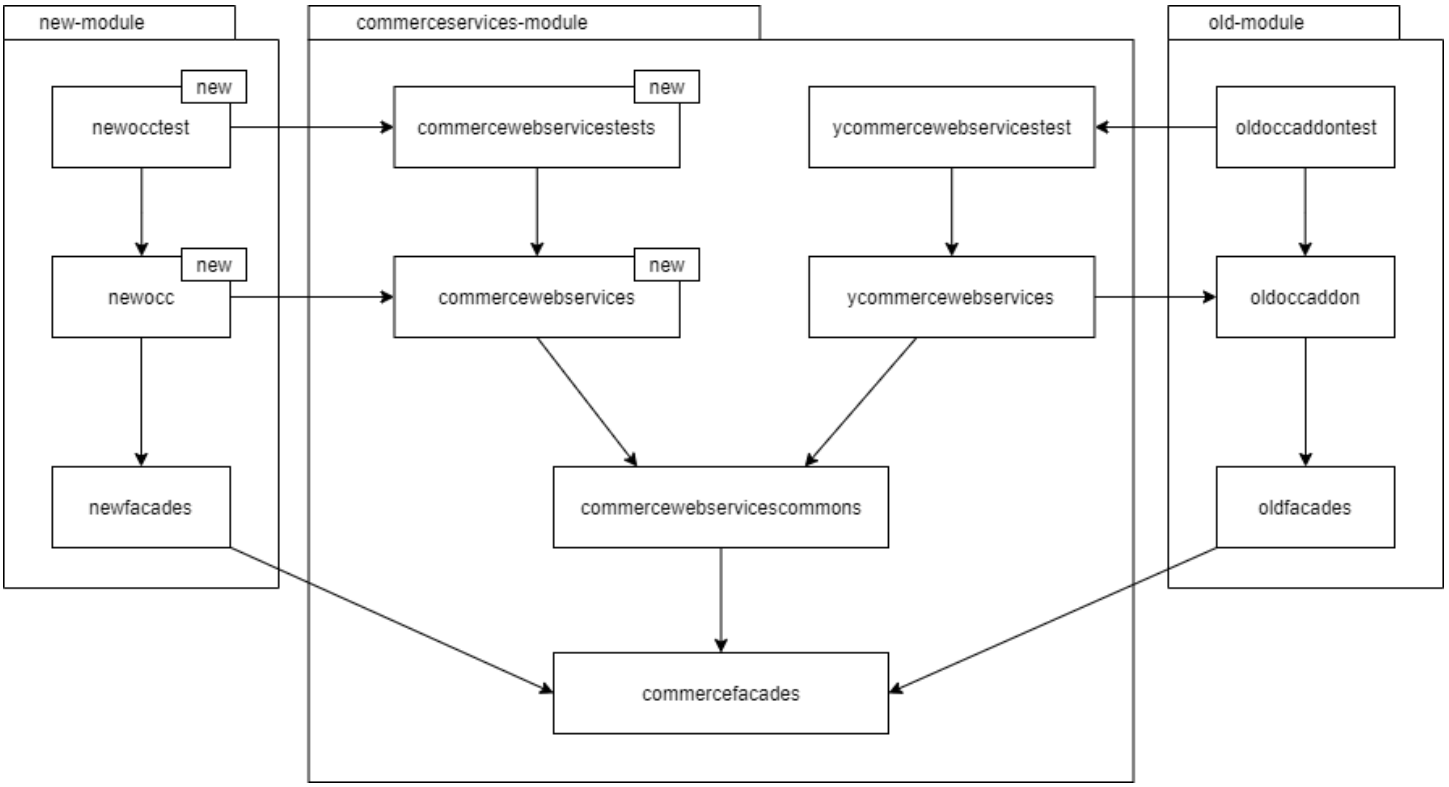
The XML configuration file import statement containing bean definitions must meet the following requirements:

- 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 ""C:\
```

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

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

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 customs steps. When the script is run, the filenames of the default steps and customs 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.nev
```

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.

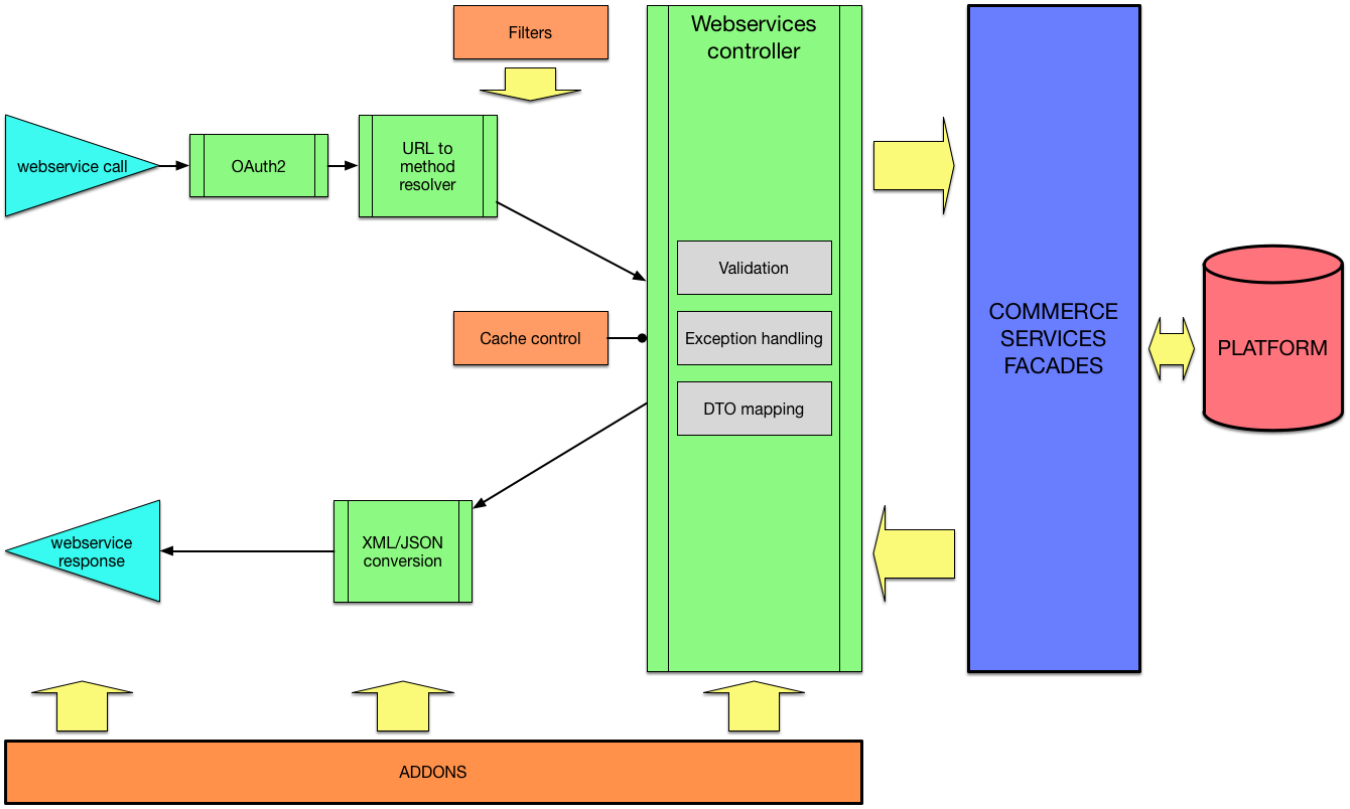
OCC provides a consistent way for web services to communicate with the platform commerce layer. Its main purpose is to expose existing commerce layer functionality to the world of web services. OCC web services are as transparent as possible, but contain some additional elements such as a separate cache,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

Commerce Services Extensions

The Commerce Services module contains following extensions:

- `commercewebservicescommons` extension
- `ycommercewebservices` extension
- `ycommercewebservicestest` extension

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



ycommercewebservices Extension

This is the main template extension of the OCC module. The most important part of this extension is advanced webservices 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.ycommercewebservices.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.

ycommercewebservicestest 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 `ycommercewebservicestest` 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.

> **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.

Overview

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.

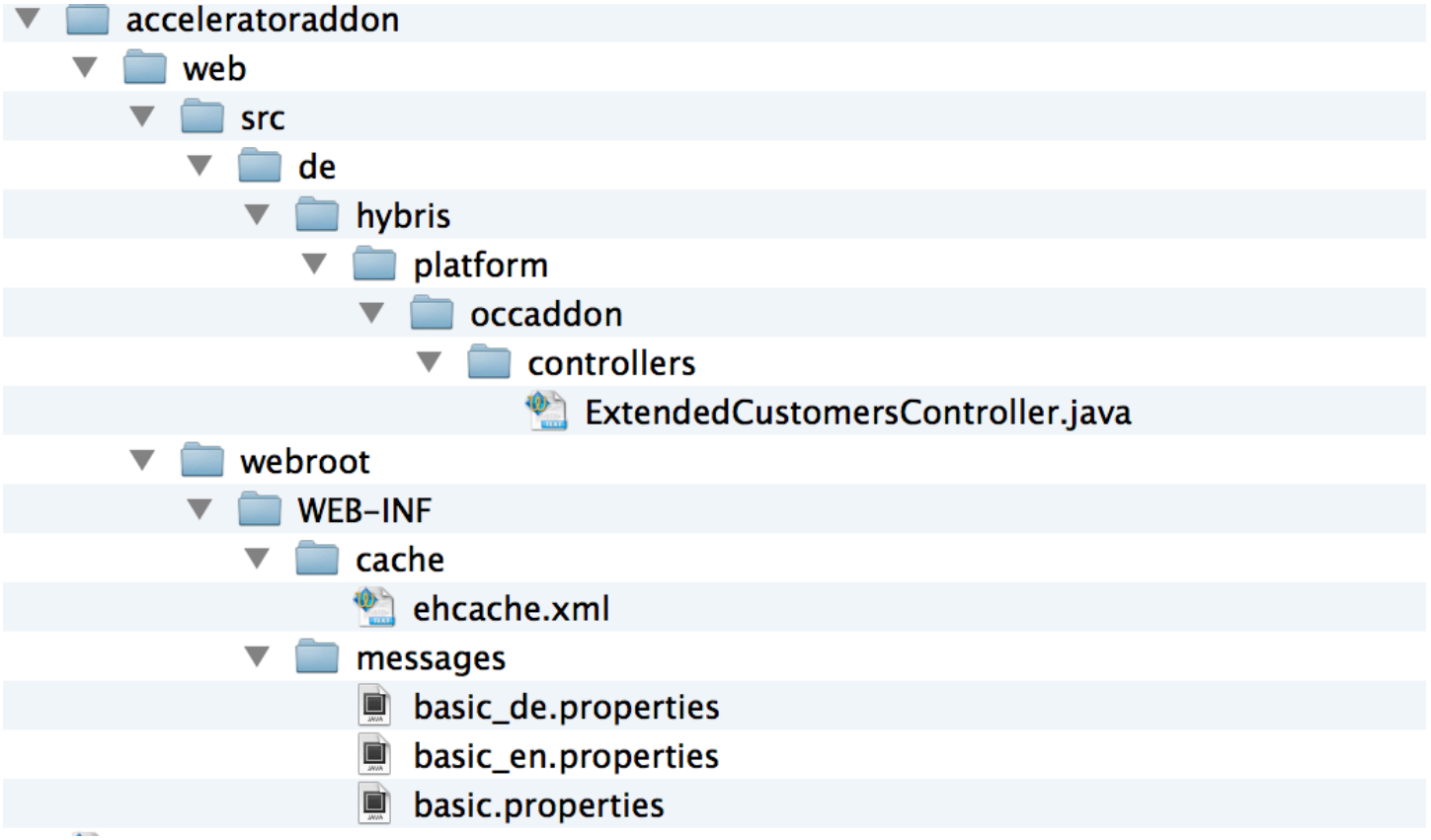
AddOn Structure

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
`<addonname>-web-spring.xml`	This file should be created in the resource directory, for example `resource\occaddon\web\spring\occaddon-web-spring.xml`. Also, it should contain the AddOn web context which is added to the OCC web services web context.
`project.properties.template`	This file should be created so you can use the AddOn installation script. It is a template for the project.properties file that holds configuration properties. These properties are created during the installation process.

Directory/File name	Remarks
	Note For details on installing an AddOn for a storefront refer to Installing an AddOn for a Specific Storefront
`\acceleratoraddon` directory	The folder's structure mirrors the structure of folders that contain the web components of a regular extension.

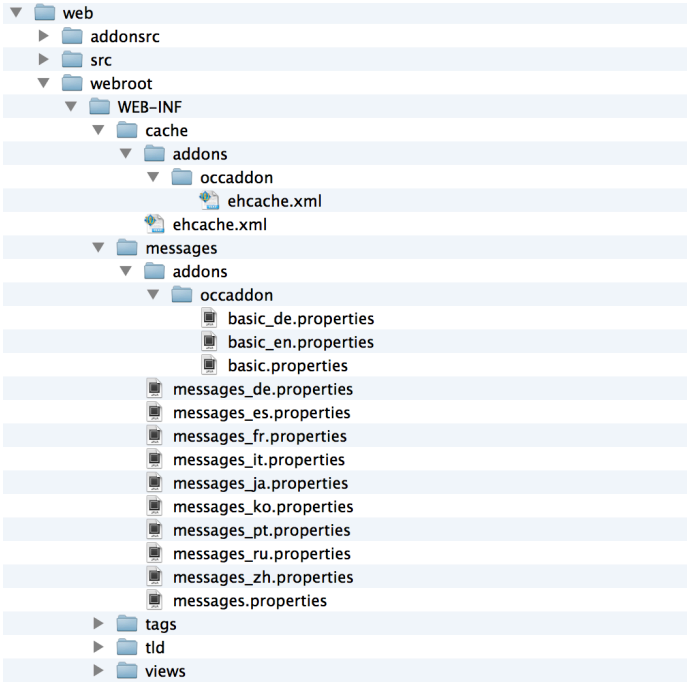
The structure of an AddOn looks as follows:



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
`\web\addonsrc`	Contains the source code.
`\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 sttructure 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.

> **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.

> **Note**
>
> If the controller is not created in the `<addonPackage>.controllers` package, the spring web context has to be adjusted as described in the CreatingtheWebSpringContext 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="ycommercewebservices"
    ```

    b. For the `occaddon` and `mycommercewebservices` the command looks like the following:

    ```
    ant addoninstall -Daddonnames="occaddon" -DaddonStorefront.ycommercewebservices="mycommercewebservices
    ```

2. Once the script has completed its work successfully, rebuild the system.

Related Information

Generating Beans and Enums

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" />
        <!-- commerceWebservices 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.PlatformFilterChain">
        <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" />
        <!-- commerceWebservices 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, incuding 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?co
  ```

  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

Access depends on roles granted by OAuth2.

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 `webservicescommons` 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 modyfying wsCacheManagerL

    <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.commercewebservicescommons.cache.TenantAwareEhCacheManagerF
        <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.CommerceCacheKe
        <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,#productCode
```

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 you `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.gbasket)
- 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/51ecf334a103df146a85b486af01aad57df4efa0/ent
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

Method	URL
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	Body P Conten
POST	https://localhost:9002/authorizationserver/oauth/token		client_i
POST	https://localhost:9002/rest/v2/electronics/users		login=c &firstN
POST	https://localhost:9002/authorizationserver/oauth/token		client_i &grant_
POST	https://localhost:9002/rest/v2/electronics/users/current/addresses	fields=id	firstNa

Method	URL	Query Parameters	Body P Conten
GET	https://localhost:9002/rest/v2/electronics/products/search	query=tripod	
GET	https://localhost:9002/rest/v2/electronics/products/3429337		
POST	https://localhost:9002/rest/v2/electronics/users/current/carts		
POST	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/entries		code=3
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/addresses/delivery		address
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymodes		
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymode		delivery
POST	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/paymentdetails		accoun 16-19&b

Method	URL	Query Parameters	Body P Conten
POST	https://localhost:9002/rest/v2/electronics/users/current/orders		cartId=
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

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	

Method	URL	Query Parameters
POST	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/entries	fields=statusCode,quantity,e
PATCH	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/entries/0	
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,pickupItemsQua pickupOrderGroups(entries(I
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,pickupItems(pickupOrderGroups(entri
GET	`https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/consolidate`	

Method	URL	Query Parameters
POST	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/consolidate	
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	

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 s 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

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.

Request Flow

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

Method	URL	Query Parameters
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,pickupItem pickupOrderGroups(ent deliveryItemsQuantity,
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: **applic**
POST	https://localhost:9002/rest/v2/electronics/users/current/addresses	firstName=John&lastN
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/addresses/delivery	addressId=879615878
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=premiu
POST	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/paymentdetails	accountHolderName=J 16-19&billingAddress.p

Method	URL	Body Parameters
		Content-Type: **applic**
POST	https://localhost:9002/rest/v2/electronics/users/current/orders	cartId=00012001&secu

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

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 Parameters
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&fields=
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

ⓘ 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	Body
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	Body
GET	`https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/payment/sop/request`		resp orde
POST	***postUrl*** got in previous request		***para*** card
POST	`https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/payment/sop/response`		para
POST	`https://localhost:9002/rest/v2/electronics/users/current/orders`		cartI
PUT	`https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/addresses/delivery`		addr
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

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
GET	https://localhost:9002/rest/v2/electronics/users/current/addresses		
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	fields=BASIC	
PUT	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/deliverymode		c
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/payment/sop/request		e c
POST	*payment provider Url* (postUrl got in previous request)		p c
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/payment/sop/response		
	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	*payment provider Url*		p c
GET	https://localhost:9002/rest/v2/electronics/users/current/carts/00012001/payment/sop/response		
POST	https://localhost:9002/rest/v2/electronics/users/current/orders		c

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`
POST	`https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efa0/`
POST	`https://localhost:9002/authorizationserver/oauth/token`
PUT	`https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efa0/`

Method	URL
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efa0/
GET	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efa0/
PUT	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efa0/
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efa0/
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/orders

Method	URL
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

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

Method	URL
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efa0/ent
POST	https://localhost:9002/authorizationserver/oauth/token
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efa0/ema
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efa0/add
GET	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efa0/del
PUT	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efa0/del

Method	URL
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efa0/pay
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/orders
GET	https://localhost:9002/rest/v2/electronics/users/anonymous/orders/51ecf334a103df146a85b486af01aad57df4efa0

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/51ecf334a103df146a85b486af01aad57df4efa0/ent
GET	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efa0
GET	https://localhost:9002/authorizationserver/oauth/token
POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efa0/vou
GET	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad57df4efa0
POST	https://localhost:9002/rest/v2/electronics/users
POST	https://localhost:9002/authorizationserver/oauth/token

Method	URL
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/02017b6dad8834ba6f3fd26eb9ab4f270e9b1858/ent

Method	URL
GET	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/02017b6dad8834ba6f3fd26eb9ab4f270e9b1858
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

Related Information

RESTful Implementation in v2
Response Configuration in v2
Single Calls for Customer Buying Scenarios

Single Calls for Customer Buying Scenarios

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

Single Calls

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

ℹ Note

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

ℹ Note

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

Authorization

Main Topic	Method	URL	Query Parameters	Body Parameters Content-Type: `application/x-www-for`
New Customer Registration	POST	`https://localhost:9002/authorizationserver/oauth/token`		client_id=$CLIENT_ID$&client_secret=$CL
	POST	`https://localhost:9002/rest/v2/electronics/users`		login=john.doe@mail.com&password=your_ &firstName=John&lastName=Doe&titleCod
Get Customer Token	POST	`https://localhost:9002/authorizationserver/oauth/token`		client_id=$CLIENT_ID$&client_secret=$CL &grant_type=password&username=john.do

Customer Profile Management

Main Topic	Method	URL	Query Param
Authorization	POST	`https://localhost:9002/authorizationserver/oauth/token`	
Address management	GET	`https://localhost:9002/rest/v2/electronics/users/current/addresses`	fields=addres
	POST	`https://localhost:9002/rest/v2/electronics/users/current/addresses`	

Main Topic	Method	URL	Query Param
	PATCH	https://localhost:9002/rest/v2/electronics/users/current/addresses/8796158590999	
Payment Management	GET	https://localhost:9002/rest/v2/electronics/users/current/paymentdetails	saved=true&f
	GET	https://localhost:9002/rest/v2/electronics/users/current/paymentdetails/8796191359018	fields=id,acco expiryYear,exp
	DELETE	https://localhost:9002/rest/v2/electronics/users/current/paymentdetails/8796191359018	
Customer Profile Management	PATCH	https://localhost:9002/rest/v2/electronics/users/current	
Password Management	POST	https://localhost:9002/occ/v2/electronics/users/current/password	None

Customer Profile Management (Using Customer Manager)

Main Topic	Method	URL	Query Parameters
Authorization	POST	https://localhost:9002/authorizationserver/oauth/token	
Address Management	GET	https://localhost:9002/rest/v2/electronics/users/current/addresses	fields=DEFAU
	POST	https://localhost:9002/rest/v2/electronics/users/current/addresses	

Main Topic	Method	URL	Query Parameters
	PATCH	https://localhost:9002/rest/v2/electronics/users/current/addresses/8796158590999	
Payment Management	GET	https://localhost:9002/rest/v2/electronics/users/current/paymentdetails	saved=true
	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	
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

Main Topic	Method	URL	Parameters
	GET	https://localhost:9002/rest/v2/electronics/products/search	query=camera&pageSize=40&fields=products(name,sur
	GET	https://localhost:9002/rest/v2/electronics/products/search	query=camera:relevance:category:575& fields=products(BASIC),pagination(DEFAULT)
	GET	https://localhost:9002/rest/v2/electronics/products/search	query=camera:relevance:category:575:brand:brand_10
	GET	https://localhost:9002/rest/v2/electronics/products/search	query=camera:relevance:category:575:brand:brand_10:pr

Main Topic	Method	URL	Parameters

Future Stock Availability

Main Topic	Method	URL	Parameters
Future Stock Availability	GET	https://localhost:9002/rest/v2/electronics/users/current/futureStocks	productCodes=489702,4897
	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
	GET	https://localhost:9002/rest/v2/electronics/products/1382080	
	GET	https://localhost:9002/rest/v2/electronics/products/1382080/stock	location=Tokio&fields=stores(name),pagination(

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

Main Topic	Method	URL
Cart Management	POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad5
	POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad5
	POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad5
	PATCH	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad5
	PATCH	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad5
	PUT	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad5

Main Topic	Method	URL
	DELETE	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad5
	GET	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad5
Cart Validation	POST	https://localhost:9002/rest/v2/electronics/users/anonymous/carts/51ecf334a103df146a85b486af01aad5

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,created,deli

Store Search, Display Store Details

Main Topic	Method	URL	Parameters	Des
Store Search	GET	`https://localhost:9002/rest/v2/electronics/stores`	query=Tokio&fields=stores(BASIC),pagination(BASIC)	Sea for tha nea loc spe by par (in cas Tok Ret of s
Display Store Details	GET	`https://localhost:9002/rest/v2/electronics/stores/Nakano`	fields=DEFAULT	Get det sto ide 'Na

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 withdrawl

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, based on
            "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.

