Summer Cart
SOAP API and Synchronization Guide

Overview

This guide gives a practical tutorial and a reference section with full details on how to work with the Summer Cart SOAP API and specifically on how to synchronize the data from your custom data management software with your Summer Cart-based web store. The guide lists SOAP API usage best practices and provides troubleshooting tips for the most common issues you might encounter. Finally, the How Do I section gives quick answers to common questions.

Prerequisites

Before you begin, make sure you have the following prerequisites in place:

- A working Summer Cart installation;
- An active Sync API account – see the “How Do I?” section for more information on how to create an API account;
- Any development environment that allows you to work with web services.

Contents

SOAP API Overview

The Summer Cart SOAP API allows you to search and manage the data in your Summer Cart store. The API works as a standard web service, so you can use it with any development platform that supports web services. There are many scenarios when this can be useful:

- **Access Summer Cart data for reporting purposes**. In the addition to the many built-in Summer Cart reports available from your admin area, you can use the SOAP API to access your products, orders, and other business information to produce any reports you may need.
- **Synchronize data from your business management system** with your web store. If you are using any type of business management system, you can easily synchronize your products and other information with your web store, so no need to enter anything by hand.
- **Create data dynamically**. For example, if you use your Summer Cart site only to process payments, dynamic products can be created in your system on-the-fly, and added to the user's shopping cart for checkout.

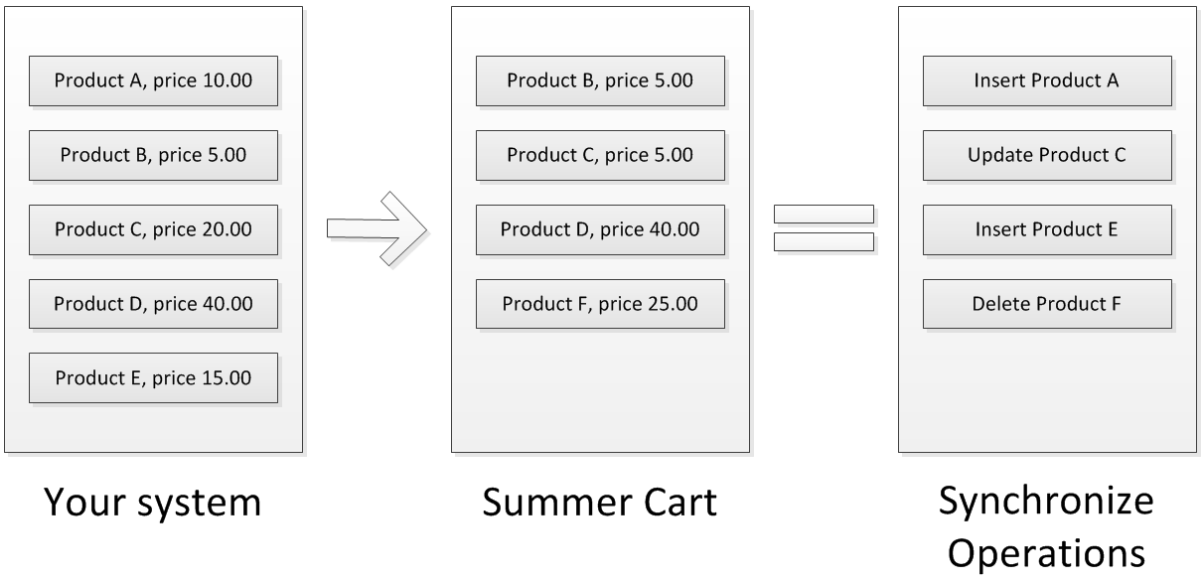
Synchronization Overview

Any system that is given a valid Summer Cart Sync API account (one you create through the Admin area, see the "How Do I" section) can manage the content of the web store through our convenient platform- and technology-independent SOAP API interface.

The first step in the synchronization process is to determine what data needs to be synchronized. Typically you would have certain items in your data management system (e.g. manufacturers, customers, products) and certain items in your Summer Cart store that were imported with a previous synchronization. (Obviously, the first time you synchronize with Summer Cart you would probably have an empty store.) For each one of your items there are 4 possible scenarios when it comes to synchronization:

- **The item was not changed**. This means that the item was imported with a previous synchronization and was not changed ever since. No action should be performed.
- **The item is missing in Summer Cart**. This means the item was not yet imported into Summer Cart, so it should be imported with the next synchronization.
- **The item is missing from your system, but present in Summer Cart**. This could either mean that the item was imported into Summer Cart with a previous synchronization, and then deleted from your system, or it was created directly in Summer Cart and does not exist in your system. Based on the type of item you may want to either delete the item from Summer Cart or insert the item into your system.
- **The item is present in Summer Cart, but is not the same as in your system**. This means the item was imported into Summer Cart with a previous synchronization, but was then modified either by your system or by Summer Cart. You may either want to update the item in Summer Cart from the copy in your system or vice versa depending on your scenario.

