# Order Web Service Sample Project

## Introduction

The purpose of this document is intended for developers that will be integrating with the CostGuard Order Placement (OP) SOAP web service. It aims to provide you with an introduction and examples of some of the more common tasks you may need to accomplish through the OP web service. This guide assumes a basic level of familiarity with CostGuard web services and the concepts behind them (SOAP vs. OData requests, how to create the security token used in requests, etc.). If you would like further background in these areas, please see the Introduction to CostGuard Web Services and Security with CostGuard Web Services guides.

### Location of Web Service Documentation

The OP web service is extensive and you are highly encouraged to review the full OP documentation to best understand the full capabilities offered as well as the details of each action available. Useful links are included below.

[CostGuard Developer Resources](https://support.idibilling.com/devtools.aspx)%20(including%20all%20web%20service%20documentation)

[Order Placement Web Service Documentation](https://support.idibilling.com/webapi/Order%20Placement%20Service%20API%20Documentation/" \l "81989.htm)

### Tasks Covered

The following examples will be provided in this document:

* Creating a cart for a customer
* Adding a new service to the cart
* Adding a feature on the service
* Adding a feature on the customer account
* Replicating a template service into multiple copies
* Checking out and submitting the cart

### Order Placement URLs

Before you can start making requests to the OP web service (or any other CostGuard web service) you need to understand how the URL for each service is structured.

The format of the URL you will use to access the OP web service is: https://<api-domain>/orderplacement/1x/<environment>

Let's break this down into each part:

* https:// - All CostGuard web services must be accessed using https.
* <api-domain> - This is the base domain of the CostGuard web service farm. Most customers will use the value api.idibilling.com although some customers run on remote farms. In this case the actual URL will be provided for you.
* /orderplacement - This is the name of the web service you are submitting requests to. All examples in this guide will use orderplacement. If you are making requests to other web services this value would change.
* /1x - This is the major version of the web service. The OP web service currently only supports /1x.
* /<environment> - This is the name of the environment your requests should go against. This value should always be lower-case.

Putting all of this together, an example URL for the TEL01\_S\_STAGE environment might look something like this: https://api.idibilling.com/orderplacement/1x/tel01\_s\_stage.

### Language / IDE

The sample code paired with this guide is written in C# using Visual Studio 2017 Enterprise. The sample project is a Console (.NET Standard) project with a service reference created to the OP service. The WSDL for the service reference is available from the OP web service documentation or via the link http://support.idibilling.com/webapi/wsdl/OrderPlacementService1.wsdl.

SOAP messages are also available as examples for other languages. These can be found in the /doc/soap directory under the project.

### Requirements

To use this guide it is assumed that you have web service credentials for the environment you are submitting requests to and are able to obtain the Simple Web Token (SWT) needed to authenticate requests to the OP service. It is also assumed your machine and Visual Studio project are configured as specified in the Getting Started with Visual Studio guide. For this project you will need to add a Service Reference and a new endpoint to your app.config / web.config file for the Order Placement service.

## Create a Cart

Most requests for anything other than a simple OData lookup will begin with the creation of a cart. Carts can be stand-alone - such as when someone walks in off the street to make a retail purchase - or associated with a customer and a service. In this example we will be associating the cart with a customer.

Here we encounter something you will see in many CG web service requests and is worth some explanation. All SOAP-based requests to the Order Placement service (and any other CG SOAP-based service) will involve two basic objects: an [Action]Request object and an [Action]Req object. The [Action]Request object is the object passed in the web service request. It in turn will always contain two child objects: a Requestor object that contains authorization / user information and the [Action]Req object. The [Action]Req object contains the information pertinent to the operation being performed. You can see this in the example project in Block 1.

CreateCartReq createCartReq = new CreateCartReq()

{

...

};

CreateCartRequest createCartRequest = new CreateCartRequest

{

Request = createCartReq,

Requestor = requestor

};

We want to populate the following information in our request to create a cart:

* CustomerID - This is the ID (not account number) of the customer we're creating the cart for
* Description - This is a free-form text field that can be used to describe the cart. There is also a Note field available for notes about the cart.
* OrderTypeID - This is the type of order the cart will represent. The OrderTypeID is used downstream to determine how the order should be processed.

Our example code with these values filled in now looks like this:

CreateCartReq createCartReq = new CreateCartReq()

{

CustomerID = new AuthenticatedID

{

AuthenticatedKey = customerID.ToString(),

ID = customerID

},

AdditionalCartInformation = new AdditionalCartInformation

{

Description = "Cart for Order Placement sample project",

OrderTypeID = newServiceOrderTypeID.ToString()

}

};

With this, we can submit a request to our Order Placement channel and create our first cart.