> **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	Body Pa Content
GET	https://localhost:9002/rest/v2/electronics/users/current/orders/00000001	fields=DEFAULT	

Method	URL	Query Parameters	Body Pa Content

POST	https://localhost:9002/rest/v2/electronics/users/current/orders/00000001/cancellation		{
			"
			
			]
			}

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: application
GET	https://localhost:9002/rest/v2/electronics/users/current/orders/00000001	fields=DEFAULT	

POST	https://localhost:9002/rest/v2/electronics/users/current/orderReturns		{
			"orderCode": "0
			"returnRequestE
			{
			"orderE
			"quanti
			},
			{
			"orderE
			"quanti
			}
			]
			}

Cancel a Return Request

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.
- 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 Parameters Content-Type: ap
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":"CA`

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 `commercewebservicecommons` 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 `commercewebservicecommons/resources/commercewebservicecommons-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.SaveCa
<property name="savedCartData" type="de.hybris.platform.commercewebservices
</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`

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 Orica 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.commercewebservicescommons.dto.user.UserSignUpWsDTO"/>
        <property name="destClass"
                value="de.hybris.platform.commercefacades.user.data.RegisterData"/>
        <property name="fieldMapping">
            <map>
                <entry key="uid" value="login"/>
            </map>
        </property>
    </bean>
    <bean parent="fieldMapper">
        <property name="sourceClass"
                value="de.hybris.platform.commercefacades.user.data.CustomerData"/>
        <property name="destClass"
                value="de.hybris.platform.commercewebservicescommons.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.commercewebservicescommons.dto.user.UserGroupWsDTO"/>
        <property name="levelMapping">
            <map>
                <entry key="BASIC" value="membersCount,subGroups,members,uid,name"/>
                <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.commercewebservicescommons.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 `RestExceptionResolver` class replaces the `RestHandlerExceptionResolver`. The `RestExceptionResolver` allows you to define generic error messages so that internal information is not leaked. An example of internal information is class structure.

RestHandlerExceptionResolver

Note

The `RestHandlerExceptionResolver` is deprecated since version 2105.

The `RestHandlerExceptionResolver` class is the deprecated implementation of the `HandlerExceptionResolver` Spring interface. It is deprecated from version 2105 onward. The `RestHandlerExceptionResolver` class is located in the `webservicescommons` extension. For details see: RestHandlerExceptionResolver.

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

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

<bean id="restHandlerExceptionResolverV2" class="de.hybris.platform.webservicescommons.resolver.RestHandlerExceptior
    parent="abstractRestHandlerExceptionResolverV2">
    <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
```

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.

RestExceptionResolver

The `RestExceptionResolver` class is the implementation of the `HandlerExceptionResolver` Spring interface. The `RestExceptionResolver` class is located in the `webservicescommons` extension. The class is used in the `ycommercewebservices` extension to handle exceptions. For further details view RestExceptionResolver.

```
<bean id="restHandlerExceptionResolverV2" class="de.hybris.platform.webservicescommons.resolver.RestExceptionResolve
      parent="wsBaseRestExceptionResolver">
    <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 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=FORWA
```

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 excep
webservicescommons.resthandlerexceptionresolver.ycommercewebservices.CartAddressException.messageFormatterType=GENER
```

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.Defaul
        <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.WebserviceException`
`ExceptionConverter`	`java.lang.Exception excluding descendants of de.hybris.platform.webservicescommons.errors.converters.WebserviceException`

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` extesion. 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="oAuth2ExceptionRendererV2" parent="oAuth2ExceptionRenderer">
        <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&lt;de.hybris.platform.commercewebservicescommons.dto.product.PromotionResultW
        <property name="potentialProductPromotions"
                  type="java.util.List&lt;de.hybris.platform.commercewebservicescommons.dto.product.PromotionResultW
    </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, incuding 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	`https://localhost:9002/rest/v2/{baseSiteId}/users/{userId}/carts/{cartId}/paymentdetails`	Defines payment details for customer.
POST	`https://localhost:9002/rest/v2/{baseSiteId}/users/{userId}/orders`	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.

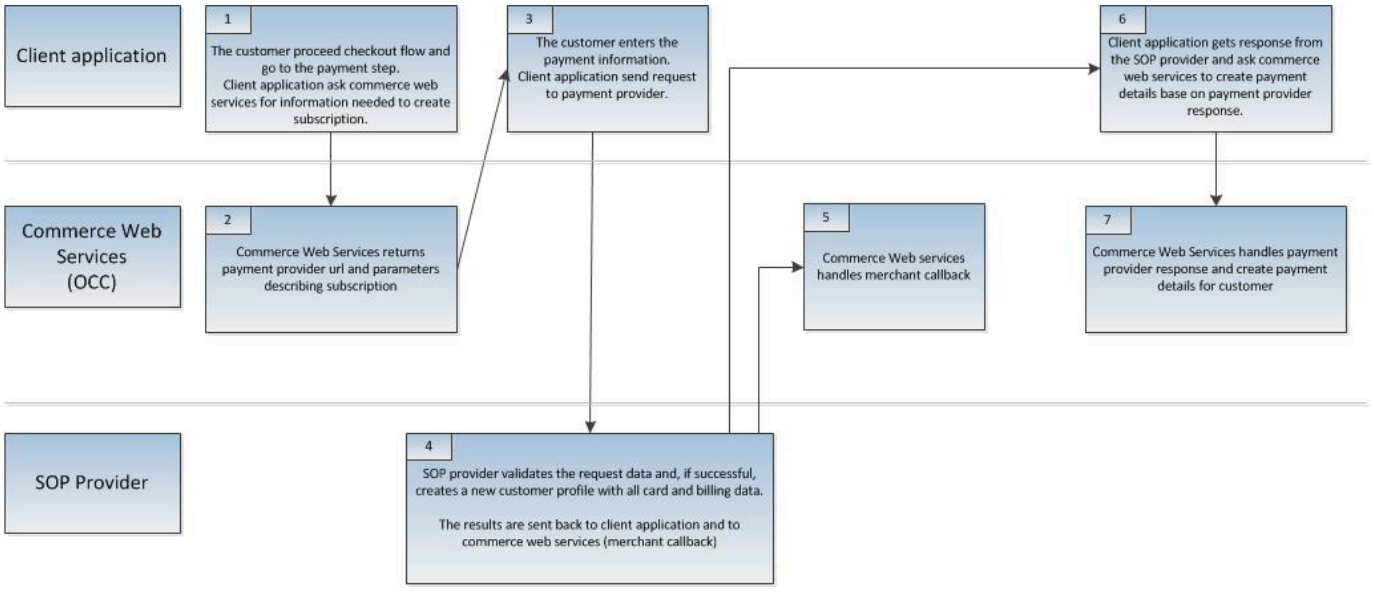
ℹ Note

For details on SOP requests see the available resources described in the acceleratorwebservicesaddon AddOn document.

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.

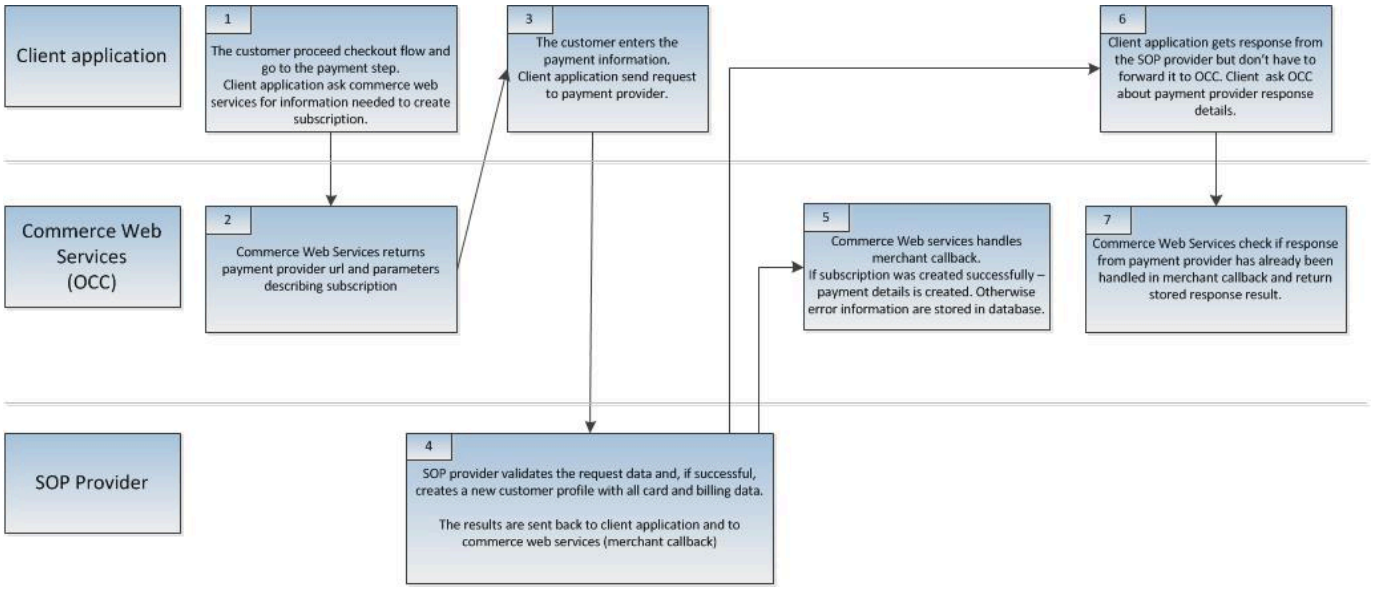


To complete create payment details step in this flow you have to send the requests listed below :

Method	URL	Descritpion
GET	`https://localhost:9002/rest/v2/{baseSiteId}/users/{userId}/carts/{cartId}/payment/sop/request?responseUrl=sampleUrl`	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	`https://localhost:9002/rest/v2/{baseSiteId}/users/{userId}/carts/{cartId}/payment/sop/response`	Handles response from payment provider and creates payment details.
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 ne subscription contac payment provider. **i Note** It is important to s `extendedMerch` because this para url will be returne callback.
POST	postUrl got in previous request	Creates a subscript with the payment p
GET	`https://localhost:9002/rest/v2/{baseSiteId}/users/{userId}/carts/{cartId }/payment/sop/response`	Get information abo response which wa callback.The respor following values: • **200 (OK)** - returned an step of chec • **202 (Accep** received th payment pr client has t • **400 (Bad R** provider rej subscriptio information returned in
POST	`https://localhost:9002/rest/v2/electronics/users/{userId}/orders`	Authorizes card - th in the case of a star

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 `acceleratorservices` extension.

Property Key	Description
`sop.post.url`	This is the URL for the Silent Order Post. It is recommended that it uses a secure connection for data security. Example : `#sop.post.url=https://orderpagetest.ic3.com/hop/ProcessOrder.do sop.post.url=/acceleratorservices/sop-mock/process`

Property Key	Description
`hop.cybersource.sharedSecret`	It is key generated in your cybersource profile which signs the transaction data and is required for each transaction. → **Tip** For site-specific settings in a project that has multiple storefronts, append the name of your site. Example : `hop.cybersource.sharedSecret.wsIntegrationTest=MIGfMA0GCSqGSIb3DQEBAQUAA_Your_shared_`
`hop.cybersource.merchantID`	This is the CyberSource-assigned merchant ID issued when creating your account with them. → **Tip** For site-specific settings in a project that has multiple storefronts, append the name of your site. Example : `hop.cybersource.merchantID.wsIntegrationTest=your_merchant_id`

The table below provides a description of the key properties used by acceleratorwebservicesaddon AddOn.

Property Key	Description

`webroot.commercewebservices.http`
 	
`webroot.commercewebservices.https`
	This is the base URL for commerce web services. It is required to create the full merchant callback URL.
	
	Example:
	
	`webroot.commercewebservices.http=http://localhost:9001/rest`
	`webroot.commercewebservices.https=https://localhost:9002/rest`
	
	→ **Tip**
	For site specific settings in project that has multiple storefronts, add the name of your site after the `webroot.commercewebservices` part.
	
	Example:
	
	`webroot.commercewebservices.wsIntegrationTest.http=http://localhost:9001/rest`
	`webroot.commercewebservices.wsIntegrationTest.https=https://localhost:9002/rest`

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 model co
 * 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.order.Ch
    <property name="orderStatusUpdateChannel" ref="orderStatusUpdateChannel"/>
</bean>
```

Enabling Cron Jobs

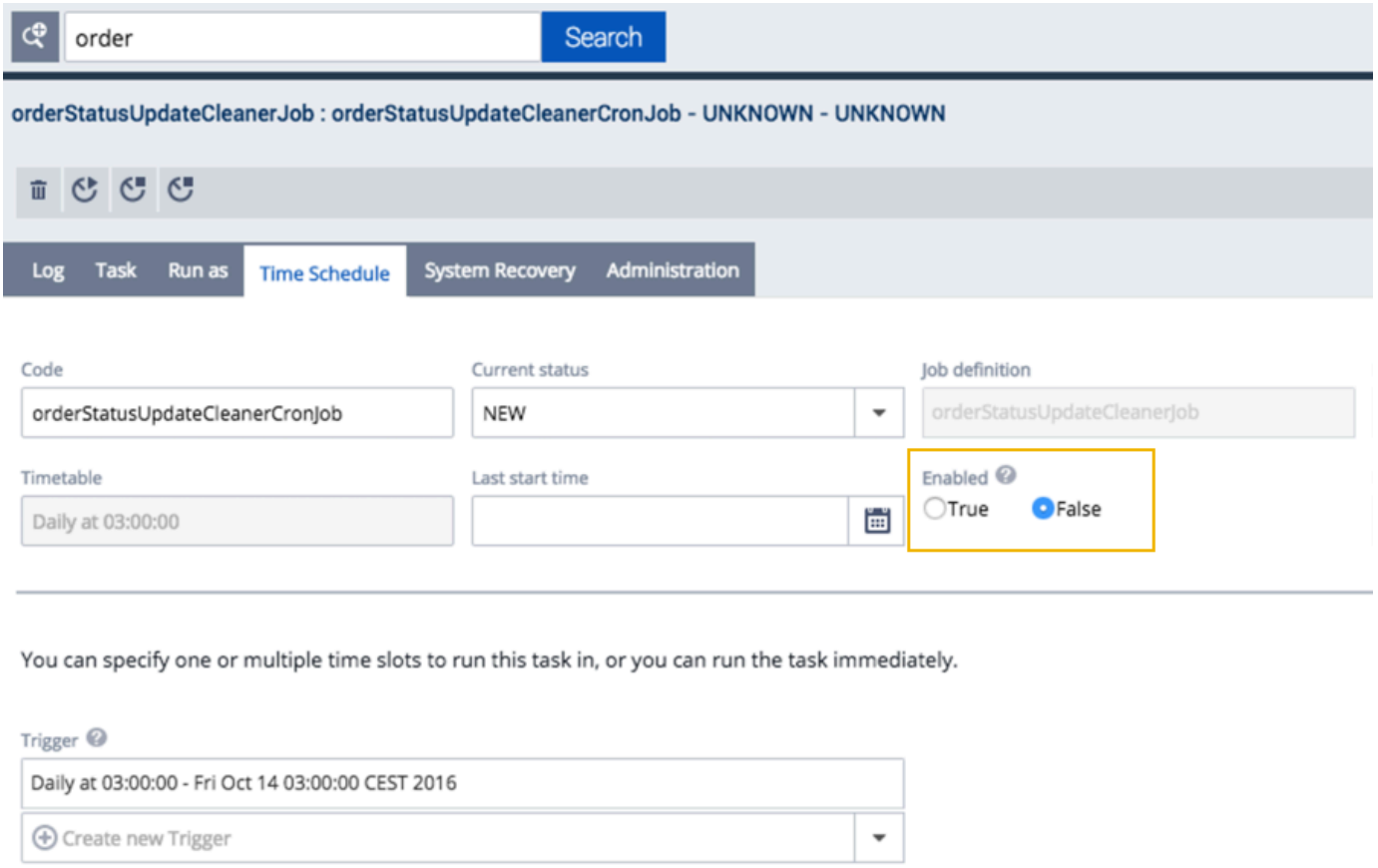
The Order Status Update Queue is an in-memory structure, which has to be cleaned from time to time. The `OrderStatusUpdateCleanerJob` is predefined for you but not active by default.

Note

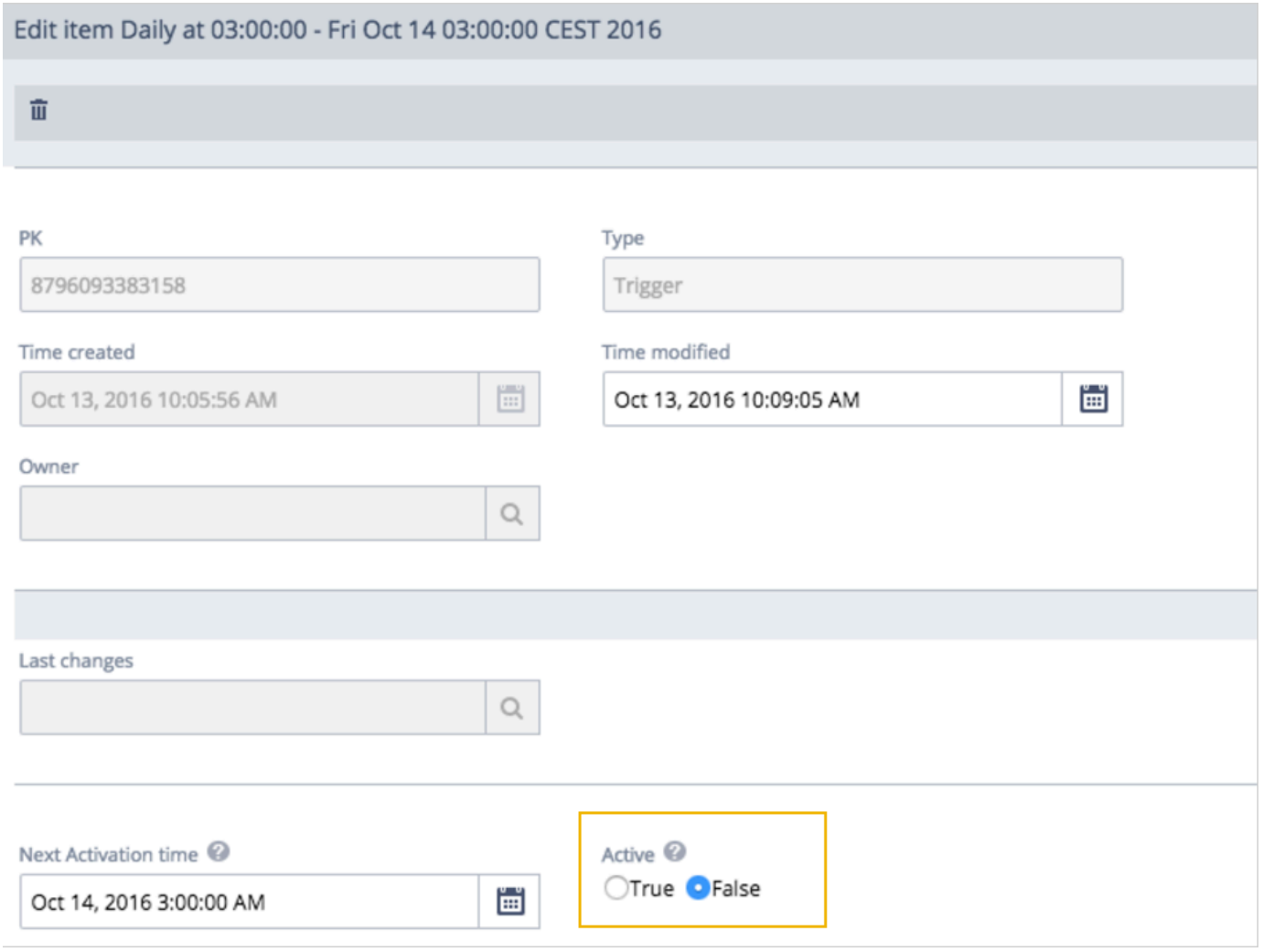
The `orderStatusUpdateCleanerJob` settings are loaded together with the `ycommercewebsevices` extension project data. If the project data is not loaded, you need to define the cronjob based on the `orderStatusUpdateCleanerJob` job definition.

To make it running open theBackoffice Administration Cockpit and navigate to System > Background Processes > CronJobs. Select the `OrderStatusUpdateCleanerJob`from the list and perform the following steps:

- Set the Enabled flag to **True** for the `OrderStatusUpdateCleanerJob`.



- Set the Active flag to **True** for the trigger.



Rest Resource

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