You can see these scenarios illustrated in the diagram on the next page. The diagram shows what operations you need to perform in order to synchronize the products from your system with the products in Summer Cart.



Fortunately Summer Cart will figure out all these operations for you! The only thing you have to do is to pass the data you have in your system, and Summer Cart will compare it with the data in its own database and tell you what operations you should perform to sync each item. However, as your system may contain a large amount of data, and each item may have many properties, it would be ineffective if you had to send to Summer Cart your entire items. Instead, you send only two properties of each item: its ID (any unique string identifier generated by your system) and a hash code (any string generated by your system that would change if any of the properties of an item change). Then what Summer Cart does is to compare these ID's and hash codes to what it already has in its database. So:

- If an ID exists in your system, but is missing from Summer Cart, then the item should be inserted in Summer Cart;
- If an ID exists in Summer Cart, but is missing from your system, the item should be deleted from Summer Cart (or, alternatively, inserted in your system);
- If an ID exists in both your system and Summer Cart, but hash code does not match, the item should be updated in Summer Cart;
- If an ID exists in both your system and Summer Cart, and hash code matches, then no action should be performed as the item is already synchronized.

Note that when an item is modified by Summer Cart, its hash code is cleared. This way you can tell if an item was modified by your system (both hash codes are not blank) or by Summer Cart (Summer Cart hash code is blank).

Once you know what operations you should do to get your store in sync, it's time for action. For each of the items that you want to insert, update or delete from Summer Cart, you should send an operation request containing the operation you want to perform (e.g. insert, update, delete) and the corresponding data item. Summer Cart will return a response for each operation letting you know if there were any problems, such as incorrect or invalid data from your system.

SOAP API Reference

Methods Overview

Two are the SOAP API methods that are most important for synchronization: **getSync** and **modify**. Call **getSync** with the ID's and hash codes of the items you want to synchronize, and it will return the operations you should perform to synchronize your items with Summer Cart. Then send a request to **modify** with these operations and they will be applied to your store.

Following is an outline of the entire process. On the next page you can find the same illustrated with a diagram.

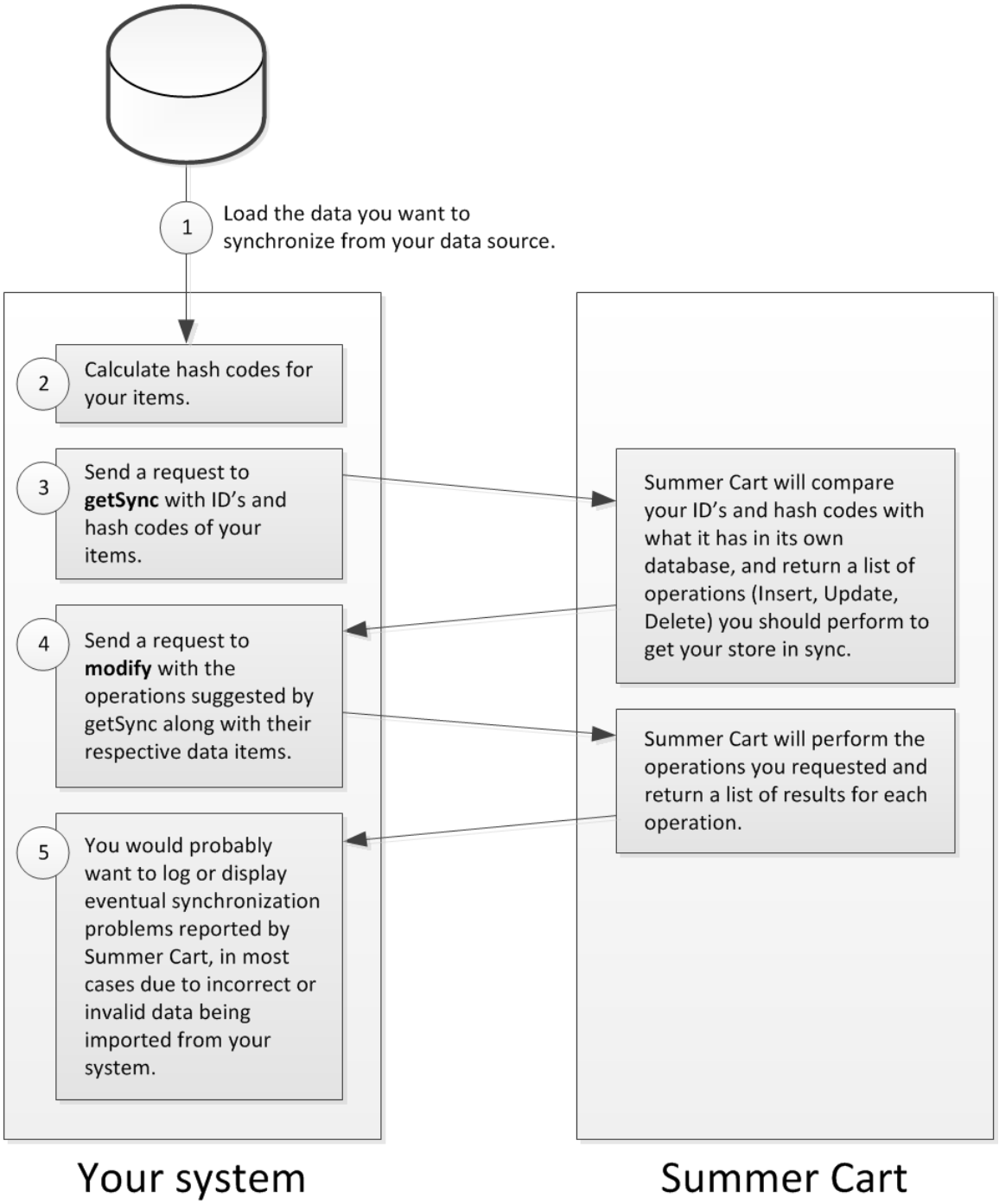
1. Read the data you want to synchronize from your data source. You may want to synchronize your data all in once, or you might want to synchronize in several steps, for example first synchronizing your manufacturers, then your product classes, then your products, and so on.