CreateCartResponse createCartResponse =

channel.CreateCart(createCartRequest);

Like the [Action]Request / [Action]Req breakout explained above, all SOAP-based requests will return a similar two-part set of objects in response. The first will be the [Action]Response object. This is another wrapper object that will always contain two items: a list of Messages indicating any warnings or errors that occurred during the request and an [Action]Result object that contains information specific to the action being performed. In most cases, you will want to start by looking for any errors in the messages and handle those appropriately. If there are no errors then you can process the [Action]Result object. An example of this is provided in the example project.

Now that we have created a cart and received a successful response we have a CartID which is the key to make modifications and eventually submit the cart.

## Add a New Service to the Cart

A ModifyCartRequest is very similar to a CreateCartRequest except that you need to specify a CartID on the request. For our first ModifyCart request we are going to add a service to the customer.

We want to populate the following information in our request to add a service:

* CartID - This is the ID of the cart retrieved from the response to create the cart
* CatalogID - This is the ID of the service in Product Catalog that is being added
* Description - A description of the service
* ServiceNumber - The phone number of the service (ex. 555-555-5555)
* StartDate - The date the service will become active

Our example code with these values filled in now looks like this:

ModifyCartReq modifyCartReq = new ModifyCartReq()

{

CartID = new AuthenticatedID

{

AuthenticatedKey = cartID.ToString(),

ID = cartID

}

};

modifyCartReq.AdditionalCartInformation.ServiceInformation.Add(new ServiceInformationItem()

{

CatalogID = 24392,

Description = "VOIP Line",

ServiceNumber = "55558576309",

StartDate = "2017-09-19T00:00:00.000000" // Must match the ISO 8601 format

});

We also want to specify a **response group** on both this request and subsequent requests. A **response group** is used to control what information gets returned in response to a request. The total size of items such as a cart or customer record can get fairly large and it can be inefficient to return the entire entity in response to every action so CostGuard allows developers to specify which parts of the entity we are interested in on a per-request basis. The Order Placement service supports four response groups around carts:

* AssignmentFilters
* BulkServices
* CartSummary
* OrderSummary

Details on what each response group provides can be found on the [CartResponseGroup](https://support.idibilling.com/webapi/Order%20Placement%20Service%20API%20Documentation/#61569.htm) page in your Order Placement service documentation. For our purposes, we're going to use the CartSummary response group to return information like sub-totals and the # of items in the cart.

Adding a response group is a single line of code appended on to the rest of our ModifyCart request:

modifyCartReq.ResponseGroups.Add(CartResponseGroup.CartSummary);

## Add Features to a Cart

Now that we have a service let's add a feature to it. In this case we are going to add a package that acts as the service offering. Additionally, let's add a contract at the account level to guarantee a minimum number of services are kept active. All of this is done via a second ModifyCartRequest.

For this request want to populate the following information in our request to add a feature:

* CartID - This is the ID of the cart retrieved from the response to create the cart
* CatalogID - This is the ID of the package / product in Product Catalog that is being added
* PriceOverride - This can be used to override the MRC of a feature on a per-instance basis
* Quantity - This value must be specified. For most non-retail products, it would be set to 1.
* ServiceInformationItemID - This is the ID of the new service we created in the last step and is used to link the package to the service

Our example code for adding a package to the service looks like this:

modifyCartReq = new ModifyCartReq()

{

CartID = new AuthenticatedID

{

AuthenticatedKey = cartID.ToString(),

ID = cartID

}

};

modifyCartReq.PurchaseItems.Add(new PurchaseItem

{

AdditionalPurchaseItemInformation = new AdditionalPurchaseItemInformation

{

PriceOverride = 59.99m,

ServiceInformationItemID = serviceInformationItemID // This is how we tie the product to the service

},

CatalogID = 24829,

Quantity = 1

});

We also want to add a new contract on the account level which is similar to how the package was added but not linked to a service and in our case with fewer other fields:

modifyCartReq.PurchaseItems.Add(new PurchaseItem

{

CatalogID = 24322,

Quantity = 1

});

## Add a Service and Features in the Same Request

One of the advantages of the SOAP-based web services offered by CostGuard is the power of how much can be done in a single request. For example, you can create the same service and service-level feature as created in the last two requests in a single step. The important piece to note is the TemporaryServiceInformationItemID added to the service. When you assign a value to this variable, you can then use that value to reference the service in related objects. In the code below (extra fields omitted for brevity) you can see this linkage. This idea of a *temporary ID* is used in other CostGuard SOAP web services as well.