```
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.Sess
 <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>OccConsentLayerFilter</value>
      <value>cxOccPersonalizationFilter</value>
      <value>cartMatchingFilter</value>
      <value>guestRoleFilterV2</value>
     </list>
    </property>
   </bean>
   ```

OCC API v1 Reference

Version one of the OCC API offers stateful interaction with SAP Commerce functionality.

The legacy v1 OCC API continues to be supported, but is not recommended for new development. Instead, use the current and more fully-featured v2 implementation. For more details, see OCC API Implementation.

> ⚠ **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 v1 comparison with OCC v2

The SAP Commerce OCC API offers both the legacy v1 stateful implementation, alongside the newer, default stateless REST API. Get a full overview of the differences.

OCC v1 REST Calls

Get an overview of REST calls available in Version 1 of the SAP Commerce OCC API.

OCC v1 Sample Flows

Sample flow calls for the SAP Commerce OCC API in **v1**

Error Responses from OCC v1

Error responses returned by the `ycommercewebservices` extension.

Cross-Origin Resource Sharing

Cross-Origin Resource Sharing (CORS) is a W3C effort to introduce a standard mechanism for enabling cross-domain requests in web browsers and participating servers.

Upgrading to OCC Extensions

Guidelines on upgrading from v1 to Extensions-based implementation of the OCC API.

OCC v1 comparison with OCC v2

The SAP Commerce OCC API offers both the legacy v1 stateful implementation, alongside the newer, default stateless REST API. Get a full overview of the differences.

Overview

The SAP Commerce OCC API has two available implementations. OCC v2 ensures a configurable and stateless implementation of RESTful web services, supporting future growth in a headless commerce environment. However, the legacy v1 implementation is still supported for existing customers using that version. For documention on the default version two, see OCC API Implementation.

Version 2 (default)

The key features of Version 2 are as follows:

Feature	Description
Stateless	• The calls are now stateless so the customer **data is no longer preserved** between subsequent requests. • Each time the customer needs to provide all the required data such as customer id or cart id.
RESTful implementation	• URL resources have been refactored and are now more RESTful. • The customer needs to provide resource type and resource identifier of the resource he would like to work with.
Data creation	• Executed using the URL parameter list or RequestBody.

Version 1

The key features of Version 1 are as follows:

Feature	Description
Stateful	• A stateful way of communication with commerce layer • Flows some extent similar to storefront flows, based on the `acceleratorservices` extension (for example the checkout process). • The customer cart and other customer data are stored in the session, and are preserved between subsequent requests (basing on the assumption the JSESSIONID is stored). • The customer can be more focused on the actual actions he wants to perform and not on holding and providing the whole context data.
Response format	• Responses are provided in XML or JSON format., depending on the request.

Version Resources Separation

Separate Servlets

For each version, there is a separate servlet defined in the `web.xml` file. These servlets have different java-based configurations. There is also a servlet defined for OAuth authorization - in this case it is a feature common for both versions.

v1

The v1 servlet definition is as follows:

```
...
        <servlet>
        <servlet-name>springmvc-v1</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextClass</param-name>
            <param-value>
                org.springframework.web.context.support.AnnotationConfigWebApplicationContext
            </param-value>
        </init-param>
```

```
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>
                de.hybris.platform.ycommercewebservices.v1.config.WebConfig
            </param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>springmvc-v1</servlet-name>
        <url-pattern>/v1/*</url-pattern>
    </servlet-mapping>
...
```

v2

The v2 servlet definition is as follows:

```
...
    <servlet>
        <servlet-name>springmvc-v2</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextClass</param-name>
            <param-value>
                org.springframework.web.context.support.AnnotationConfigWebApplicationContext
            </param-value>
        </init-param>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>
                de.hybris.platform.ycommercewebservices.v2.config.WebConfig
            </param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>springmvc-v2</servlet-name>
        <url-pattern>/v2/*</url-pattern>
    </servlet-mapping>
...
```

Spring Configuration

Separate Web Context for Servlets

Each version has its own web context defined based on the Java configuration. The Java configuration defines the request mapping handler, message converters, and handler exception resolvers. It also imports standard XML configuration for the proper version.

`WebConfig.java` configuration for v1:

```
/**
 * Spring configuration which replace <mvc:annotation-driven> tag. It allows override default
 * RequestMappingHandlerMapping with our own mapping handler
 *
 */
@Configuration
@ImportResource(
{ "WEB-INF/config/v1/springmvc-v1-servlet.xml" })
public class WebConfig extends WebMvcConfigurationSupport
{
    @Resource
    private List<HttpMessageConverter<?>> messageConvertersV1;
```

```
    @Resource
    private List<HandlerExceptionResolver> exceptionResolversV1;

    private ApplicationContext applicationContext;

    @Override
    @Bean
    public RequestMappingHandlerMapping requestMappingHandlerMapping()
    {
        final CommerceHandlerMapping handlerMapping = new CommerceHandlerMapping("v1");
        handlerMapping.setOrder(0);
        handlerMapping.setDetectHandlerMethodsInAncestorContexts(true);
        handlerMapping.setInterceptors(getInterceptors());
        handlerMapping.setContentNegotiationManager(mvcContentNegotiationManager());
        return handlerMapping;
    }

    @Override
    protected void configureMessageConverters(final List<HttpMessageConverter<?>> converters)
    {
        converters.addAll(messageConvertersV1);
    }

    @Override
    protected void configureHandlerExceptionResolvers(final List<HandlerExceptionResolver> exceptionResolvers)
    {
        final ExceptionHandlerExceptionResolver exceptionHandlerExceptionResolver = new ExceptionHandlerExceptionRes
        exceptionHandlerExceptionResolver.setApplicationContext(applicationContext);
        exceptionHandlerExceptionResolver.setContentNegotiationManager(mvcContentNegotiationManager());
        exceptionHandlerExceptionResolver.setMessageConverters(getMessageConverters());
        exceptionHandlerExceptionResolver.afterPropertiesSet();

        exceptionResolvers.add(exceptionHandlerExceptionResolver);
        exceptionResolvers.addAll(exceptionResolversV1);
        exceptionResolvers.add(new ResponseStatusExceptionResolver());
        exceptionResolvers.add(new DefaultHandlerExceptionResolver());
    }

    @Override
    public void setApplicationContext(final ApplicationContext applicationContext) throws BeansException
    {
        super.setApplicationContext(applicationContext);
        this.applicationContext = applicationContext;
    }
}
```

`WebConfig.java` configuration for v2:

```
/**
 * Spring configuration which replace <mvc:annotation-driven> tag. It allows override default
 * RequestMappingHandlerMapping with our own mapping handler
 *
 */
@Configuration
@ImportResource(
{ "WEB-INF/config/v2/springmvc-v2-servlet.xml" })
public class WebConfig extends WebMvcConfigurationSupport
{
    @Resource
    private List<HttpMessageConverter<?>> messageConvertersV2;

    @Resource
```

```
    private List<HandlerExceptionResolver> exceptionResolversV2;

    private ApplicationContext applicationContext;

    @Override
    @Bean
    public RequestMappingHandlerMapping requestMappingHandlerMapping()
    {
        final CommerceHandlerMapping handlerMapping = new CommerceHandlerMapping("v2");
        handlerMapping.setOrder(0);
        handlerMapping.setDetectHandlerMethodsInAncestorContexts(true);
        handlerMapping.setInterceptors(getInterceptors());
        handlerMapping.setContentNegotiationManager(mvcContentNegotiationManager());
        return handlerMapping;
    }

    @Override
    protected void configureMessageConverters(final List<HttpMessageConverter<?>> converters)
    {
        converters.addAll(messageConvertersV2);
    }

    @Override
    protected void configureHandlerExceptionResolvers(final List<HandlerExceptionResolver> exceptionResolvers)
    {
        final ExceptionHandlerExceptionResolver exceptionHandlerExceptionResolver = new ExceptionHandlerExceptionRes
        exceptionHandlerExceptionResolver.setApplicationContext(applicationContext);
        exceptionHandlerExceptionResolver.setContentNegotiationManager(mvcContentNegotiationManager());
        exceptionHandlerExceptionResolver.setMessageConverters(getMessageConverters());
        exceptionHandlerExceptionResolver.afterPropertiesSet();

        exceptionResolvers.add(exceptionHandlerExceptionResolver);
        exceptionResolvers.addAll(exceptionResolversV2);
        exceptionResolvers.add(new ResponseStatusExceptionResolver());
        exceptionResolvers.add(new DefaultHandlerExceptionResolver());
    }

    @Override
    public void setApplicationContext(final ApplicationContext applicationContext) throws BeansException
    {
        super.setApplicationContext(applicationContext);
        this.applicationContext = applicationContext;
    }
}
```

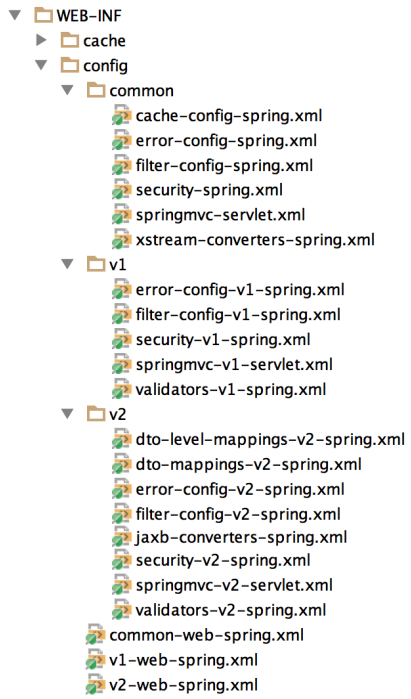
Common Context

In the `web.xml` file there is a common context defined. Beans defined in this context are available for both servlets. This context should contain security configuration and beans used in filters.

```
...
        <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>WEB-INF/ycommercewebservices-web-spring.xml</param-value>
    </context-param>
...
```

Configuration Files Localization

Spring configuration files can be found in the `WEB-INF/config` directory. Configuration files specific for versions are in **v1**, **v2** sub-directories.



Enabling Version 1

The old v1 version is turned off by default. To enable it, uncomment the filters and the servlet definition in the `web.xml` file as shown in the example below.

```
<!-- Uncomment to make v1 version available -->
        <!--
        <filter>
                <description>
                        Spring configured chain of spring filter beans
                </description>
                <filter-name>commerceWebServicesFilterChainV1</filter-name>
                <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
        </filter>
        <filter-mapping>
                <filter-name>commerceWebServicesFilterChainV1</filter-name>
                <servlet-name>springmvc-v1</servlet-name>
        </filter-mapping>
        <filter-mapping>
                <filter-name>httpPutFormFilter</filter-name>
                <servlet-name>springmvc-v1</servlet-name>
        </filter-mapping>
        <servlet>
                <servlet-name>springmvc-v1</servlet-name>
                <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
                <init-param>
                        <param-name>contextClass</param-name>
                        <param-value>
                                org.springframework.web.context.support.AnnotationConfigWebApplicationContext
                        </param-value>
                </init-param>
                <init-param>
                        <param-name>contextConfigLocation</param-name>
                        <param-value>
                                de.hybris.platform.ycommercewebservices.v1.config.WebConfig
                        </param-value>
                </init-param>
                <load-on-startup>1</load-on-startup>
        </servlet>
        <servlet-mapping>
                <servlet-name>springmvc-v1</servlet-name>
                <url-pattern>/v1/*</url-pattern>
        </servlet-mapping>
        -->
        <!-- END Uncomment to make v1 version available -->
```

Additionally, enable v1 spring configuration in the `ycommercewebservices-web-spring.xml` file:

```
<!-- Uncomment to enable version v1 -->
<!--<import resource="config/v1-web-spring.xml"/>-->
```

Related Information

RESTful Implementation
Upgrading to OCC Extensions

OCC v1 REST Calls

Get an overview of REST calls available in Version 1 of the SAP Commerce OCC API.

> ⚠ **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.

Overview

Currently it is possible to retrieve a list of all products fulfilling given criteria and details of a product specified by its code. The child pages of this document hold information about currently available calls grouped by resource. Each document contains tables presenting examples of the URL requests along with available parameters.

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. Parameters are handled separately in a specialized HTTP request filter. Check the following examples:

  - `https://localhost:9001/rest/v1/mysite/cart?curr=USD&lang=en`: Use English language and US Dollar for the request.
  - `https://localhost:9001/rest/v1/mysite/cart/entry?curr=USD`: Use US Dollar and default language of mysite's store.
  - `https://localhost:9002/rest/v1/mysite/customers/current?lang=de`: Use German language and default currency of mysite's store.

Site Parameters

In many URLs of resources there is a site path parameter, for example `https://localhost:9001/rest/v1/electronics/catalogs` where *<electronics>* is the *<BaseSite.uid>* property of the website. In your custom implementation you need to add your own base site:

- URL pattern, for example `(?i)^https?://localhost(:[]+)?/rest/.*$`

  Configure `productUrlPattern`, for example `{category-path}/{product-name}/p/{product-code}`

  You can use following parameters:

  - *<{baseSite-uid}>*
  - *<{category-path}>*
  - *<{product-name}>*
  - *<{product-code}>*
- Configure `categoryUrlPattern`, for example `/{category-path}/c/{category-code}`

  You can use following parameters:

  - *<{baseSite-uid}>*

- *<{category-path}>*
- *<{category-code}>*
- *<{catalog-id}>*
- *<{catalogVersion}>*

Here is an example of the response from:
`https://localhost:9001/rest/v1/electronics/catalogs/electronicsProductCatalog/Online/categories/brands?options=PRODUCTS`

```
<category>
   <id>brands</id>
   <lastModified>2012-08-13T09:11:00+02:00</lastModified>
   <name>Brands</name>
   <url>/Brands/c/brands</url>

<subcategories/>
   <products>
      <product>
      <averageRating>3.5</averageRating>
      <purchasable>true</purchasable>
      <code>478828</code>
      <url>/Open-Catalogue/Cameras/Digital-Cameras/Digital-SLR/10-2-Megapixel-D-SLR-with-Standard-Zoom-Lens/p/47882
      <manufacturer>Sony</manufacturer>
      <name>10.2 Megapixel D-SLR with Standard Zoom Lens</name>
   <baseOptions/>
```

If you do not specify the site path parameter or it does not exist, you will recevie the following exception.

```
de.hybris.platform.ycommercewebservices.exceptions.InvalidResourceException: Base site electronis doesn't exist
```

You can easily configure the site properties using the Backoffice Administration Cockpit under WCMS > Website.

Stateless and Stateful Usage of the API

While most of the API resources can be used in a stateless fashion, there are some resources that require the API client to keep track of cookies for the session management. Generally, we advise the API clients to keep track of all cookies they receive from the server and automatically overwrite existing cookies in case the server sends a new cookie value for an existing cookie name.

Using cookies for session management is especially important for all **Cart Resources**. A POST to `/{site}/cart/entry` would create a new in-memory cart each time you execute the request if you omit the cookies that you receive from the server. If you omit passing the cookies, you can neither keep the cart of the customer, nor add multiple products in separate requests that way.

New cookies are sent from the server by HTTP responses and the Set-Cookie header. Your API client needs to check if a Set-Cookie header is available. If it is available, then extract the cookie information. See an example based on Groovy code.

```
//con is a HttpUrlConnection or a HttpsUrlConnection
def cookie = con.getHeaderField('Set-Cookie') //for example JSESSIONID=C69059FD51C68E610321A818B0019DB2; Path=/rest;
def cookieNoPath = cookie.split(';')[0] //only the cookie value now
```

Once your API issues an HTTP request to call one of the resources, the cookie needs to be a part of the request headers.

```
con.setRequestProperty("Cookie", cookie)
```

Resource Cart

Overview of the available **Cart** resources.

Note

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

Sequence of Calling Cart API Methods

The sequence of calling methods from the Cart resource is essential. The following table presents an example for a simple workflow.

Method	URL	Request Body
POST	https://localhost:9002/rest/v1/electronics/customers	
POST	https://localhost:9002/authorizationserver/oauth/token	
POST	https://localhost:9002/rest/v1/customers/current/addresses	
GET	https://localhost:9002/rest/v1/electronics/cart	
POST	https://localhost:9002/rest/v1/electronics/cart/entry	
POST	https://localhost:9002/rest/v1/electronics/cart/entry	
GET	https://localhost:9002/rest/v1/electronics/cart	
PUT	https://localhost:9002/rest/v1/electronics/cart/entry/0	
DELETE	https://localhost:9002/rest/v1/electronics/cart/entry/0	
GET	https://localhost:9002/rest/v1/electronics/cart	
PUT	https://localhost:9002/rest/v1/electronics/cart/address/delivery/id	
DELETE	https://localhost:9002/rest/v1/electronics/cart/address/delivery	
GET	https://localhost:9002/rest/v1/electronics/cart/deliverymodes	

Method	URL	Request Body
PUT	https://localhost:9002/rest/v1/electronics/cart/address/delivery/id	
GET	https://localhost:9002/rest/v1/electronics/cart/deliverymodes	
PUT	https://localhost:9002/rest/v1/electronics/cart/deliverymodes/premium-gross	
DELETE	https://localhost:9002/rest/v1/electronics/cart/deliverymodes	
PUT	https://localhost:9002/rest/v1/electronics/cart/deliverymodes/standard-gross	
GET	https://localhost:9002/rest/v1/electronics/customers/current/paymentinfos	
PUT	https://localhost:9002/rest/v1/electronics/cart/paymentinfo/$paymentInfoId$	
POST	https://localhost:9002/rest/v1/electronics/cart/authorize	
POST	https://localhost:9002/rest/v1/electronics/cart/placeorder	
GET	https://localhost:9002/rest/v1/orders	
GET	https://localhost:9002/rest/v1/orders/$orderNumber$	
POST	https://localhost:9002/rest/v1/customers/current/logout	
POST	https://localhost:9002/rest/v1/electronics/cart/paymentinfo	Should set header Content-Type: to application/x-www-form-urlencode accountHolderName: "Anja Hertz" cardNumber: "4111111111111111" cardType: "visa" expiryMonth: "01" expiryYear: "2013" saved: "true" defaultPaymentInfo: "true" billingAddress.titleCode: "mr" billingAddress.firstName: "Anja" billingAddress.lastName: "Hertz" billingAddress.line1: "test1" billingAddress.line2: "test2" billingAddress.postalCode: "1234 billingAddress.town: "somecity" billingAddress.country.isocode: '

Resource: /{site}/cart

Method	**GET**
Description	Returns a session cart if such a cart exists. If there was no cart in the current session: • For a logged in user, cart is restored if it exists, and returned • For anonymous user or when there is no saved cart, an empty mock-cart data object is created and returned
Example URL	`https://localhost:9002/rest/v1/electronics/cart`
Authentication	None
Request Parameters	Not applicable
Headers	• **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**
Representations	**application/xml**, **application/json**
Cookies	It requires session information sent in a cookie or via ;jsessionid=... in the URL. Otherwise a new session and therefore new cart is created.

Resource: /{site}/cart/entry

Method	**POST**
Description	Adds a product to the session cart. If session cart does not exist yet, it is created beforehand.
Example URL	`https://localhost:9002/rest/v1/electronics/cart/entry`
Authentication	None
Request Parameters	Parameters need to be provided as POST body. • *<code>* (Style: code, Required: true): Code of the product to be added to cart. Product look-up is performed for the current session product catalog version. • *<qty>* (Style: qty, Required: false, Default: 1): New cart item's quantity.
Headers	• **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**
Representations	**application/xml**, **application/json**
Cookies	It requires session information sent in a cookie or via ;jsessionid=... in the URL. Otherwise new session and therefore new cart is created. For the cart to be accessible in next requests, add **JSESSIONID** cookie to the next calls.

Resource: /{site}/cart/entry/{entryNumber}

Method	**PUT**
Description	Updates the cart entry quantity. If there is no cart entry with the requested number, the `CommerceCartModificationException` is returned in the response.
Example URL	`https://localhost:9002/rest/v1/electronics/cart/entry/0?qty=5`
Authentication	None
Request Parameters	Parameters need to be provided in the PUT body. HttpPutFormContentFilter is used to read the PUT body. • *<qty>* (Style: qty, Required: true): New quantity for cart entry.
Headers	• **Content-Type** application/x-www-form-urlencoded (Style: header, Required: true): The **application/x-www-form-urlencoded** content type is required by **HttpPutFormContentFilter** • **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**
Representations	**application/xml,** application/json
Cookies	It requires session information sent in a cookie or via ;jsessionid=... in the URL. Otherwise new session and therefore new cart is created. For the cart to be accessible in next requests, add **JSESSIONID** cookie to the next calls.

Resource: /{site}/cart/entry/{entryNumber}

Method	**DELETE**
Description	Deletes cart entry. If there is no cart entry with the requested number, the **CommerceCartModificationException** is returned in the response.
Example URL	`https://localhost:9002/rest/v1/electronics/cart/entry/0`
Authentication	None
Request Parameters	Not applicable.
Headers	• **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**
Representations	**application/xml**, **application/json**
Cookies	It requires session information sent in a cookie or via ;jsessionid=... in the URL. Otherwise a new session and therefore a new cart is created. For the cart to be accessible in next requests, add **JSESSIONID** cookie to the next calls.

Resource: /{site}/cart/address/delivery/{id}

Method	**PUT**
Description	Sets a delivery address for a session cart. The address ID must be a PK of a current address. The address country must be among the delivery countries of a current base store. Method responses with cart data that should reflect the change in the cart.
Checkout State Prerequisites	There must be a session cart created.
Example URL	https://localhost:9002/rest/v1/electronics/cart/address/delivery/123
Authentication	Access is restricted to the **ROLE_CUSTOMERGROUP**. This method is restricted to HTTPS channel.
Request Parameters	Not applicable.
Headers	• **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**
Representations	**application/xml , application/json**
Cookies	It requires session information sent in a cookie or via ;jsessionid=... in the URL. Otherwise a new session and subsequently a new cart is created. For the cart to be accessible in next requests, add **JSESSIONID** cookie to the next calls.

Resource: /{site}/cart/address/delivery/

Method	**DELETE**
Description	Deletes delivery address from a session cart. The response includes cart data that should reflect the change in the cart.
Example URL	`https://localhost:9002/rest/v1/electronics/cart/address/delivery`
Authentication	Access is restricted to the **ROLE_CUSTOMERGROUP**. This method is restricted to HTTPS channel.
Request Parameters	Not applicable.
Headers	• **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**
Representations	**application/xml , application/json**
Cookies	It requires session information sent in a cookie or via ;jsessionid=... in the URL. Otherwise a new session and subsequently a new cart is created. For the cart to be accessible in next requests, add **JSESSIONID** cookie to the next calls.

Resource: /{site}/cart/deliverymodes/

Method	**GET**
Description	Returns all delivery modes supported for the current base store and cart delivery address.

Checkout State Prerequisites	There must be a delivery address for the cart, otherwise an empty list will be returned.
Example URL	`https://localhost:9002/rest/v1/electronics/cart/deliverymodes`
Authentication	Access is restricted to the **ROLE_CUSTOMERGROUP**. This method is restricted to HTTPS channel.
Request Parameters	Not applicable.
Headers	• **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**
Representations	**application/xml , application/json**
Cookies	It requires session information sent in a cookie or via ;jsessionid=... in the URL. Otherwise a new session and subsequently a new cart is created. For the cart to be accessible in next requests, add **JSESSIONID** cookie to the next calls

Resource: /{site}/cart/deliverymodes/{code}

Method	**PUT**
Description	Sets delivery mode according to the code for the current cart. Method responses with the cart data that should reflect the change in the cart.
Example URL	`https://localhost:9002/rest/v1/electronics/cart/deliverymodes/standard-gross`
Authentication	Access is restricted to the **ROLE_CUSTOMERGROUP**. This method requires basic HTTP authentication and is restricted to HTTPS channel.
Request Parameters	Not applicable.
Headers	• **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**.
Representations	**application/xml , application/json**
Cookies	It requires session information sent in a cookie or via ;jsessionid=... in the URL. Otherwise a new session and subsequently a new cart is created. For the cart to be accessible in the next requests, add **JSESSIONID** cookie to the next calls.

Resource: /{site}/cart/deliverymodes/

Method	**DELETE**
Description	Removes delivery mode from the current cart. The response includes cart data that should reflect the change in the cart.
Example URL	`https://localhost:9002/rest/v1/electronics/cart/deliverymodes`
Authentication	Access is restricted to the **ROLE_CUSTOMERGROUP**. This method requires basic HTTP authentication and is restricted to HTTPS channel.
Request Parameters	Not applicable.
Headers	• **Accept**

	(Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json.**
Representations	**application/xml , application/json**
Cookies	It requires session information sent in a cookie or via ;jsessionid=... in the URL. Otherwise a new session and subsequently a new cart is created. For the cart to be accessible in next requests, add **JSESSIONID** cookie to the next calls.

Resource: /{site}/cart/paymentinfo/

Method	**POST**
Description	Creates a new credit card payment info for the current user. The response includes the payment info data.
Checkout State Prerequisites	There must be a session cart.
Example URL	`https://localhost:9002/rest/v1/electronics/cart/paymentinfo`
Authentication	Access is restricted to the **ROLE_CUSTOMERGROUP**. This method requires basic HTTP authentication and is restricted to HTTPS channel.
Request Body	Need to be provided as URL encoded post body: • *<accountHolderName>* (Required: true): • *<cardNumber>* (Required: true): • *<cardType>* (Required: true): Call `GET /{site}/cardtypes` beforehand to see what card types are supported • *<expiryMonth>* (Required: true): • *<expiryYear>* (Required: true): • *<saved>* (Required: false): If the payment info should be saved for the customer and than could be reused for future orders. • *<defaultPaymentInfo>* (Required: false, Default: false): • *<billingAddress.titleCode>* (Required: true): • *<billingAddress.firstName>* (Required: true): • *<billingAddress.lastName>* (Required: true): • *<billingAddress.line1>* (Required: true):

	• *<billingAddress.line2>* (Required: false): • *<billingAddress.postalCode>* (Required: true): • *<billingAddress.town>* (Required: true): • *<billingAddress.country.isocode>* (Required: true):
Headers	• **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json.**
Representations	**application/xml , application/json**
Cookies	It requires session information sent in a cookie or via ;jsessionid=... in the URL. Otherwise a new session and subsequently a new cart is created. For the cart to be accessible in next requests, add **JSESSIONID** cookie to the next calls.

Resource: /{site}/cart/paymentinfo/{id}

Method	**PUT**
Description	Adds a credit card payment info according to the ID of the current user's cart.
Checkout State Prerequisites	There must be a session cart. The response includes cart data that should reflect the change in the cart.
Example URL	`https://localhost:9002/rest/v1/electronics/cart/paymentinfo/1234`
Authentication	Access is restricted to the **ROLE_CUSTOMERGROUP**. This method requires basic HTTP authentication and is restricted to HTTPS channel.
Headers	• **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**.
Representations	**application/xml , application/json**
Cookies	It requires session information sent in a cookie or via ;jsessionid=... in the URL. Otherwise a new session and subsequently a new cart is created. For the cart to be accessible in next requests, add **JSESSIONID** cookie to the next calls.

Resource: /{site}/cart/authorize

Method	**POST**
Description	Authorizes the credit card payment with the CCV security code. The response contains the cart data.
Checkout State Prerequisites	There must be a session cart with associated credit card payment. The response contains cart data, that should reflect the change in the cart.
Example URL	`https://localhost:9002/rest/v1/electronics/cart/authorize`

Authentication	Access is restricted to the **ROLE_CUSTOMERGROUP**. This method requires basic HTTP authentication and is restricted to HTTPS channel.
Request Parameters	Need to be provided as URL encoded post body: • *<securityCode>* (Required: true):
Headers	• **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**
Representations	**application/xml , application/json**
Cookies	It requires session information sent in a cookie or via ;jsessionid=... in the URL. Otherwise a new session and subsequently a new cart is created. For the cart to be accessible in next requests, add **JSESSIONID** cookie to the next calls.

Resource: /{site}/cart/placeorder

Method	**POST**
Description	Places an order based on the session cart. The response contains the new order data.
Checkout State Prerequisites	User session must have a valid cart.
Example URL	`https://localhost:9002/rest/v1/electronics/cart/placeorder`
Authentication	Access is restricted to the **ROLE_CUSTOMERGROUP**. This method requires basic HTTP authentication and is restricted to HTTPS channel.
Headers	• **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**.
Representations	**application/xml , application/json**
Cookies	It requires session information sent in a cookie or via ;jsessionid=... in the URL. Use **JSESSIONID** cookie to the next calls.

Resource: /{site}/cart/restore

Method	**GET**
Description	Restores anonymous cart by **guid**.
Example URL	`https://localhost:9002/rest/v1/electronics/cart/restore?guid=xxxxx`
Authentication	Access is restricted to the **ROLE_CLIENT** and **ROLE_TRUSTED_CLIENT**. This method is restricted to HTTPS channel.
Headers	• **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**.

Representations	**application/xml , application/json**

Resource: /{site}/cart/promotion/{promotionCode}

Method	**POST**
Description	Enables order promotion given by **promotionCode** for a current cart. If promotion is applied successfully, then the modified cart will be returned.
Example URL	`https://localhost:9002/rest/v1/electronics/cart/OrderThreshold15Discount`
Authentication	Access is restricted to the **ROLE_CLIENT**, **ROLE_CUSTOMERGROUP**, and **ROLE_TRUSTED_CLIENT**. This method is restricted to HTTPS channel.
Headers	• **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**.
Representations	**application/xml , application/json**
Cookies	It requires session information sent in a cookie or via ;jsessionid=... in the URL. Use **JSESSIONID** cookie to the next calls.

Resource: /{site}/cart/promotion/{promotionCode}

Method	**DELETE**
Description	Disables order promotion given by **promotionCode** for a current cart. If promotion is removed successfully, then the modified cart will be returned.
Example URL	`https://localhost:9002/rest/v1/electronics/cart/promotion/OrderThreshold15Discount`
Authentication	Access is restricted to the **ROLE_CLIENT, ROLE_CUSTOMERGROUP** and **TRUSTED_CLIENT**. This method requires client or customer HTTP authentication and is restricted to HTTPS channel.
Headers	• **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**.
Representations	**application/xml , application/json**
Cookies	It requires session information sent in a cookie or via ;jsessionid=... in the URL. Use **JSESSIONID** cookie to the next calls.

Resource: /{site}/cart/voucher/{voucherCode}

Method	**POST**
Description	Applies a voucher given by a **voucherCode** for the current cart. If the voucher code is applied successfully, then the modified cart will be returned. The JSON response should contain the **appliedVouchers** attribute with the voucher information. The XML response should contain the **appliedVouchers** tag with the voucher information.
Example URL	`https://localhost:9002/rest/v1/electronics/cart/voucher/xyz-MHE2-B8L5-LPHE`

Authentication	Access is restricted to the **ROLE_CLIENT**, **ROLE_CUSTOMERGROUP** and **ROLE_TRUSTED_CLIENT**. This method is restricted to HTTPS channel.
Request Parameters	Not applicable.
Headers	• **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**.
Representations	**application/xml , application/json**
Cookies	It requires session information sent in a cookie or via ;jsessionid=... in the URL. Use **JSESSIONID** cookie to the next calls.

Resource: /{site}/cart/voucher/{voucherCode}

Method	**DELETE**

Description	Method removes a voucher given by the **voucherCode** from the current cart. If the voucher code is removed successfully, then the modified cart will be returned.
	• In JSON response, the **appliedVouchers** attribute should not contain released voucher anymore.
	`"appliedVouchers": []`
	• In XML response, the **appliedVouchers** tag should not contain released voucher anymore.
	`<appliedVouchers />`

Example URL	`https://localhost:9002/rest/v1/electronics/cart/voucher/xyz-MHE2-B8L5-LPHE`
Authentication	Access is restricted to the **ROLE_CLIENT**, **ROLE_CUSTOMERGROUP** and **TRUSTED_CLIENT**. This method is restricted to HTTPS channel.
Request Parameters	Not applicable.
Headers	• **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**.
Representations	**application/xml , application/json**
Cookies	It requires session information sent in a cookie or via *<;jsessionid=...>* in the URL. Use **JSESSIONID** cookie to the next calls.

Resource Catalogs

an overview on the available Catalogs resource

ℹ Note

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

Resource: /{site}/catalogs

Method	**GET**
Description	Returns all catalogs with versions that are defined for the base store.
Example URL	`https://localhost:9002/rest/v1/electronics/catalogs`
Request Parameters	• *<options>* (Style: query, Required: false): It defines the level of details requested. This parameter can have a combination of the following values, separated by ,:**BASIC, CATEGORIES, PRODUCTS, SUBCATEGORIES**. Keep in mind that the value needs to be properly URL-encoded. Options are dependent on each other: for example **PRODUCTS** option makes sense only if **CATEGORIES** option is also used, as the products elements are embedded in categories in the response object. Example: *<options>*=**CATEGORIES,PRODUCTS** requests root categories with products.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/catalogs/{id}

Method	**GET**
Description	Returns catalog by ID with versions defined for the current base store.
Example URL	`https://localhost:9002/rest/v1/electronics/catalogs/electronicsProductCatalog`
Request Parameters	• *<options>* (Style: query, Required: false): It defines the level of details requested. This parameter can have a combination of the following values, separated by ,:**BASIC,CATEGORIES,PRODUCTS,SUBCATEGORIES**. Keep in mind that the value needs to be properly URL-encoded. Options are dependent on each other: **PRODUCTS** option makes sense only if **CATEGORIES** option is also used, as products elements are embedded in categories in the response object. Example: *<options>*=CATEGORIES, PRODUCTS requests root categories with products.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/catalogs/{catalogId}/{catalogVersionId}

Method	**GET**
Description	Returns information about catalog version that exists for the current base store.

Example URL	`https://localhost:9002/rest/v1/electronics/catalogs/electronicsProductCatalog/Online`
Request Parameters	• *<options>* (Style: query, Required: false): It defines the level of details requested. This parameter can have a combination of the following values, separated by,:**BASIC,CATEGORIES,PRODUCTS,SUBCATEGORIES.** Keep in mind that the value needs to be properly URL-encoded. Options are dependent on each other: for example **PRODUCTS** option makes sense only if **CATEGORIES** option is also used, as products elements are embedded in categories in the response object. Example: *<options>*=**CATEGORIES,PRODUCTS** requests root categories with products.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/catalogs/{catalogId}/{catalogVersionId}/categories/{categoryCode}

Method	**GET**
Description	Returns information about category that exist in a base store's catalog version.
Example URL	`https://localhost:9002/rest/v1/electronics/catalogs/electronicsProductCatalog/Online/categories/1`
Request Parameters	• *<options>* (Style: query, Required: false): It defines the level of details requested. This parameter can have a combination of the following values, separated by,:**BASIC,PRODUCTS,SUBCATEGORIES**. Keep in mind that the value needs to be properly URL-encoded. Options are dependent on each other: **PRODUCTS** option makes sense only if **CATEGORIES** option is also used, as products elements are embedded in categories in the response object. Example: *<options>*=**CATEGORIES,PRODUCTS** requests root categories with products.
Headers	• Accept(Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource Customergroup

An overview of the available **customergroup** resources.

Note

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

Resource: /{site}/customergroups

Method	**POST**
Description	Creates new customer group - direct subgroup of a **customergroup**. The method requires **ROLE_CUSTOMERMANAGERGROUP** and secured HTTPS channel.
Example URL	`https://localhost:9002/rest/v1/electronics/customergroups`
Request Parameters	• *<uid>* (Style: body, Required: true): Uid for new customer group. • *<localizedName >*(Style: body, Required: false): Name in current locale.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/customergroups/{uid}/members

Method	**PUT**
Description	Assign users to the customer group with given *<uid>*. Requires **CUSTOMERMANAGERGROUP** role and HTTPS secured channel.
Example URL	`https://localhost:9002/rest/v1/electronics/customergroups/customergroup/members`
Request Parameters	• *<member>* (Style: body, Required: true): List of users' uid's to assign to customer group. List should be in form: *<member>=<uid1>&<member>=<uid2>...*
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/customergroups

Method	**GET**
Description	Returns all customer groups that are direct subgroups of **customergroup**. Requires **CUSTOMERMANAGERGROUP** role and HTTPS secure channel.
Example URL	`https://localhost:9002/rest/v1/electronics/customergroups?currentPage=0&pageSize=20`
Request Parameters	• *<currentPage>* (Style: url parameter, Required: false, Default: 0): current page number(starts with 0) • *<pageSize>* (Style: url parameter, Required: false, Default: 2147483647): number of customer group returned in one page
Headers	• Accept (Style: header, Required: true):

	Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/customergroups/{uid}

Method	**GET**
Validity	5.0
Description	Returns customer group with specified **uid**. Requires **CUSTOMERMANAGERGROUP** role. The requested group must be a sub group of **customergroup**, otherwise **InvalidCustomerGroupException** is thrown. Requires secure channel.
Example URL	`https://localhost:9002/rest/v1/electronics/customergroups/customergroup/\{myGroup\}?options=MEMBERS,SUBGROUPS`
Request Parameters	• *<options>* (Style: url parameter, Required: false): BASIC - default value. Response contains only *<uid>*, *<localized name>* and number of members, MEMBERS - with this option, response will contain information about member users (customers), SUBGROUP - with this option the response will contain information about this group subgroups. You can combine options.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/customergroups/{uid}/members/{userId}

Method	**DELETE**
Description	Remove user with given *<userId>* from customer group with given *<uid>*. Requires **CUSTOMERMANAGERGROUP** role.
Example URL	`https://localhost:9002/rest/v1/electronics/customergroups/customergroup/members/dab61415-7832-4daf-ae1d-c48988e5bfef`
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application**/xml, **application/json**

Resource Customers

An overview on the available **Customers** resources.

ℹ Note

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

Resource: /customers/current/logout

Method	**POST**
Description	Web service to logs out the customer. Terminates session. Incorporates spring security logout filter.
Example URL	`https://localhost:9002/rest/v1/customers/current/logout`
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.

Resource: /{site}/customers

Method	**POST**
Description	Register a customer.
Example URL	• Access is restricted to ROLE_CLIENT so that only registered API clients can gain it. The client access token must be sent with the request. To obtain the token for a given *<client_id>* an authorization call must be sent first: ◦ `curl --location 'https://localhost:9002/authorizationserver/oauth' \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'client_secret={clientSecret\}' \ --data-urlencode 'grant_type=client_credentials' \ --data-urlencode 'client_id={client_id}'` • Now you can send a registration call: ◦ `https://localhost:9002/rest/v1/electronics/customers`
Request Parameters	• *<login>* (Style: body, Required: true): Customer's login. Customer login is case insensitive. • *<password>* (Style: body, Required: true): Customer's password. • *<firstName>* (Style: body, Required: true): Customer's first name. • *<lastName>* (Style: body, Required: true): Customer's last name. • *<titleCode>* (Style: body, Required: false): Customer's title code. For a list of codes, see StoreController • *<access_token>* (Style: body, Required: false): Access token granted for the given client.
Headers	• Accept(Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/customers/current/addresses/default/{id}

Method	**PUT**
Description	Set address as customer's default address.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/current/addresses/default/12345`
Request Parameters	• *<id>* (Style: body, Required: true): Address id.
Authentication	• Access is restricted to ROLE_CUSTOMERGROUP so the customer access token must be sent with the request. To obtain the token for a given customer an authorization call must be sent first: ◦ `curl --location 'https://localhost:9002/authorizationserver/oauth' \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'username={uid}' \ --data-urlencode 'password={pwd}' \ --data-urlencode 'client_secret={clientSecret\}' \ --data-urlencode 'grant_type=password' \ --data-urlencode 'client_id={client_id}'` • HTTPS channel is required
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml, application/json**

Resource: /{site}/customers/current/profile

Method	**POST**
Description	Update customer's profile.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/current/profile`
Request Parameters	Need to be provided as URL encoded post body. Only non-null values will be updated. • *<titleCode>* (Required: false) • *<firstName>* (Required: false) • *<lastName>* (Required: false) • *<language>* (Required: false) • *<currency>* (Required: false)
Authentication	• Access is restricted to ROLE_CUSTOMERGROUP so the customer access token must be sent with the request. To obtain the token for a given customer an authorization call must be sent first: ◦ `curl --location 'https://localhost:9002/authorizationserver/oauth' \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'username={uid}' \ --data-urlencode 'password={pwd}' \ --data-urlencode 'client_secret={clientSecret\}' \ --data-urlencode 'grant_type=password' \ --data-urlencode 'client_id={client_id}'` • HTTPS channel is required
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.

Representations	**application/xml**, **application/json**

Resource: /{site}/customers/current/addresses

Method	**GET**
Description	Get customer's addresses.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/current/addresses`
Authentication	• Access is restricted to ROLE_CUSTOMERGROUP so the customer access token must be sent with the request. To obtain the token for a given customer an authorization call must be sent first: ◦ `curl --location 'https://localhost:9002/authorizationserver/oauth' \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'username={uid}' \ --data-urlencode 'password={pwd}' \ --data-urlencode 'client_secret={clientSecret\}' \ --data-urlencode 'grant_type=password' \ --data-urlencode 'client_id={client_id}'` • HTTPS channel is required
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/customers/current/addresses

Method	**POST**
Description	Create new address.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/current/addresses`
Request Parameters	See `HttpRequestAddressDataPopulator` class. Need to be provided as URL encoded post body. • *<titleCode>* (Required: true) • *<firstName>* (Required: true) • *<lastName>* (Required: true) • *<line1>* (Required: true) • *<line2>* (Required: false) • *<town>* (Required: true) • *<postalCode>* (Required: true) • *<country.isocode>* (Required: true) • *<region.isocode>* (Required: false)
Authentication	• Access is restricted to ROLE_CUSTOMERGROUP so the customer access token must be sent with the request. To obtain the token for a given customer an authorization call must be sent first: ◦ `curl --location 'https://localhost:9002/authorizationserver/oauth' \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'username={uid}' \ --data-urlencode 'password={pwd}' \ --data-urlencode 'client_secret={clientSecret\}' \`

	`--data-urlencode 'grant_type=password' \`
	`--data-urlencode 'client_id={client_id}'`
	• HTTPS channel is required

Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/customers/current/addresses/{id}

Method	**PUT**
Description	Update address.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/current/addresses/12345`
Request Parameters	See `HttpRequestAddressDataPopulator` class. Need to be provided as URL encoded post body. Only non-null values will be updated. • *<titleCode>* (Required: false) • *<firstName>* (Required: false) • *<lastName>* (Required: false) • *<line1>* (Required: false) • *<line2>* (Required: false) • *<town>* (Required: false) • *<postalCode>* (Required: false) • *<country.isocode>* (Required: false) • *<region.isocode>* (Required: false)