*Note: Because the temporary ID absolutely must be unique to link properly, we recommend that it always be a negative number to ensure it doesn't match an existing ServiceInformationItemID on the request.*

You'll also notice that the service was assigned a temporary number (XXXXXXXXXX) and a ReplicateQuantity. These fields are required for the next request where we use this service as a template.

modifyCartReq.AdditionalCartInformation.ServiceInformation.Add(new ServiceInformationItem()

{

CatalogID = 24392,

ReplicateQuantity = 3,

ServiceNumber = 'XXXXXXXXXX',

TemporaryServiceInformationItemID = -1 // This is the temporary ID

});

modifyCartReq.PurchaseItems.Add(new PurchaseItem

{

AdditionalPurchaseItemInformation = new AdditionalPurchaseItemInformation

{

ServiceInformationItemID = -1 // This is how we tie the product to the service

},

CatalogID = 24829,

Quantity = 1

});

## Replicate Three Services

The OP web service supports a concept of using a ServiceInformationItem record in a cart as a template that can be used to create multiple copies of itself. The full details of capabilities are available in the OP documentation but for us, this means we can take a service with one or more features and create multiple copies of it very easily.

There are two forms of replication - bulk and single. Bulk allows you to create hundreds of services very quickly entirely server-side while Single lets you specify service-specific information like the service number or alternate service number. For our example, we will be doing a single-style replication.

The first thing needed to replicate services is a template service. A ServiceInformationItem record is marked as a template when its ReplicateQuantity is greater than 0. We set this value to 3 in our last request so we can go ahead and use that service. *Note: Once the ReplicateQuantity for a template service is decremented to 0 - which is done automatically as new services are generated from the template - the template service is removed from the cart.*

Secondly, we need to submit a new ModifyCart request specifying the details for these new services. This looks very similar to previous requests except that we will be populating items into the ModifyCartReq.AdditionalCartInformation.ReplicateServices collection.

ModifyCartReq modifyCartReq = new ModifyCartReq()

{

AdditionalCartInformation = new AdditionalCartInformation

{

ReplicateServices = new ReplicateServices()

},

CartID = new AuthenticatedID

{

AuthenticatedKey = cartID.ToString(),

ID = cartID

}

};

// Repeat this three times to replicate three services

modifyCartReq.AdditionalCartInformation.ReplicateServices.Add(new ReplicateService

{

ServiceInformationItemID = serviceInformationItemID,

ServiceNumber = (++serviceNumberBase).ToString()

});

Once we submit the modify request, magic happens and now we have three brand new, identical (other than service number) services in our cart.

## Remove a Service

Sometimes you may need to remove an item in the cart. Most items in a cart that can be removed have a Remove flag on them. You can use this to remove that item from the cart in a modify request. In our example, we would like to remove the first service we created so that we only have the replicated services and the account-level feature in our cart. This can be done as follows by specifying the ServiceInformationItemID of the service to remove along with the Remove flag:

ModifyCartReq modifyCartReq = new ModifyCartReq()

{

AdditionalCartInformation = new AdditionalCartInformation

{

ServiceInformation = new ServiceInformation()

},

CartID = new AuthenticatedID

{

AuthenticatedKey = cartID.ToString(),

ID = cartID

}

};

modifyCartReq.AdditionalCartInformation.ServiceInformation.Add(new ServiceInformationItem

{

Remove = true,

ServiceInformationItemID = serviceInformationItemID

});

## Checkout and Submit

Once you've added all items to a cart and are ready to finalize things you will need to first Checkout and then Submit the cart.

Checkout prepares the cart for submission by performing tendering and tax calculations, accepting tender in retail transactions, finding any exclusive group overlaps, and calculating promotions among other things. Only once a cart has been checked out can it can be submitted. However, if you make another change to the cart you will need to checkout again.

Submission is the final request for a cart and is what sends it off for processing to create / change the system billing records. Once a cart is submitted, further changes to its contents (services, features, etc.) are not allowed. Both requests are simple to perform with the OP web service.

To checkout our cart the request will look like:

CheckoutReq checkoutReq = new CheckoutReq

{

CartID = new AuthenticatedID

{

AuthenticatedKey = cartID.ToString(),

ID = cartID

}

};

CheckoutRequest checkoutRequest = new CheckoutRequest

{

Request = checkoutReq,

Requestor = requestor

};

CheckoutResponse checkoutResponse = channel.Checkout(checkoutRequest);

The SubmitCart request is very similar:

SubmitCartReq submitCartReq = new SubmitCartReq

{

CartID = new AuthenticatedID

{

AuthenticatedKey = cartID.ToString(),

ID = cartID

}

};

SubmitCartRequest submitCartRequest = new SubmitCartRequest

{

Request = submitCartReq,

Requestor = requestor

};

SubmitCartResponse submitCartResponse = channel.SubmitCart(submitCartRequest);

Congratulations. At this point you have created a cart, added multiple services to it manually as well as with replication, added features to both services and at the account level, removed an extra service and then checked out and submitted the cart. There are many more ways to create, modify, and submit orders with more advanced functionality that was not covered in this guide and we certainly encourage you to review the OP web service documentation as well as by playing around with the OP service in a non-production environment.