Authentication	• Access is restricted to ROLE_CUSTOMERGROUP so the customer access token must be sent with the request. To obtain the token for a given customer an authorization call must be sent first:
	◦ `curl --location 'https://localhost:9002/authorizationserver/oauth' \`
	`--header 'Content-Type: application/x-www-form-urlencoded' \`
	`--data-urlencode 'username={uid}' \`
	`--data-urlencode 'password={pwd}' \`
	`--data-urlencode 'client_secret={clientSecret\}' \`
	`--data-urlencode 'grant_type=password' \`
	`--data-urlencode 'client_id={client_id}'`
	• HTTPS channel is required

Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json. • Content-Type application/x-www-form-urlencoded (Style: header, Required: true) application/x-www-form-urlencoded content type is required by `HttpPutFormContentFilter`
Representations	**application/xml**,application/json

Resource: /{site}/customers/current/addresses/{id}

Method	**DELETE**

Description	Delete address.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/current/addresses/12345`
Request Parameters	• *<id>*(Style: body, Required: true): Address id.
Authentication	• Access is restricted to ROLE_CUSTOMERGROUP so the customer access token must be sent with the request. To obtain the token for a given customer an authorization call must be sent first: ◦ `curl --location 'https://localhost:9002/authorizationserver/oauth' \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'username={uid}' \ --data-urlencode 'password={pwd}' \ --data-urlencode 'client_secret={clientSecret\}' \ --data-urlencode 'grant_type=password' \ --data-urlencode 'client_id={client_id}'` • HTTPS channel is required
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/customers/current

Method	**GET**
Description	Returns customer profile.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/current`
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Authentication	• Access is restricted to ROLE_CUSTOMERGROUP so the customer access token must be sent with the request. To obtain the token for a given customer an authorization call must be sent first: ◦ `curl --location 'https://localhost:9002/authorizationserver/oauth' \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'username={uid}' \ --data-urlencode 'password={pwd}' \ --data-urlencode 'client_secret={clientSecret\}' \ --data-urlencode 'grant_type=password' \ --data-urlencode 'client_id={client_id}'` • HTTPS channel is required
Representations	**application/xml**, **application/json**

Resource: /{site}/customers/current/password

Method	**POST**
Description	Change customer's password.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/current/password`

Request Parameters	• *<old>* (Style: body, Required: true): Old password. • *<new>* (Style: body, Required: true): New password.
Authentication	• Access is restricted to ROLE_CUSTOMERGROUP so the customer access token must be sent with the request. To obtain the token for a given customer an authorization call must be sent first: ◦ `curl --location 'https://localhost:9002/authorizationserver/oauth' \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'username={uid}' \ --data-urlencode 'password={pwd}' \ --data-urlencode 'client_secret={clientSecret\}' \ --data-urlencode 'grant_type=password' \ --data-urlencode 'client_id={client_id}'` • HTTPS channel is required
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**,application/json

Resource: /{site}/customers/current/forgottenpassword

Method	**POST**
Description	Restore customer's forgotten password.
Example URL	• Access is restricted to **ROLE_CLIENT** so that only registered API clients can gain it. The client access token must be sent with the request. To obtain the token for a given *<client_id>* an authorization call must be sent first: ◦ `curl --location 'https://localhost:9002/authorizationserver/oauth' \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'client_secret={clientSecret\}' \ --data-urlencode 'grant_type=client_credentials' \ --data-urlencode 'client_id={client_id}'` • Now you can send a call: ◦ `https://localhost:9002/rest/v1/electronics/customers/current/forgottenpassword`
Request Parameters	• *<login>* (Style: body, Required: true): Customer's login. Customer login is case insensitive.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/customers/current/paymentinfos

Method	**GET**
Description	Return current customer's credit card payment infos that were previously saved during checkout.

Example URL	`https://localhost:9002/rest/v1/electronics/customers/current/paymentinfos`
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Authentication	• Access is restricted to ROLE_CUSTOMERGROUP so the customer access token must be sent with the request. To obtain the token for a given customer an authorization call must be sent first: ◦ `curl --location 'https://localhost:9002/authorizationserver/oauth/token \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'username={uid}' \ --data-urlencode 'password={pwd}' \ --data-urlencode 'client_secret={clientSecret\}' \ --data-urlencode 'grant_type=password' \ --data-urlencode 'client_id={client_id}'` • HTTPS channel is required
Representations	**application/xml**, **application/json**

Resource: /{site}/customers/current/paymentinfos/{id}

Method	**GET**
Description	Return customer's credit card payment info by payment info id.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/current/paymentinfos/1234`
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Authentication	• Access is restricted to ROLE_CUSTOMERGROUP so the customer access token must be sent with the request. To obtain the token for a given customer an authorization call must be sent first: ◦ `curl --location 'https://localhost:9002/authorizationserver/oauth/token \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'username={uid}' \ --data-urlencode 'password={pwd}' \ --data-urlencode 'client_secret={clientSecret\}' \ --data-urlencode 'grant_type=password' \ --data-urlencode 'client_id={client_id}'` • HTTPS channel is required
Representations	**application/xml**, **application/json**

Resource: /{site}/customers/current/paymentinfos/{id}

Method	**DELETE**
Description	Removes customer's credit card payment info by payment info ID.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/current/paymentinfos/1234`
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Authentication	• Access is restricted to ROLE_CUSTOMERGROUP so the customer access token must be sent with the request. To obtain the token for a given customer an authorization call must be sent first:

	○ `curl --location 'https://localhost:9002/authorizationserver/oauth/token \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'username={uid}' \ --data-urlencode 'password={pwd}' \ --data-urlencode 'client_secret={clientSecret\}' \ --data-urlencode 'grant_type=password' \ --data-urlencode 'client_id={client_id}'` • HTTPS channel is required
Representations	**application/xml**, **application/json**

Resource: /{site}/customers/current/paymentinfos/{id}

Method	**PUT**
Description	Updates existing customer's credit card payment info by payment info ID.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/current/paymentinfos/1234`
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Request Parameters	Need to be provided as URL encoded post body: • *<accountHolderName>* (Required: true): • *<cardNumber>* (Required: true): • *<cardType>* (Required: true): Call GET /{site}/cardtypes beforehand to see what card types are supported. • *<expiryMonth>* (Required: true): • *<expiryYear>* (Required: true): • *<saved>* (Required: false): Tells whether the payment info is saved for the customer and than could be reused for future orders. • *<defaultPaymentInfo>* (Required: false):
Authentication	• Access is restricted to ROLE_CUSTOMERGROUP so the customer access token must be sent with the request. To obtain the token for a given customer an authorization call must be sent first: ○ `curl --location 'https://localhost:9002/authorizationserver/oauth/token \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'username={uid}' \ --data-urlencode 'password={pwd}' \ --data-urlencode 'client_secret={clientSecret\}' \ --data-urlencode 'grant_type=password' \ --data-urlencode 'client_id={client_id}'` • HTTPS channel is required
Representations	**application/xml**, **application/json**

Resource: /{site}/customers/current/paymentinfos/{id}/address

Method	**POST**
Description	Updates billing address of existing customer's credit card payment info by payment info ID.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/current/paymentinfos/1234/address`

Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Request Parameters	Need to be provided as URL encoded post body: • *<titleCode>* (Required: true): • *<firstName>* (Required: true): • *<lastName>* (Required: true): • *<line1>* (Required: true): • *<line2>* (Required: true): • *<postalCode>* (Required: true): • *<town>* (Required: true): • *<country.isocode>* (Required: true): If the payment info should be saved for the customer and than could be reused for future orders. • *<defaultPaymentInfo>* (Required: false):
Authentication	• Access is restricted to ROLE_CUSTOMERGROUP so the customer access token must be sent with the request. To obtain the token for a given customer an authorization call must be sent first: ◦ `curl --location 'https://localhost:9002/authorizationserver/oauth/token \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'username={uid}' \ --data-urlencode 'password={pwd}' \ --data-urlencode 'client_secret={clientSecret\}' \ --data-urlencode 'grant_type=password' \ --data-urlencode 'client_id={client_id}'` • HTTPS channel is required
Representations	**application/xml**, **application/json**

Resource: /{site}/customers/current/customergroups

Method	**GET**
Description	Returns all customer groups of current customer. Requires **customergroup** access role and a secured HTTPS channel.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/current/customergroups`
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Authentication	• Access is restricted to ROLE_CUSTOMERGROUP so the customer access token must be sent with the request. To obtain the token for a given customer an authorization call must be sent first: ◦ `curl --location 'https://localhost:9002/authorizationserver/oauth' \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'username={uid}' \ --data-urlencode 'password={pwd}' \ --data-urlencode 'client_secret={clientSecret\}' \ --data-urlencode 'grant_type=password' \ --data-urlencode 'client_id={client_id}'` • HTTPS channel is required
Representations	**application/xml**, **application/json**

Resource: /{site}/customers/current/login

Method	**POST**
Description	Change customer's login.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/current/login`
Request Parameters	• *<newLogin>* (Style: body, Required: true): Customer's new login. Customer login is case insensitive. • *<password>* (Style: body, Required: true): Customer's current password.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/customers/{uid}/customergroups

Method	**GET**
Description	Returns all customer groups of a given customer. Requires **admingroup** access role and a secured HTTPS channel.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/demo/customergroups`
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Authentication	• Access is restricted to ROLE_CUSTOMERGROUP so the customer access token must be sent with the request. To obtain the token for a given customer an authorization call must be sent first: ◦ `curl --location 'https://localhost:9002/authorizationserver/oauth' \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'username={uid}' \ --data-urlencode 'password={pwd}' \ --data-urlencode 'client_secret={clientSecret\}' \ --data-urlencode 'grant_type=password' \ --data-urlencode 'client_id={client_id}'` • HTTPS channel is required
Representations	**application/xml**, **application/json**

Resource Orders

A list of the available **Orders** resources.

Note

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

Resource: /{site}/orders

Method	**GET**
Description	Returns order history data for all orders placed by the current user for the current base store. Response contains orders search result displayed in several pages if needed.
Example URL	`https://localhost:9002/rest/v1/electronics/orders`
Authentication	This method requires basic HTTP authentication and is restricted to HTTPS channel.
Request Parameters	As URL query parameters: • *<statuses>* (Required: false) Filters only certain order statuses. It means: *<statuses>*=**CANCELLED, CHECKED_VALID** would only return orders with status **CANCELLED** or **CHECKED_VALID**. • *<currentPage>* (Required: false) Pagination attribute saying which page is requested • *<pageSize>* (Required: false) Pagination attribute saying the requested page size • *<sort>* (Required: false) Pagination attribute saying sort preference
Headers	• Accept(Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/orders/{code}

Method	**GET**
Description	Returns specific order details. Response contains a detailed order info object.
Example URL	`https://localhost:9002/rest/v1/electronics/orders/000001`
Authentication	This method requires basic HTTP authentication and is restricted to HTTPS channel.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource Products

A list of the available **Products** resources.

ℹ Note

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

Resource: /{site}/products

Method	**GET**

Description	Returns a list of products and additional data like available facets, available sort options, and pagination options. It can include spelling suggestions, for example:
	
	`<spellingSuggestion>`
	`<suggestion>sony</suggestion>`
	`<query>sony:relevance</query>`
	`</spellingSuggestion>`
	
	To make it work you need to:
	• Make sure *<enableSpellCheck>* on the `SearchQuery` is set to **true**. By default it should be already set to **true**.
	• Have indexed properties configured to be used for spell checking.

Example URL	`https://localhost:9002/rest/v1/electronics/products?query=a&pageSize=40`
Authentication	None
Request Parameters	• *<query>* (Style: query, Required: true): Serialized query, freetextsearch, facets. The format of serialized query: `freeTextSearch:sort:facetKey1:facetValue1:facetKey2:facetValue2` The query string needs to be URL-encoded and the client has to make sure that no `:` elements are part of the `freetextsearch`. The query value needs to be properly URL-encoded, too. • *<currentPage>* (Style: query, Required: false): The current result page requested. Optional. Default value: 0. • *<pageSize>* (Style: query, Required: false): The number of results returned per page. Optional. Default value: 20. • *<sort>* (Style: query, Required: false): Sorting method applied to the display search results.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/products/{product_code}

Method	**GET**
Description	Returns details of a single product specified by product code. Additional options can be expressed using **options** parameter.
Example URL	`https://localhost:9002/rest/v1/electronics/products/553637?options=DESCRIPTION`
Authentication	None
Request Parameters	• *<options>* (Style: query, Required: false):

	Defines level of the requested details. It can have a combination of the following values, separated by a comma: BASIC, DESCRIPTION, GALLERY, CATEGORIES, PROMOTIONS, STOCK, REVIEW, CLASSIFICATION, REFERENCES, PRICE. Keep in mind that the value needs to be properly URL-encoded. Example: `options=BASIC,REVIEW` requests the basic level of details plus reviews of a product.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/products/export/incremental

Method	**GET**
Description	Returns all products with pagination applied.
Example URL	`https://localhost:9002/rest/v1/electronics/products/export/incremental?timestamp=2012-03-28T07:50:49%2B00:00¤tPage=0&pageSize=10&options=BASIC`
Authentication	• Access is restricted to the **ROLE_TRUSTED_CLIENT** so that only registered and trusted API clients can gain it. • HTTPS channel is required
Request Parameters	• *<currentPage >*(Style: query, Required: false): The current result page requested, optional. Default value: 0. • *<timestamp>* in RFC-8601 format • *<pageSize>* (Style: query, Required: false): The number of results returned per page. Optional. Default value: unlimited. • *<options>* (Style: query, Required: false): It defines the level of details requested. This parameter can have a combination of the following values, separated by a comma : BASIC, DESCRIPTION, GALLERY, CATEGORIES, PROMOTIONS, STOCK, REVIEW, CLASSIFICATION, REFERENCES. Keep in mind that the value needs to be properly URL-encoded. Example: `options=BASIC|REVIEW` requests the basic level of details plus reviews of a product. ||
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/products/export/full

Method	**GET**
Description	Returns all products with pagination applied.

Example URL	`https://localhost:9002/rest/v1/electronics/products/export/full?currentPage=0&pageSize=10&options=BASIC`
Authentication	• Access is restricted to the **ROLE_TRUSTED_CLIENT** so that only registered and trusted API clients can gain it. • HTTPS channel is required
Request Parameters	• *<currentPage >*(Style: query, Required: false): The current result page requested, optional. Default value: 0. • *<pageSize>* (Style: query, Required: false): The number of results returned per page. Optional. Default value: unlimited. • *<options>* (Style: query, Required: false): Defines the level of details requested. It can have a combination of the following values, separated by a comma : BASIC, DESCRIPTION, GALLERY, CATEGORIES, PROMOTIONS, STOCK, REVIEW, CLASSIFICATION, REFERENCES. Keep in mind that the value needs to be properly URL-encoded. Example: `options=BASIC|REVIEW` requests the basic level of details plus reviews of a product.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/products/suggest

Method	**GET**

Description	Related to spell check correction topic. It is a list of all available suggestions related to the given *<term>* and limits the results to a given value of the *<max>* parameter. It returns the result in the XML format.
	
	`<suggestions>`
	`<suggestion>`
	`<value>tripod</value>`
	`</suggestion>`
	`<suggestion>`
	`<value>tape</value>`
	`</suggestion>`
	`</suggestions>`

Example URL	`https://localhost:9002/rest/v1/electronics/products/suggest?term=t&max=3`
Request Parameters	• *<term>* (Style: body, Required: true): Specified term • *<max>* (Style: body, Required: false): Specifies the limit of results. Default value: 10.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.

Resource: /{site}/products/{code}/reviews

Method	**POST**
Description	Method creates new customer review as an anonymous user. Method responses with a review data.
Example URL	`https://localhost:9002/rest/v1/electronics/product/816780/reviews`
Request Parameters	Need to be provided as URL encoded post body. • *<rating>* Required. Value needs to be between 1 and 5. • *<alias>* Optional • *<headline>* Required • *<comment>* Required
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/products/expressUpdate

Method	**GET**
Description	Method returns products added to the express update feed. The queue is cleared using the defined cronjob.
Example URL	`https://localhost:9002/rest/v1/electronics/products/expressUpdate?timestamp=2013-06-21T11%3A30%3A21%2B01%3A00`
Authentication	• Access is restricted to the **ROLE_TRUSTED_CLIENT** so that only registered and trusted API clients can gain it. • HTTPS channel is required
Request Parameters	• *<timestamp>*: Required in RFC-8601 format. Only items newer than the given parameter are retrieved from the queue. • *<catalog>*: Optional. Only products from this catalog are returned. Format: `catalogId:catalogVersion`
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/products/{code}/nearLocation

Method	**GET**
Description	Method returns product's stock level sorted by distance from specific location passed by the free-text parameter.

Example URL	`https://localhost:9002/rest/v1/electronics/products/816780/nearLocation?location=Tokio&pageSize=5¤tPage=2`
Request Parameters	Need to be provided as URL encoded post body. • *<location>*: Required. Free-text location parameter. • *<pageSize>*: Optional • *<currentPage>*: Optional
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/products/{code}/nearLatLong

Method	**GET**
Description	Method returns product's stock level sorted by distance from specific location passed by the longitude and latitude parameters.
Example URL	`https://localhost:9002/rest/v1/electronics/products/816780/nearLatLong?latitude=35.6816951&longitude=139.7650482&pageSize=5¤tPage=2`
Request Parameters	Need to be provided as URL encoded post body. • *<longitude>*: Required. Longitude location parameter. • *<latitude>*: Required. Latitude location parameter. • *<pageSize>*: Optional • *<currentPage>*: Optional
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Miscellaneous Resources

A list of available **Miscellaneous resources** that do not belong to other groups.

ℹ Note

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

Resource: /{site}/languages

Method	**GET**
Description	List all available languages (all base store's languages). In case of an empty languages list for the base store, it returns list of all languages in the system.
Example URL	`https://localhost:9002/rest/v1/electronics/languages`

Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.

Resource: /{site}/currencies

Method	**GET**
Description	List all available currencies (all usable currencies for the current store). In case of an empty currencies list for stores, it returns the list of all currencies in the system.
Example URL	`https://localhost:9002/rest/v1/electronics/currencies`
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.

Resource: /{site}/deliverycountries

Method	**GET**
Description	List all supported delivery countries for the current store. The list is sorted alphabetically.
Example URL	`https://localhost:9002/rest/v1/electronics/deliverycountries`
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.

Resource: /{site}/cardtypes

Method	**GET**
Description	List supported payment card types.
Example URL	`https://localhost:9002/rest/v1/electronics/cardtypes`
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.

Resource: /{site}/titles

Method	**GET**
Description	List all localized titles.
Example URL	`https://localhost:9002/rest/v1/electronics/titles`
Headers	• Accept (Style: header, Required: true):

	Accept header needs to be sent with each request. It can be application/xml or **application/json**.

Resource Stores

A list of available **Stores** resources.

Note

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

Resource: /{site}/stores

Method	**GET**

Description	Lists all store locations that are near the location specified in a query or by latitude and longitude.
	
	`<searchResult>`
	`<results />`
	`<pagination>`
	`<pageSize>10</pageSize>`
	`<currentPage>0</currentPage>`
	`<totalResults>49</totalResults>`
	`<totalPages>5</totalPages>`
	`</pagination>`
	`<boundEastLongitude>135.2738964</boundEastLongitude>`
	`<boundSouthLatitude>33.5829778</boundSouthLatitude>`
	`<sourceLatitude>48.1366069</sourceLatitude>`
	`<boundNorthLatitude>62.69023599999999</boundNorthLatitude>`
	`<sourceLongitude>11.5770851</sourceLongitude>`
	`<locationText>munich</locationText>`
	`<boundWestLongitude>-112.1197262</boundWestLongitude>`
	`</searchResult>`

Example URL	`https://localhost:9002/rest/v1/electronics/stores?query=munich|https://localhost:9002/rest/v1/electronics/stores?query=munich`
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Request Parameters	Not applicable.
Representations	**application/xml**, **application/json**

Resource: /{site}/stores/{name}

Method	**GET**
Description	Returns store location by its unique name.
Example URL	`https://localhost:9002/rest/v1/electronics/stores/Chiba`
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.

Request Parameters	Not applicable.
Representations	**application/xml**, **application/json**

Resource Promotions

A list of available **Promotions** resources.

ℹ Note

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

Resource: /{site}/promotions

Method	**GET**
Description	Method returns promotion list
Example URL	• `https://localhost:9002/rest/v1/electronics/promotions?type=all` • `https://localhost:9002/rest/v1/electronics/promotions?type=all&promotionGroup=electronicsPromoGrp`

Authentication	• Access is restricted to the ROLE_CLIENT and ROLE_TRUSTED_CLIENT. Only registered API clients can gain it. The client access token must be sent with the request. To obtain the token for a given *<client_id,>* you need to send an authorization call first:
	◦ POST - `https://localhost:9002/rest/oauth/token`
	Header:
	`Content-Type: application/x-www-form-urlencoded`
	Request body:
	`client_id=${clientId}&client_secret=${clientSecret}&grant_type=client_credentials`
	• HTTPS channel is required

Request Parameters	• *<type>* (Required: true): Defines what type of promotions should be returned. Values supported for that parameter are: ◦ *<all>*: All available promotions are returned ◦ *<product>*: Only product promotions are returned ◦ *<order>*: Only order promotions are returned • *<promotionGroup>* (Required: false): Only promotions from this group are returned
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource: /{site}/promotions/{code}

Method	**GET**

Description	Returns details of a single promotion specified by a promotion code. Additional options can be expressed by using the *<options>* parameter.
Example URL	`https://localhost:9002/rest/v1/electronics/promotions/MultiBuy?options=EXTENDED`
Authentication	Access is restricted to the **ROLE_CLIENT** and ROLE_TRUSTED_CLIENT. This method requires client HTTP authentication and is restricted to HTTPS channel.
Request Parameters	• *<options>* (Style: query, Required: false): Defines level of the requested details. It can have a combination of the following values, separated by comma: **BASIC**, **EXTENDED**. The *<BASIC>* value is included by default. The value needs to be properly URL-encoded. Example: `options=BASIC,EXTENDED` requests the basic level of details plus extended.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource Vouchers

A list of available **Vouchers** resources.

ℹ Note

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

Resource: /{site}/vouchers/{code}

Method	**GET**
Description	Returns details of a single voucher specified by a voucher code.
Example URL	`https://localhost:9002/rest/v1/electronics/vouchers/abc-9PSW-EDH2-RXKA`

Authentication	• Access is restricted to the **ROLE_CLIENT** so that only registered API clients can gain it. The client access token must be sent with the request. To obtain the token for a given *<client_id>*, you need to send an authorization call first: POST - `https://localhost:9002/authorizationserver/oauth/token`
	Header:
	`Content-Type: application/x-www-form-urlencoded`
	Request body:
	`client_id=${clientId}&client_secret=${clientSecret}&grant_type=client_credentials`
	• HTTPS channel is required

Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be application/xml or application/json.
Representations	**application/xml**, **application/json**

Resource WCMS Components

A list of the available **WCMS Components** resources.

ℹ Note

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

Resource: /{site}/wcms/images/{code}

Method	**GET**
Validity	5.0.1
Description	Returns details of a single CMSImageComponent specified by code.
Example URL	`https://localhost:9002/rest/v1/electronics/wcms/images/image_code`
Request Parameters	• Accept (Style: header, Required: true): It can be **application/xml**,**application/json**, **image/*** If an Accept header is set to image/* and component is not visible, then the server returns the 404 (Not Found) error. If an Accept header field is present, and if the server cannot send a response which is acceptable according to the combinedAccept field value, then the server returns the 406 (Not Acceptable) error. If no Accept header field is present, then the JSON representation is returned.
Representations	• Accept (Style: header, Required: true): It can be **application/xml**, **application/json**, **image/***

Resource: /{site}/wcms/paragraphs/{code}

Method	**GET**
Validity	5.0.1
Description	Returns details of a single `CMSParagraphComponent` specified by code.
Example URL	`https://localhost:9002/rest/v1/electronics/wcms/paragraphs/paragraph_code`
Request Parameters	• Accept header needs to be sent with each request. It can be **application/xml**, **application/json**, **text/html** If an Accept header is set to text/html and component is not visible, then the server returns the 404 (Not Found) error. If an Accept header field is present, and if the server cannot send a response which is acceptable according to the combined Accept field value, then the server returns the 406 (Not Acceptable) error. If no Accept header field is present, then the JSON representation is returned.
Representations	• Accept (Style: header, Required: true): It can be **application/xml**, **application/json**, **text/html**

OCC v1 Sample Flows

Sample flow calls for the SAP Commerce OCC API in **v1**

Overview

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

In general, REST requests are stateless, but the `ycommercewebservices` extension uses a standard HTTP session to simplify calls when operating on a customer cart. It is the same approach as the end-user session on a web front end.

Despite that a session is needed only for a few calls, it is recommended to always use it. The server creates a new session and sends back a session identifier in response to the first request. It is good practice to store a session ID and use it in the following calls every time the server generates it. Using a session ID from the beginning also simplifies the application because there is no need to analyze if it should be remembered or not.

HTTP Session

A session is a sequence of operations (request, response) done by one user. Usually these operations share and use some common data, for example a customer cart or information about a logged-in user. The session is started during the first request and is finished when the user logs out or after a long time of inactivity.

HTTP protocol is not session-aware and particular requests are linked to its session based on the session identifier which is sent as a cookie with each request.

Every time a server sends a new session identifier (JSESSIONID cookie in java applications), the client application should remember it and use it in the following calls, otherwise a new session is created each time.

Example:

End-user would like to insert a product to a cart. The client application sends a request:

Request #1

```
POST https: //localhost:9002/rest/v1/electronics/cart

POST data:
code=3429337

Cookie data:
```

This is the first request, so the client application does not have any cookies. The server does not receive a JSESSIONID cookie and assumes that this is a new session. It creates the session and sends its identifier to the client application in **Set-Cookie** header:

```
Set-Cookie: JSESSIONID=911CFAE8289BFB29AB347EC95B874EF7; Path=/rest/; HttpOnly
```

The client application should remember this cookie and resend it with subsequent requests to this server. So for example, if an end-user would like to get his cart contents, the application sends a request:

Request #2

```
GET https: //localhost:9002/rest/v1/electronics/cart

Cookie Data:
JSESSIONID=911CFAE8289BFB29AB347EC95B874EF7
```

The server receives the JSESSIONID and finds the appropriate session that contains the cart contents of the current user and returns it as the response.

Cookie Attributes

Cookies are defined as a key-value pair with some additional attributes:

```
Set-Cookie: JSESSIONID=911CFAE8289BFB29AB347EC95B874EF7; Path=/rest/; Secure; HttpOnly
```

- *<Path>* attribute sets the cookie 'visibility range'. So in this case, the JSESSIONID cookie will be used only with requests to URLs starting with **/rest**.
- *<Secure>* flag indicates that this cookie may be transmitted only with a secure communication channel.

Session ID Change

Session identifier may change when a new security level changes, in particular when:

- The user logs in

  This is protection against **Session fixation** attack.

- The protocol changes from HTTPS to HTTP

  This is protection against the **Man-in-the-middle** attack. When a cookie is generated while the HTTPS protocol is used, the server sets a **Secure** flag in this cookie. It then cannot be used with the HTTP protocol as described before, so a new JSESSIONID is generated in this case. But in general, when an application switches to an HTTPS protocol, it should not switch back to HTTP until the session ends (user logout, etc.). **The simplest rule is to always use HTTPS channel**.

HTTP Persistent Connection

For performance reasons, it is important to reuse existing connections, especially when subsequent communication is expected. It is very important when using the HTTPS protocol because creating a secure connection needs more time and more network communication.

When using an HTTP 1.0 client that supports a persistent connection, send the following header with the request:

```
Connection: Keep-Alive
```

If the server puts the same header in the response, the client may use this connection with the following calls.

In HTTP 1.1, connections are considered persistent by default unless declared otherwise.

Despite the connection being 'persistent', it does not mean that is kept alive forever. It should be closed by the client or server after a period of inactivity.

Multiple Site Issues

It is possible to have multiple sites supported by one server instance. In every REST call, the particular site is specified in the URL. For example:

`GET https://localhost:9002/rest/v1/electronics/cart`

`GET https://localhost:9002/rest/v1/apparel-uk/cart`

It is important that some entities are common while others are separated depending on the site.

Users

User accounts are common, so it is not possible to have two user accounts with the same login linked to other sites. Also, a user created in one site will be able to use the same account in other sites.

The OAuth access token is also connected to the user and is not dependent on the site. This is why access tokens are obtained using a URL which does not contain the site ID:

`GET https://localhost:9002/authorizationserver/oauth/token`

Once obtained, add the access token to the header, for example:

```
header 'Authorization: Bearer abc738b9-b930-4956-b2f0-f81665f78785'
```

The access token may then be used with REST calls for various sites:

GET `https://localhost:9002/rest/v1/electronics/customers/current`

GET `https://localhost:9002/rest/v1/apparel-uk/customers/current`

Orders

Order history is separated depending on the store. For example, when a user calls:

GET `https://localhost:9002/rest/v1/electronics/orders?access_token=abc738b9-b930-4956-b2f0-f81665f78785`

It returns orders placed in the electronics store (connected with the requested electronics site).

Carts

A user's cart is connected with a current user's session. In order to make it work properly across multiple sites it is very important to use separate HTTP sessions for each site.

To learn more about single steps for buying scenarios see: Single Steps of Buying Process in v1.

To go through complete scenarios visit: Customer Buying Process Scenarios in v1

Single Steps of Buying Process in v1

Examples of single steps of Buying Process Scenarios for the Omni Commerce Connect (OCC) in v1.

Overview

The sections below present sets of available calls that may be executed using the v1 API. The calls have been grouped into topics reflecting their purpose, however they should be treated as a separate examples rather than complete scenarios. For complete business scenarios, refer to: Customer Buying Process Scenarios in v1.

Register New Customer

Method	URL	Body(url encoded)
POST	`https://localhost:9002/authorizationserver/oauth/token`	client_id=$CLIENT_ID$&client_secret=$CLIENT_SECRET$&grant_type=c
POST	`https://localhost:9002/rest/v1/electronics/customers`	login=john.doe@mail.com&password=your_password &firstName=John&lastName=Doe&titleCode=mr

Get Token for Customer

Method	URL	Body(url encoded)
POST	`https://localhost:9002/authorizationserver/oauth/token`	client_id=$CLIENT_ID$&client_secret=$CLIENT_SECRET$ &grant_type=password&username=john.doe@mail.com&password=your_

Manage Customer Profile

Scenario steps:

- Get token for customer
- Manage addresses
- Manage payment information
- Update customer profile
- Change customer password

Method	URL	Body(url encoded)
POST	https://localhost:9002/authorizationserver/oauth/token	client_id=$CLIENT_ &grant_type=passw
GET	https://localhost:9002/rest/v1/electronics/customers/current/addresses	
POST	https://localhost:9002/rest/v1/electronics/customers/current/addresses	firstName=John&las 19&town=Osaka&po 27
PUT	https://localhost:9002/rest/v1/electronics/customers/current/addresses/default/8796158590999	
GET	https://localhost:9002/rest/v1/electronics/customers/current/paymentinfos	saved=true
GET	https://localhost:9002/rest/v1/electronics/customers/current/paymentinfos/8796191359018	
DELETE	https://localhost:9002/rest/v1/electronics/customers/current/paymentinfos/8796191359018	
PUT	https://localhost:9002/rest/v1/electronics/customers/current/profile	titleCode=dr&firstN
PUT	https://localhost:9002/rest/v1/electronics/customers/current/password	old=your_password&

Search for a Product

Method	URL	Parameters	Descr
GET	https://localhost:9002/rest/v1/electronics/products	query= :price-asc:category:575	Return produ from catego with ID 575 so by ascen price.

Method	URL	Parameters	Descr
GET	https://localhost:9002/rest/v1/electronics/products	query= :name-desc:brand:brand_10	Returr produ from b with ID brand sortec desce name.
GET	https://localhost:9002/rest/v1/electronics/products	query=camera&pageSize=40	Searcl for 'ca produ We ca facets the respor narrov searcl results
GET	https://localhost:9002/rest/v1/electronics/products	query= camera:relevance:category:575	Searcl for 'ca produ categ with ID 575.
GET	https://localhost:9002/rest/v1/electronics/products	query= camera:relevance:category:575:brand:brand_10	Searcl for 'ca produ from categ with ID 575 ar brand ID of brand
GET	https://localhost:9002/rest/v1/electronics/products	query= camera:relevance:category:575:brand:brand_10:price:$500-$999.99	Searcl for 'ca produ categ with ID 575 ar brand ID of brand and w price betwe 500 a 999.9

Search for Nearest Store Having Particular Product in Stock

Scenario steps:

- Search for product
- Search for nearest store having this product in stock

Method	URL	Parameters	Description
GET	https://localhost:9002/rest/v1/electronics/products	query=EOS	Searches for products containing the

Method	URL	Parameters	Description
			'EOS' string in the name.
GET	https://localhost:9002/rest/v1/electronics/products/1382080		Gets details about 'Canon EOS 450D'.
GET	https://localhost:9002/rest/v1/electronics/products/1382080/nearLocation	location=Tokio	Searches for stores having 'Canon EOS 450D' in stock. The result is sorted by distance from given location.

Adding Product to Cart

⚠ Caution

Remember to set **JSESSIONID** in cookies. It is required for requests that change the cart. You will get it in Set-Cookie header in response.

Method	URL	Parameters	Description
GET	https://localhost:9002/rest/v1/electronics/cart		Gets an empty cart.
POST	https://localhost:9002/rest/v1/electronics/cart/entry	code=3429337	Adds product with code 3429337 to the cart.
POST	https://localhost:9002/rest/v1/electronics/cart/entry	code=1934795&qty=2	Adds two products with code 1934795 to the cart.
POST	https://localhost:9002/rest/v1/electronics/cart/entry	code=1934795&qty=1&storeName=Nakano	Adds one product with code 1934795 to the cart. Product will be picked up in Nakano store.
PUT	https://localhost:9002/rest/v1/electronics/cart/entry/0	qty=3	Changes quantity for first product in the cart from 1 to 3.
PUT	https://localhost:9002/rest/v1/electronics/cart/entry/2/store	storeName=Tokio	Changes store where product will be picked up.
DELETE	https://localhost:9002/rest/v1/electronics/cart/entry/2/store		Resets the store where the entry should be picked up. Entry is

Method	URL	Parameters	Description
			delivered by the selected delivery method.
DELETE	`https://localhost:9002/rest/v1/electronics/cart/entry/0`		Removes first product from the cart.
GET	`https://localhost:9002/rest/v1/electronics/cart`		Gets an updated cart.

Checkout Process

Checkout with Creating New Address and Payment Information for Customer

Assumption: a customer is registered and has a cart with products for current session.

> ⚠ **Caution**
> Remember to set **JSESSIONID** in cookies. It is required for requests that make changes to the cart. You will get it in the Set-Cookie header in the response.

Scenario steps:

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

Method	URL	Request Body
POST	`https://localhost:9002/rest/v1/electronics/customers/current/addresses`	
PUT	`https://localhost:9002/rest/v1/electronics/cart/address/delivery/8796158787607`	
GET	`https://localhost:9002/rest/v1/electronics/cart/deliverymodes`	
PUT	`https://localhost:9002/rest/v1/electronics/cart/deliverymodes/premium-gross`	
POST	`https://localhost:9002/rest/v1/electronics/cart/authorize`	
POST	`https://localhost:9002/rest/v1/electronics/cart/placeorder`	
POST	`https://localhost:9002/rest/v1/electronics/cart/paymentinfo`	Should set header Content-Type to : `application/x-www-form-urlenc`

Method	URL	Request Body
		accountHolderName: "John Doe" cardNumber: "4111111111111111" cardType: "visa" expiryMonth: "01" expiryYear: "2017" saved: "true" defaultPaymentInfo: "true" billingAddress.titleCode: "mr" billingAddress.firstName: "Joh billingAddress.lastName: "Doe" billingAddress.line1: "Toyosak billingAddress.line2: "3-16-19 billingAddress.postalCode: "53 billingAddress.town: "Osaka" billingAddress.country.isocode billingAddress.region.isocode:

Checkout with Setting Existing Address and Payment Information for Customer

Assumption: a customer is registered and has a cart with products for current session.

> ⚠ **Caution**
> Remember to set **JSESSIONID** in cookies. It is required for requests that make changes to the cart. You will get it in the Set-Cookie header in the response.

Scenario steps :

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

Method	URL	Parameters	Authorization	De
GET	`https://localhost:9002/rest/v1/electronics/cart/deliverymodes`		OAUTH token for customer	Ge de Re va no ac
GET	`https://localhost:9002/rest/v1/electronics/customers/current/addresses`		OAUTH token for customer	Ge cu ac
PUT	`https://localhost:9002/rest/v1/electronics/cart/address/delivery/8796158754839`		OAUTH token for customer	Se wi 87 as ac
GET	`https://localhost:9002/rest/v1/electronics/cart/deliverymodes`		OAUTH token for customer	Ge de Th sh a v

Method	URL	Parameters	Authorization	De
PUT	https://localhost:9002/rest/v1/electronics/cart/deliverymodes/premium-gross		OAUTH token for customer	Se gr cu m
GET	https://localhost:9002/rest/v1/electronics/customers/current/paymentinfos	saved=true	OAUTH token for customer	Ge pa int wa sa ch
PUT	https://localhost:9002/rest/v1/electronics/cart/paymentinfo/8796158591018		OAUTH token for customer	As pa int ca 87 (re pr
POST	https://localhost:9002/rest/v1/electronics/cart/authorize	securityCode=123	OAUTH token for customer	Au ca
POST	https://localhost:9002/rest/v1/electronics/cart/placeorder		OAUTH token for customer	Pl

Get Customer Orders

Method	URL	Body(url encoded)
POST	https://localhost:9002/authorizationserver/oauth/token	client_id=$CLIENT_ID$&client_secret=$CLIENT_SECRET$ &grant_type=password&username=john.doe@mail.com&password=
GET	https://localhost:9002/rest/v1/electronics/orders	
GET	https://localhost:9002/rest/v1/electronics/orders/00001002	

Store Search, Display Store Details

Scenario steps:

- Search for store
- Display store details

Method	URL	Parameters	Description
GET	https://localhost:9002/rest/v1/electronics/stores	query=Tokio	Searches for stores that are near the location specified by the parameters (in this case near Tokio city). Returns list of stores.

Method	URL	Parameters	Description
GET	https://localhost:9002/rest/v1/electronics/stores/Nakano		Gets store details for a store identified by name (in this case "Nakano").

Cart Restoration

Scenario steps:

- Search for products, session "1" is created automatically
- Get product details
- Insert product(s) to cart, cart "1" is created automatically
- Get cart in order to show it and remember cart **guid**
- Session 1 expires after long inactivity
- Get cart in order to show it, session "2" and cart "2" are created automatically
- Try to restore cart "1" to the current session "2"
- Get cart "1"

Method	URL	Parameters	Cookies
GET	https://localhost:9002/rest/v1/electronics/products	query=camera&pageSize=40	
GET	https://localhost:9002/rest/v1/electronics/products/489702		JSESSIONID obtained in step 1
POST	https://localhost:9002/rest/v1/electronics/cart/entry	code=489702	JSESSIONID obtained in step 1
GET	https://localhost:9002/rest/v1/electronics/cart		JSESSIONID obtained in step 1

Method	URL	Parameters	Cookies
-	-	-	-
GET	https://localhost:9002/rest/v1/electronics/cart		JSESSIONID obtained in step 1
GET	https://localhost:9002/rest/v1/electronics/cart/restore	guid=02017b6dad8834ba6f3fd26eb9ab4f270e9b1858	JSESSIONID obtained in step 6
GET	https://localhost:9002/rest/v1/electronics/cart		JSESSIONID obtained in step 7

Customer Buying Process Scenarios in v1

The following provides full Customer Buying Process Scenarios for Omni Commerce Connect (OCC) in v1.

Overview

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

Registered Customer Scenario

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).

⚠ Caution

Remember to set **JSESSIONID** in cookies. It is required for requests that make changes to the cart. You get it in the Set-Cookie header in the response.

Scenario steps:

- Search product
- Get product details
- Add product to cart (basket)
- Get access token for customer
- Checkout
- Display placed order for customer
- Log out

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 (url encoded)
GET	`https://localhost:9002/rest/v1/electronics/products`	query=camera&pageSize=40
GET	`https://localhost:9002/rest/v1/electronics/products /489702`	
POST	`https://localhost:9002/rest/v1/electronics/cart/entry`	code=489702
POST	`https://localhost:9002/authorizationserver/oauth/token`	client_id=$CLIENT_ID$&client_secret=$ &grant_type=password&username=johr
GET	`https://localhost:9002/rest/v1/electronics/cart`	
GET	`https://localhost:9002/rest/v1/electronics/customers/current/addresses`	
PUT	`https://localhost:9002/rest/v1/electronics/cart/address/delivery/8796158754839`	
GET	`https://localhost:9002/rest/v1/electronics/cart/deliverymodes`	
PUT	`https://localhost:9002/rest/v1/electronics/cart/deliverymodes/premium-gross`	

Method	URL	Body (url encoded)
GET	https://localhost:9002/rest/v1/electronics/customers/current/paymentinfos	saved=true
PUT	https://localhost:9002/rest/v1/electronics/cart/paymentinfo/8796158591018	
POST	https://localhost:9002/rest/v1/electronics/cart/authorize	securityCode=123
POST	https://localhost:9002/rest/v1/electronics/cart/placeorder	
GET	https://localhost:9002/rest/v1/electronics/orders	
GET	https://localhost:9002/rest/v1/electronics/orders/00001002	
POST	https://localhost:9002/rest/v1/customers/current/logout	

New Customer Scenario

Prerequisites: a customer is not registered in the store.

⚠ Caution

Remember to set **JSESSIONID** in cookies. It is required for requests that make changes to the cart. You will get it in the Set-Cookie header in the response.

Scenario steps:

- Register a new customer and get access token
- Create a new address for the customer
- Search for a specific product
- Get product details
- Add products to the cart
- Checkout: set delivery address, set delivery mode, set credit card information (paymentinfo), card authorization
- Place order
- Display placed order for the customer
- Log out

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	Request Body
POST	https://localhost:9002/authorizationserver/oauth/token	
POST	https://localhost:9002/rest/v1/electronics/customers	
POST	https://localhost:9002/authorizationserver/oauth/token	
POST	https://localhost:9002/rest/v1/electronics/customers/current/addresses	
GET	https://localhost:9002/rest/v1/electronics/products	
GET	https://localhost:9002/rest/v1/electronics/products /3429337	
POST	https://localhost:9002/rest/v1/electronics/cart/entry	
PUT	https://localhost:9002/rest/v1/electronics/cart/address/delivery/8796158754839	
GET	https://localhost:9002/rest/v1/electronics/cart/deliverymodes	
PUT	https://localhost:9002/rest/v1/electronics/cart/deliverymodes/premium-gross	
POST	https://localhost:9002/rest/v1/electronics/cart/authorize	
POST	https://localhost:9002/rest/v1/electronics/cart/placeorder	
GET	https://localhost:9002/rest/v1/electronics/orders	
GET	https://localhost:9002/rest/v1/electronics/orders/00001002	
POST	https://localhost:9002/rest/v1/customers/current/logout	

Method	URL	Request Body
POST	https://localhost:9002/rest/v1/electronics/cart/paymentinfo	Should set Head Content-Type to: application/x-www-form-urlenc accountHolderName: "John Doe" cardNumber: "4111111111111111" cardType: "visa" expiryMonth: "01" expiryYear: "2017" saved: "true" defaultPaymentInfo: "true" billingAddress.titleCode: "mr" billingAddress.firstName: "Joh billingAddress.lastName: "Doe" billingAddress.line1: "Toyosak billingAddress.line2: "3-16-19 billingAddress.postalCode: "53 billingAddress.town: "Osaka" billingAddress.country.isocode billingAddress.region.isocode:

Pick Up in Store Scenario

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

⚠ Caution

Remember to set **JSESSIONID** in cookies. It is required for requests that make changes to the cart. You will get it in the Set-Cookie header in the response.

Scenario steps:

- Get an access token for the customer
- Add products that will be picked up in the store
- Checkout and set 'pickup' as the delivery mode
- Display placed order for the customer
- Log out

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/x-www-form-urlenc
POST	https://localhost:9002/authorizationserver/oauth/token	client_id=$CLIENT_ID$&client_secret=$CLIEN
POST	https://localhost:9002/rest/v1/electronics/cart/entry	code=1382080
PUT	https://localhost:9002/rest/v1/electronics/cart/entry/0/store	storeName=Nakano
POST	https://localhost:9002/rest/v1/electronics/cart/entry	code=1934795&qty=1&storeName=Nakano

Method	URL	Body Parameters
		Content-Type: application/x-www-form-urlenc
GET	https://localhost:9002/rest/v1/electronics/cart	
GET	https://localhost:9002/rest/v1/electronics/cart/deliverymodes	
PUT	https://localhost:9002/rest/v1/electronics/cart/deliverymodes/pickup	
GET	https://localhost:9002/rest/v1/electronics/customers/current/paymentinfos	saved=true
PUT	https://localhost:9002/rest/v1/electronics/cart/paymentinfo/8796158591018	
POST	https://localhost:9002/rest/v1/electronics/cart/authorize	securityCode=123
POST	https://localhost:9002/rest/v1/electronics/cart/placeorder	
GET	https://localhost:9002/rest/v1/electronics/orders	
GET	https://localhost:9002/rest/v1/electronics/orders/00002036	
POST	https://localhost:9002/rest/v1/customers/current/logout	

Pick Up in Store with Store Consolidation - Scenario

Prerequisitess:

- Customer is already registered in the store and has defined all information needed for the checkout process (such as address and payment information)
- **acceleratorwebservicesaddon** is installed

⚠ Caution

Remember to set **JSESSIONID** in cookies. It is required for requests that make changes to the cart. You will get it in the Set-Cookie header in the response.

Scenario steps:

- Get an access token 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 consolidate store functionality
- Checkout
- Log out

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/x-www-form-urlenc
POST	https://localhost:9002/authorizationserver/oauth/token	client_id=$CLIENT_ID$&client_secret=$CLIEN
POST	https://localhost:9002/rest/v1/electronics/cart/entry	code=2053226&qty=1&storeName=Shinbashi
POST	https://localhost:9002/rest/v1/electronics/cart/entry	code=1934793&qty=1&storeName=Nakano
GET	https://localhost:9002/rest/v1/electronics/cart	
GET	https://localhost:9002/rest/v1/electronics/cart/consolidate	
POST	https://localhost:9002/rest/v1/electronics/cart/consolidate	storeName=Shinbashi
GET	https://localhost:9002/rest/v1/electronics/cart/deliverymodes	
PUT	https://localhost:9002/rest/v1/electronics/cart/deliverymodes/pickup	
GET	https://localhost:9002/rest/v1/electronics/customers/current/paymentinfos	saved=true

Method	URL	Body Parameters
		Content-Type: application/x-www-form-urlenc
PUT	https://localhost:9002/rest/v1/electronics/cart/paymentinfo/8796158591018	
POST	https://localhost:9002/rest/v1/electronics/cart/authorize	securityCode=123
POST	https://localhost:9002/rest/v1/electronics/cart/placeorder	
POST	https://localhost:9002/rest/v1/customers/current/logout	

Combining Delivery Mode With Pick Up in Store - Scenario

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

⚠ Caution

Remember to set **JSESSIONID** in cookies. It is required for requests that make changes in the cart. You will get it in the Set-Cookie header in the response.

Scenario steps:

- Get an access token for the customer
- Add a product that will be delivered
- Add a product that will be picked up in a store
- Checkout
- Display placed order for the customer
- Log out

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/x-www-form
POST	https://localhost:9002/authorizationserver/oauth/token	client_id=$CLIENT_ID$&client_secret=$
POST	https://localhost:9002/rest/v1/electronics/cart/entry	code=1382080
POST	https://localhost:9002/rest/v1/electronics/cart/entry	code=1934795&qty=1&storeName=Nak
GET	https://localhost:9002/rest/v1/electronics/cart	

Method	URL	Body Parameters
		Content-Type: application/x-www-form
GET	https://localhost:9002/rest/v1/electronics/customers/current/addresses	
PUT	https://localhost:9002/rest/v1/electronics/cart/address/delivery/8796192014359	
GET	https://localhost:9002/rest/v1/electronics/cart/deliverymodes	
PUT	https://localhost:9002/rest/v1/electronics/cart/deliverymodes/premium-gross	
GET	https://localhost:9002/rest/v1/electronics/customers/current/paymentinfos	saved=true
PUT	https://localhost:9002/rest/v1/electronics/cart/paymentinfo/8796158591018	
POST	https://localhost:9002/rest/v1/electronics/cart/authorize	securityCode=123
POST	https://localhost:9002/rest/v1/electronics/cart/placeorder	
GET	https://localhost:9002/rest/v1/electronics/orders	
GET	https://localhost:9002/rest/v1/electronics/orders/00002036	
POST	https://localhost:9002/rest/v1/customers/current/logout	

Guest Checkout Scenario with Creating Full Account

⚠ Caution

Remember to set **JSESSIONID** in cookies. It is required for requests that make changes to the cart. You will get it in the Set-Cookie header in the response.

Scenario steps:

- Search a product
- Get the product details
- 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 order list

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/x-www-form
GET	https://localhost:9002/rest/v1/electronics/products	
GET	https://localhost:9002/rest/v1/electronics/products /489702	
POST	https://localhost:9002/rest/v1/electronics/cart/entry	
POST	https://localhost:9002/authorizationserver/oauth/token	client_id=$CLIENT_ID$&client_secret=$
POST	https://localhost:9002/rest/v1/electronics/customers/current/guestlogin	
POST	https://localhost:9002/rest/v1/electronics/customers/current/addresses	
PUT	https://localhost:9002/rest/v1/electronics/cart/address/delivery/8796158754839	
GET	https://localhost:9002/rest/v1/electronics/cart/deliverymodes	
PUT	https://localhost:9002/rest/v1/electronics/cart/deliverymodes/premium-gross	
POST	https://localhost:9002/rest/v1/electronics/cart/authorize	securityCode=123
POST	https://localhost:9002/rest/v1/electronics/cart/placeorder	
POST	https://localhost:9002/rest/v1/electronics/customers/current/convert	password=your_password

Method	URL	Body Parameters
		Content-Type: application/x-www-form
POST	https://localhost:9002/rest/v1/customers/current/logout	
	...	
POST	https://localhost:9002/authorizationserver/oauth/token	client_id=$CLIENT_ID$&client_secret=$ &grant_type=password&username=johr
GET	https://localhost:9002/rest/v1/electronics/orders	
GET	https://localhost:9002/rest/v1/electronics/orders/00002009	
POST	https://localhost:9002/rest/v1/electronics/cart/paymentinfo	accountHolderName: "John Doe" cardNumber: "4111111111111111" cardType: "visa" expiryMonth: "01" expiryYear: "2017" saved: "true" defaultPaymentInfo: "true" billingAddress.titleCode: "mr" billingAddress.firstName: "Joh billingAddress.lastName: "Doe" billingAddress.line1: "Toyosak billingAddress.line2: "3-16-19 billingAddress.postalCode: "53 billingAddress.town: "Osaka" billingAddress.country.isocode billingAddress.region.isocode:

Guest Checkout Scenario with Checking Order Status

⚠ Caution

Remember to set **JSESSIONID** in cookies. It is required for requests that make changes to the cart. You will get it in the Set-Cookie header in the response.

Scenario steps:

- Search a product
- Get the product details
- Add a product to cart
- Get a client access token
- Introduce oneself as a guest
- Checkout
- Log out
- Get order information

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 Parame Content-Type
GET	https://localhost:9002/rest/v1/electronics/products	
GET	https://localhost:9002/rest/v1/electronics/products /489702	
POST	https://localhost:9002/rest/v1/electronics/cart/entry	
POST	https://localhost:9002/authorizationserver/oauth/token	client_id=$CL
POST	https://localhost:9002/rest/v1/electronics/customers/current/guestlogin	
POST	https://localhost:9002/rest/v1/electronics/customers/current/addresses	
PUT	https://localhost:9002/rest/v1/electronics/cart/address/delivery/8796158754839	
GET	https://localhost:9002/rest/v1/electronics/cart/deliverymodes	
PUT	https://localhost:9002/rest/v1/electronics/cart/deliverymodes/premium-gross	
POST	https://localhost:9002/rest/v1/electronics/cart/authorize	securityCode
POST	https://localhost:9002/rest/v1/electronics/cart/placeorder	
POST	https://localhost:9002/rest/v1/customers/current/logout	
	...	

Method	URL	Body Parame Content-Type
GET	https://localhost:9002/rest/v1/electronics/orders/byGuid/00aa0ba88c9c7c93b886423e354556dc21c430e0	
POST	https://localhost:9002/rest/v1/electronics/cart/paymentinfo	accountHo cardNumbe cardType: expiryMon expiryYea saved: "t defaultPa billingAd billingAd billingAd billingAd billingAd billingAd billingAd billingAd billingAd

Voucher Usage Scenario

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

⚠ Caution

Remember to set **JSESSIONID** in cookies. It is required for requests that make changes to the cart. You will get it in the Set-Cookie header in the response.

Scenario steps:

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

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/x-www-**
POST	https://localhost:9002/rest/v1/electronics/cart/entry	

Method	URL	Body Parameters
		Content-Type: **application/x-www-**
GET	https://localhost:9002/rest/v1/electronics/cart	
POST	https://localhost:9002/authorizationserver/oauth/token	client_id=$CLIENT_ID$&client_secret=$
POST	https://localhost:9002/rest/v1/electronics/cart/voucher/HY2008	
GET	https://localhost:9002/rest/v1/electronics/cart	
POST	https://localhost:9002/rest/v1/electronics/customers	login=john.doe%40mail.com&password &firstName=John&lastName=Doe&title
POST	https://localhost:9002/authorizationserver/oauth/token	client_id=$CLIENT_ID$&client_secret=$
POST	https://localhost:9002/rest/v1/electronics/customers/current/addresses	
PUT	https://localhost:9002/rest/v1/electronics/cart/address/delivery/8796192014359	
GET	https://localhost:9002/rest/v1/electronics/cart/deliverymodes	
PUT	https://localhost:9002/rest/v1/electronics/cart/deliverymodes/premium-gross	
GET	https://localhost:9002/rest/v1/electronics/customers/current/paymentinfos	
PUT	https://localhost:9002/rest/v1/electronics/cart/paymentinfo/8796158591018	

Method	URL	Body Parameters
		Content-Type: **application/x-www-**
POST	https://localhost:9002/rest/v1/electronics/cart/authorize	securityCode=123
POST	https://localhost:9002/rest/v1/electronics/cart/placeorder	
GET	https://localhost:9002/rest/v1/electronics/orders	
GET	https://localhost:9002/rest/v1/electronics/orders/00002036	
POST	https://localhost:9002/rest/v1/customers/current/logout	
POST	https://localhost:9002/rest/v1/electronics/cart/paymentinfo	accountHolderName: "John Doe" cardNumber: "4111111111111111" cardType: "visa" expiryMonth: "01" expiryYear: "2017" saved: "true" defaultPaymentInfo: "true" billingAddress.titleCode: "mr" billingAddress.firstName: "Joh billingAddress.lastName: "Doe" billingAddress.line1: "Toyosak billingAddress.line2: "3-16-19 billingAddress.postalCode: "53 billingAddress.town: "Osaka" billingAddress.country.isocode billingAddress.region.isocode:

Error Responses from OCC v1

Error responses returned by the `ycommercewebservices` extension.

Error Formats

Error responses used in OCC Web services have a unified format that can describe a single error or an error list.

Error Attribute	Mandatory?	Description
<type>	Yes	Type of error, for example: `AccessDeniedError`, `ValidationError`, etc.
<message>	Yes	Text message that describes the error.
<subjectType>	No	Subject type, for example: parameter, site
<subject>	No	Subject identifier, for example: *<lastName>*, *<electronics>*
<reason>	No	Error reason code, for example: invalid, missing, *<unknownIdentifier>*

Response Formats

XML Error Format

An example of an error in XML format:

```
<errors>
    <error>
        <message>...</message>
        <reason>...</reason>
        <subject>...</subject>
        <subjectType>...</subjectType>
        <type>...</type>
    </error>

    <error>
    ...
    </error>

    ...
</errors>
```

JSON Error Format

An example of an error in JSON format:

```
{
  "errors": [
    {
      "message": "...",
      "reason": "...",
      "subject": "...",
      "subjectType": "...",
      "type": "..."
    },
    {
      "message": "...",
      "reason": "...",
      "subject": "...",
      "subjectType": "...",
      "type": "..."
    }
  ]
}
```

Standard Errors

Authentication and Authorization Errors

Response Status	Case	Response Body

HTTP/1.1 401 Unauthorized	When the request requires user authentication or the user has no rights to the resource.
	Examples:
	• GET - `https://localhost:9002/rest/v1/wsTest/customers/current/addresses` without authorization token
	• GET - `https://localhost:9002/rest/v1/wsTest/customers/current/addresses` with client authorization token
		`<?xml version="1.0" en`
		`<errors>`
		`<error>`
		`<message>Acces`
		`<type>AccessDe`
		`</error>`
		`</errors>`

HTTP/1.1 401 Unauthorized	Missing client_id parameter in token request.
	Example: POST - `https://localhost:9002/authorizationserver/oauth/token`
	Header:
	`Content-Type: application/x-www-form-urlencoded`
	Request body:
	`client_secret=secret&grant_type=client_credentials`
		`<?xml version="1.0" en`
		`<errors>`
		`<error>`
		`<message>An Au`
		`<type>Unauthor`
		`</error>`
		`</errors>`

HTTP/1.1 401 Unauthorized	Wrong client_id in token request.
	Example: POST - `https://localhost:9002/authorizationserver/oauth/token`
	Header:
	`Content-Type: application/x-www-form-urlencoded`
	Request body:
	`client_id=wrong_client_id&client_secret=secret&grant_type=client_credentials`
		`<?xml version="1.0" en`
		`<errors>`
		`<error>`
		`<message>No cl`
		`<type>Unauthor`
		`</error>`
		`</errors>`

HTTP/1.1 401 Unauthorized	Wrong client_secret in token request.
	Example: POST - `https://localhost:9002/authorizationserver/oauth/token`
	Header:
	`Content-Type: application/x-www-form-urlencoded`
	Request body:
	`client_id=mobile_android&client_secret=wrong_secret&grant_type=client_credentials`
		`<?xml version="1.0" en`
		`<errors>`
		`<error>`
		`<message>Bad c`
		`<type>BadClien`
		`</error>`
		`</errors>`

HTTP/1.1 401 Unauthorized	When the authorization token is incorrect.	`<?xml version="1.0" en`
		`<errors>`
		`<error>`
		`<message>Inval`
		`<type>InvalidT`
		`</error>`
		`</errors>`

HTTP/1.1 401 Unauthorized	When the token has expired.	`<?xml version="1.0" en`
		`<errors>`
		`<error>`
		`<message>Acces`
		`<type>InvalidT`
		`</error>`
		`</errors>`

HTTP/1.1 400 Bad Request	Wrong grant type in the token request.
	Example: POST - `https://localhost:9002/authorizationserver/oauth/token`
	Header:
	`Content-Type: application/x-www-form-urlencoded`
	Request body:
	`client_id=mobile_android&client_secret=secret&grant_type=wrong_credentials`
		`<?xml version="1.0" en`
		`<errors>`
		`<error>`
		`<message>Unsup`
		`<type>Unsuppor`
		`</error>`
		`</errors>`

Response Status	Case	Response Body

HTTP/1.1 400 Bad Request	Missing grant type in the token request.
	Example: POST - `https://localhost:9002/authorizationserver/oauth/token`
	Header:
	`Content-Type: application/x-www-form-urlencoded`
	Request body:
	`client_id=mobile_android&client_secret=secret`
		`<?xml version="1.0" en`
		`<errors>`
		`<error>`
		`<message>Missi`
		`<type>InvalidR`
		`</error>`
		`</errors>`

HTTP/1.1 400 Bad Request	Missing or wrong customer credentials in the customer's token request.
	Example: POST - `https://localhost:9002/authorizationserver/oauth/token`
	Header:
	`Content-Type: application/x-www-form-urlencoded`
	Request body:
	`client_id=mobile_android&client_secret=secret&grant_type=password`
		`<?xml version="1.0" en`
		`<errors>`
		`<error>`
		`<message>Bad c`
		`<type>InvalidG`
		`</error>`
		`</errors>`

Bad Request Errors

Response Status	Case	Response Body

HTTP/1.1 400 Bad Request	When there is no required parameter in the request.
	Example: GET - `https://localhost:9002/rest/v1/wsTest/products/export/incremental` without required timestamp parameter
		`<?xml version="1.0" encoding="UTF-16"?>`
		`<errors>`
		`<error>`
		`<message>Required String parameter`
		`<type>MissingServletRequestParamet`
		`</error>`
		`</errors>`

HTTP/1.1 404 Not Found	When the resource does not exist.
	Example: GET - `https://localhost:9002/rest/v1/electronics/methodDoesNotExist`
		`<?xml version='1.0' encoding='UTF-8'?>`
		`<errors>`
		`<error>`
		`<message>There is no resource for pa`
		`<type>UnknownResourceError</type>`
		`</error>`
		`</errors>`

HTTP/1.1 405 Method Not Allowed	When the HTTP method type is not allowed.
	Example: PUT - `https://localhost:9002/rest/v1/electronics/products`
		`<?xml version="1.0" encoding="UTF-16"?>`
		`<errors>`
		`<error>`
		`<message>Request method 'PUT' not`
		`<type>HttpRequestMethodNotSupporte`
		`</error>`
		`</errors>`

hybris Specific Errors

General Errors

Response Status	Error	Case
HTTP/1.1 400 Bad Request	`ValidationError`	General case: When there is no required parameter or a parameter has an incorrect value. Example errors are provided below.

Response Status	Error	Case

		When there is no parameter or an incorrect parameter in the add address request.
		Example: POST - `https://localhost:9002/rest/v1/electronics/customers/current/addresse`
		Header:
		`Content-Type: application/x-www-form-urlencoded`
		Request body:
		`line1=Toyosaki&line2=3-16-19&town=Osaka&postalCode=531-0072&country.isocode=JP`

		When there is no parameter or an incorrect parameter in the add payment information request.
		Example: POST - `https://localhost:9002/rest/v1/electronics/cart/paymentinfo`
		Header:
		`Content-Type: application/x-www-form-urlencoded`
		Request body:
		`accountHolderName=John+Doe&cardNumber=4111111111111111&expiryMonth=01&billingAddr`
		`16-19&billingAddress.postalCode=531-0072&billingAddress.town=Osaka&billingAddress`

		When there is no parameter or an incorrect parameter in the add review request. Example: POST - `https://localhost:9002/rest/v1/electronics/products/872912/reviews`

Response Status	Error	Case
		When the storeName parameter is incorrect (for example, the store does not exist) for a pickup in store entry. Example: • POST - `https://localhost:9002/rest/v1/electronics/cart/entry?code=489702&store` • PUT - `https://localhost:9002/rest/v1/electronics/cart/entry/0/store?storeName=`

		When there is no country object with the given code in the database.
		Example: POST - `https://localhost:9002/rest/v1/electronics/customers/current/addresse`
		Header:
		`Content-Type: application/x-www-form-urlencoded`
		Request body:
		`firstName=John&lastName=Doe&titleCode=mr&line1=Toyosaki&line2=3-16-19&town=Osaka&`

		When the login parameter has an incorrect value.
		Example: POST - `https://localhost:9002/rest/v1/electronics/customers`
		Header:
		`Content-Type: application/x-www-form-urlencoded`
		Request body:
		`login=WRONG_LOGIN&password=your_password&firstName=John&lastName=Doe&titleCode=mr`

		When the promotion type parameter is incorrect. Example: GET - `https://localhost:9002/rest/v1/electronics/promotions?type=unknownType`
		When the timestamp parameter has an incorrect format. Example: GET - https://localhost:9002/rest/v1/electronics/products/expressUpdate?timestamp=2013-06-21

Response Status	Error	Case
HTTP/1.1 400 Bad Request	`InvalidResourceError`	When there is a wrong base site ID in the request. Example: GET - `https://localhost:9002/rest/v1/wrongBaseSite/titles`
HTTP/1.1 400 Bad Request	`AmbiguousIdentifierError`	When there is more than one resource with the given ID. Example: GET - `https://localhost:9002/rest/v1/electronics/products/ 12345`
HTTP/1.1 400 Bad Request	`UnknownIdentifierError`	General case: When the resource with the given ID does not exist.
		When the product with the given code does not exist. Example: GET - `https://localhost:9002/rest/v1/electronics/products/13820867876 and pr`

		When the title code is incorrect.
		Example: POST - `https://localhost:9002/rest/v1/electronics/customers/current/addresse`
		Header:
		`Content-Type: application/x-www-form-urlencoded`
		Request body:
		`firstName=John&lastName=Doe&titleCode=WRONG_ID&line1=Toyosaki&line2=3-16-19&town=`

		When the store with the given ID does not exist. Example: GET - `https://localhost:9002/rest/v1/electronics/stores/WRONG_ID`
HTTP/1.1 400 Bad Request	`ModelSavingError`	When an error occurs while saving a resource to the database.

Resource Customer Errors

Response Status	Error	Case
HTTP/1.1 400 Bad Request	`DuplicateUidError`	When a customer cannot be created because the given login already exists. Example: POST - `https://localhost:9002/rest/v1/electronics/customers` Header:

Response Status	Error	Case

		`Content-Type: application/x-www-form-urlencoded`
		Request body:
		`login=test1@test.com&password=your_password&firstName=John&lastName=Doe&titleCode=mr`
		when customer with login test1@test.com already exists

HTTP/1.1 400 Bad Request	`PasswordMismatchError`	When the given password is incorrect.
		Example: PUT - `https://localhost:9002/rest/v1/electronics/customers/current/login`
		Header:
		`Content-Type: application/x-www-form-urlencoded`
		Request body:
		`newLogin=aa@bb.com&password=your_password`
		and password is incorrect

Resource Cart Errors

Response Status	Error	Case
HTTP/1.1 400 Bad Request	`CommerceCartModificationError`	General case: This exception is thrown when adding, updating, or removing items from the cart. Error examples are
		When cart entry with the requested number doesn't exist in current cart. Example: • PUT - `https://localhost:9002/rest/v1/electronics/cart/entry/3?qty=5` • DELETE - `https://localhost:9002/rest/v1/electronics/cart/entry/3` where
		When the given quantity is incorrect Example: POST - `https://localhost:9002/rest/v1/electronics/cart/entry?code=4`
HTTP/1.1 400 Bad Request	`ValidationError`	When the cart is invalid in the place order request. It means that there is no required information for t Example: POST - `https://localhost:9002/rest/v1/electronics/cart/placeorder`

Response Status	Error	Case
HTTP/1.1 400 Bad Request	`UnsupportedDeliveryAddressError`	When the address set as the delivery address is incorrect; for example: the address does not exist, the Example: PUT - `https://localhost:9002/rest/v1/electronics/cart/address/deliv`
HTTP/1.1 400 Bad Request	`UnsupportedDeliveryModeError`	When the delivery mode does not exist or when the delivery address is not set for the cart. Example: PUT - `https://localhost:9002/rest/v1/electronics/cart/deliverymodes`
HTTP/1.1 400 Bad Request	`InvalidPaymentInfoError`	When there was a problem with setting the payment information for the cart; for example no session c Example: PUT - `https://localhost:9002/rest/v1/electronics/cart/paymentinfo/8`
HTTP/1.1 400 Bad Request	`PaymentAuthorizationError`	When there are problems with the payment authorization; for example: there is no session cart or no p Example: POST - `https://localhost:9002/rest/v1/electronics/cart/authorize?se`
HTTP/1.1 400 Bad Request	`NoCheckoutCartError`	General case: This exception is thrown when there is no session cart during a checkout process step. Before the call
		When there is no cart in session provided in the set delivery address request. Example: PUT - `https://localhost:9002/rest/v1/electronics/cart/address/deliv` and you forgot to add a JSESSIONID cookie from the previous request
		When there is no cart in session when applying or releasing a voucher. Example: POST - `https://localhost:9002/rest/v1/electronics/cart/voucher/xyz-`

		When there is no cart in session when adding payment information.
		Example: POST - `https://localhost:9002/rest/v1/electronics/cart/paymentinfo`
		Header:
		`Content-Type: application/x-www-form-urlencoded`
		Request body:
		`accountHolderName=John+Doe&cardNumber=4111111111111111&cardType=visa&exp`
		`16-19&billingAddress.postalCode=531-0072&billingAddress.town=Osaka&billi`

Response Status	Error	Case
		When there is no cart in session in a place order request. Example: POST - `https://localhost:9002/rest/v1/electronics/cart/placeorder`
HTTP/1.1 400 Bad Request	`CommerceCartRestorationError`	When the cart that you want to restore does not exist. Example: GET - `https://localhost:9002/rest/v1/electronics/cart/restore?guid=`
HTTP/1.1 400 Bad Request	`InsufficientStockError`	General case: When the product is out of stock or there is low stock for it. Error examples are provided below.
		When the product is out of stock when adding it to the cart. Example: POST `https://localhost:9002/rest/v1/electronics/ cart/entry?code=`
		When the product is out of stock in the store where it is to be picked up. Example: PUT `https://localhost:9002/rest/v1/electronics /cart/entry/0/store`
		When the product is out of stock when trying to place the order. Example: POST `https://localhost:9002/rest/v1/electronics/ cart/placeorder`
HTTP/1.1 400 Bad Request	`StockSystemError`	When the stock system is not enabled and there is no information about the stock for the product. Example: • POST `https://localhost:9002/rest/v1/electronics/cart/entry?code=489` • PUT `https://localhost:9002/rest/v1/electronics/cart/entry/store?stc`

Response Status	Error	Case
HTTP/1.1 400 Bad Request	`CommercePromotionRestrictionError`	When there is no PromotionOrderRestriction for the promotion when we try to apply the promotion to Example: POST - `https://localhost:9002/rest/v1/electronics/cart/promotion/O`
HTTP/1.1 400 Bad Request	`VoucherOperationError`	When an error occurs during a voucher operation. Error examples are provided below.
		When trying to apply a non-existent voucher. Example: POST - `https://localhost:9002/rest/v1/electronics/cart/voucher/notE`
		When an error occurs during the voucher-application process. Example: POST - `https://localhost:9002/rest/v1/electronics/cart/voucher/xyz-`
		When an error occurs during the release voucher process. Example: DELETE - `https://localhost:9002/rest/v1/electronics/cart/voucher/xy`

Resource Customer Group Errors

Response Status	Error	Case	R
HTTP/1.1 400 Bad Request	`InvalidCustomerGroupError`	When the requested group is not a sub-group of the customergroup. Example: GET - `https://localhost:9002/rest/v1/electronics/customergroups/customermanagergroup`	

Voucher Resource Errors

Response Status	Exception	Case	Response Body

HTTP/1.1 400 Bad Request	`VoucherOperationError`	When the voucher with the given code does not exist.
		Example: GET - `https://localhost:9002/rest/v1/electronics/vouchers/abc-9PSW-EDH2`
			`<?xml version="1.0" encodin`
			`<errors>`
			`<error>`
			`<message>Voucher no`
			`<type>VoucherOperat`

Response Status	Exception	Case	Response Body
			</error> </errors>

Cross-Origin Resource Sharing

Cross-Origin Resource Sharing (CORS) is a W3C effort to introduce a standard mechanism for enabling cross-domain requests in web browsers and participating servers.

Overview

The SAP Commerce software uses the open source CORS filter, which is a standard Java EE filter.

Cross-Origin Resource Sharing allows web browsers to overcome the same origin policy, which typically prohibits cross-site HTTP requests. CORS is supported by most modern web browsers, including Internet Explorer from version 10 on.

Default Configuration for ycommercewebservices Extension

The `web.xml` file for the `ycommercewebservices` extension ships with the default configuration that allows browser-based JavaScript clients to use the RESTful `ycommercewebservices` APIs.

```
<filter>
                <filter-name>CORS</filter-name>
                <filter-class>com.thetransactioncompany.cors.CORSFilter</filter-class>
                <init-param>
                    <param-name>cors.supportedMethods</param-name>
                    <param-value>GET, POST, HEAD, PUT, DELETE, OPTIONS</param-value>
                </init-param>
        </filter>

        <filter-mapping>
                <filter-name>CORS</filter-name>
                <url-pattern>/*</url-pattern>
        </filter-mapping>
```

Once HTTP Request is made by a browser from a different origin, this filter automatically adds the default CORS Headers to the response. The initiating browser needs to understand the CORS Headers in order to allow the cross-site request.

Sample Request

```
GET http://earlybird.hybris.com:9001/rest/v1/electronics/products?query=camera
Origin:
http://nfcbiergarten.appspot.com
…
```

Sample Responses

```
200 OK
Access-Control-Allow-Credentials:
true
Access-Control-Allow-Origin:
http://nfcbiergarten.appspot.com
...
```

Disabling CORS

If you do not have a browser-based JavaScript client accessing your API, disable CORS. To do so, remove the filter and filter mapping from the `web.xml` file. Additionally, remove the CORS jar file from `\ext-hybris\ycommercewebservices\web\webroot\WEB-INF\lib` directory.

Upgrading to OCC Extensions

Guidelines on upgrading from v1 to Extensions-based implementation of the OCC API.

Modified URLs

You must change URLs to point to new resources. The major change is the version number in the URL that is now v2. All resource URL paths must also be adapted to the new convention. The specific resource mapping is described in the API resources mapping below.

New Return Types

In OCC Extensions, all returned objects have been changed to WsDTO objects, so you must change all response processing to the new response structure. All WsDTO objects returned by the API methods are described in the WsDTO Concept page.

New Request Parameters

In OCC Extensions, methods have a new request parameter that specifies the form of the returned object. This request parameter is provided by default. However, to ensure that the response objects provide all the fields that you require, you can customize the request parameter.

API Resource Mapping

The following table shows you the API resource mapping.

V1		OCC Extensions
/{site}/customers	POST	/{site}/users
/{site}/customers/current	GET	/{site}/users/{userId}
/{site}/customers/current/login	PUT	/{site}/users/{userId}/login
/{site}/customers/current/profile	PUT	/{site}/users/{userId}
/{site}/customers/current/password	PUT,POST	/{site}/users/{userId}/password
/{site}/customers/current/forgottenpassword	POST	/{site}/forgottenpasswordtokens
/{site}/customers/current/addresses	GET	/{site}/users/{userId}/addresses
/{site}/customers/current/addresses	POST	/{site}/users/{userId}/addresses
/{site}/customers/current/addresses/{id}	PUT	/{site}/users/{userId}/addresses/{a
/{site}/customers/current/addresses/{id}	DELETE	/{site}/users/{userId}/addresses/{a
/{site}/customers/current/addresses/default/{id}	PUT	removed - use /{site}/users/{userI
/{site}/customers/current/paymentinfos	GET	{site}/users/{userId}/paymentdetai
/{site}/customers/current/paymentinfos/{id}	GET	/{site}/users/{userId}/paymentdeta
/{site}/customers/current/paymentinfos/{id}	DELETE	/{site}/users/{userId}/paymentdeta
/{site}/customers/current/paymentinfos/{id}	PUT	/{site}/users/{userId}/paymentdeta
/{site}/customers/current/paymentinfos/{id}/address	PUT	/{site}/users/{userId}/paymentdeta
/{site}/customers/current/customergroups	GET	/{site}/users/{userId}/customergrou
/{site}/customers/{uid}/customergroups	GET	/{site}/users/{userId}/customergrou
/{site}/customers/current/locationLatLong	PUT	removed - because there is no sess

V1		OCC Extensions
/{site}/customers/current/location	PUT	removed - because there is no sess
/{site}/customers/current/location	GET	removed - because there is no sess
/{site}/customer/current/guestlogin	POST	/{site}/users/{userId}/carts/{cart
/{site}/customer/current/convert	POST	/{site}/users
/{site}/orders	GET	/{site}/users/{userId}/orders
/{site}/orders/{code}	GET	/{site}/users/{userId}/orders/{cod
CARTS		
/{site}/cart	GET	/{site}/users/{userId}/carts/{cart
/{site}/cart/entry	POST	/{site}/users/{userId}/carts/{cart
/{site}/cart/entry/{entryNumber}	PUT,DELETE	/{site}/users/{userId}/carts/{cart
/{site}/cart/address/delivery/{id}	PUT	/{site}/users/{userId}/carts/{cart
/{site}/cart/address/delivery	DELETE	/{site}/users/{userId}/carts/{cart
/{site}/cart/deliverymodes/{code}	PUT	/{site}/users/{userId}/carts/{cart
/{site}/cart/deliverymodes	GET,DELETE	/{site}/users/{userId}/carts/{cart
/{site}/cart/placeorder	POST	/{site}/users/{userId}/orders?cart <securityCode>
/{site}/cart/paymentinfo	POST	/{site}/users/{userId}/carts/{cart
/{site}/cart/paymentinfo/{id}	PUT	/{site}/users/{userId}/carts/{cart <id>
/{site}/cart/authorize	POST	removed
/{site}/cart/restore	GET	removed
/{site}/cart/promotion/{promotionCode}	POST	/{site}/users/{userId}/carts/{cart
/{site}/cart/promotion/{promotionCode}	DELETE	/{site}/users/{userId}/carts/{cart
/{site}/cart/voucher/{voucherCode}	POST	/{site}/users/{userId}/carts/{cart
/{site}/cart/voucher/{voucherCode}	DELETE	/{site}/users/{userId}/carts/{cart
/{site}/cart/checkout	POST	
CATALOGS		
/{site}/catalogs	GET	/{site}/catalogs
/{site}/catalogs/{id}	GET	/{site}/catalogs/{catalogId}
/{site}/catalogs/{catalogId}/{catalogVersionId}	GET	/{site}/catalogs/{catalogId}/{cata
/{site}/catalogs/{catalogId}/{catalogVersionId}/categories/{category}	GET	/{site}/catalogs/{catalogId}/{cata

V1		OCC Extensions
CUSTOMERGROUPS		
/{site}/customergroups	GET,POST	/{site}/customergroups
/{site}/customergroups/{uid}/members	PUT	/{site}/customergroups/{groupId}/me
/{site}/customergroups/{uid}/members/{userId}	DELETE	/{site}/customergroups/{groupId}/me
/{site}/customergroups/{uid}	GET	/{site}/customergroups/{groupId}
ORDERS		
/{site}/orders/byGuid/{guid}	GET	/{site}/orders/{code}
/{site}/orders/statusFeed	GET	/{site}/feeds/orders/statusfeed
PROMOTIONS		
/{site}/promotions	GET	/{site}/promotions
/{site}/promotions/{code}	GET	/{site}/promotions/{code}
MISC		
/{site}/languages	GET	/{site}/languages
/{site}/currencies	GET	/{site}/currencies
/{site}/deliverycountries	GET	/{site}/deliverycountries
/{site}/titles	GET	/{site}/titles
/{site}/cardtypes	GET	/{site}/cardtypes
STORES		
/{site}/stores	GET	/{site}/stores
/{site}/stores/{name}	GET	/{site}/stores/{storeId}
VOUCHERS		
/{site}/vouchers/{code}	GET	
PRODUCTS		
/{site}/products	GET	/{site}/products/search
/{site}/products{productCode}/stock?storeName=<name>	GET	/{site}/products/{productCode}/sto
/{site}/products/export/full	GET	/{site}/products/export/full
/{site}/products/export/incremental	GET	/{site}/products/export/incrementa

V1		OCC Extensions
/{site}/products/export/references/{code}	GET	/{site}/products/{productCode}/refe
/{site}/products/expressUpdate	GET	/{site}/products/expressUpdate
/{site}/products/{code}	GET	/{site}/products/{productCode}
/{site}/products/suggest	GET	/{site}/products/suggestions
/{site}/products/{code}/reviews	POST	/{site}/products/{productCode}/revi
/{site}/products/{code}/nearLocation	GET	/{site}/products/{productCode}/stor
/{site}/products/{code}/nearLatLong	GET	/{site}/products/{productCode}/stor <latitude>

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        ;authorizedGrantTyp
                                ;client-side              ;hybris            ;basic        ;implicit,client_cr
                                ;mobile_android           ;hybris            ;basic        ;authorization_code
```

   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        ;authorizedGrantTyp
                                ;trusted_client           ;hybris            ;extended     ;authorization_code
```

   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_android`, 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.

   ⓘ 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.