2. For each of your data items, calculate a hash code. What we suggest is that you do an MD5 of a string that is produced through concatenation of all the properties of an item, but you are free to use any other method you want. Just keep in mind that the point of using a hash code is so Summer Cart is able to easily tell if any of the properties of an item were changed (and thus the item should be updated), so your hash code calculation MUST be based on all the properties of your items.

3. Call the **getSync** method, passing ID's and hash codes of items you want to synchronize. Note that whatever items you want to synchronize (e.g. products), you should send ALL the items of this type to the service in a single method call. This is so the service knows all your items and can figure out which items are present in Summer Cart, but missing from your system (and thus should be deleted from Summer Cart). Fortunately ID's and hash codes are very small speaking of byte size, so you can send many thousands of records and not worry about message, script or memory size limits. The service will compare the ID's and hash codes of your items to what it has in its own database from previous synchronizations, and return a list of operations (Insert, Update or Delete) that should be performed with your items to get your store in sync.

4. Call the **modify** method, passing operations suggested by getSync with their respective data items from your system.

5. The service will return a list of results for each of your modify operations. You would typically want to display or log eventual problems, which in most cases will be due to incorrect or invalid data passed from your system.



The address of your Summer Cart synchronization service typically is:

http://*www.yourwebsiteaddress.com*/api/soap/v1/Service.php

(Here, replace *www.yourwebsiteaddress.com* with the actual address of your website.)

If you add "?wsdl" at the end of the service URL address, you will see the WSDL definition of the service. For example: http://*www.yourwebsiteaddress.com*/api/soap/v1/Service.php?wsdl

Authentication

Before you use any of the methods of the synchronization service, you need to authenticate your application. With every service request you should include a scSoapApiRequestHeader that has two properties:

- Username – your Sync API account username;
- Password – your Sync API account password.

The **getSync** Method

Call **getSync** with the ID's and hash codes of the items in your system. Note that you should send all the items *of the same type* with a single method call, but you are not required to send *all item types* at once. For example, you could call getSync with all your categories, and then call it again with all your products. But you can't call it with half your categories, and then again with the other half. The service should know all your items of a certain type in order to figure out if there are items that are present in Summer Cart, but missing from your system, and thus should be deleted in Summer Cart as well. The method returns a list of operations (Insert, Update, Delete) and their corresponding data items (specified by ID and hash code), so you know to send those items through the modify method.

Here is an example in pseudo code showing synchronization of categories:

```
scSoapApiRequestHeader header = CreateRequestHeader();
scSoapApiSyncRequest request = new scSoapApiSyncRequest();
scSoapApiSyncResponse response;

// Load the categories from your data source
Category[] categories = DataSource.GetCategories();

// Prepare your getSync request
foreach(Category category in categories)
{
  scSoapApiCategorySyncInfo syncInfo = new scSoapApiCategorySyncInfo();
  syncInfo.CategorySyncId = category.Id;
```

```
  syncInfo.HashCode = category.GetHashCode();
  request.CategorySyncInfo.Add(syncInfo);
}

// Set ReturnNotSynced to true to return items that were created directly
// in Summer Cart (not imported from your system and thus not synced)
request.ReturnNotSynced = false;

service.getSync(header, request, out response);

foreach (scSoapApiCategorySyncInfoResult result in
response.CategorySyncInfoResult)
{
  // In each result Summer Cart returns the Id and HashCode of your item,
  // as well as what operation should be performed with the item.
  // You should mark your items accordingly so you can later build
  // your modify() request.
}
```

Each item has two ID's – an ID sent from your system (any string that uniquely identifies the item within your system) and an ID created in Summer Cart (an integer value that uniquely identifies the item within Summer Cart). The Summer Cart ID is called "Id", as in "CategoryId", while the ID from your system is called "SyncId", as in "CategorySyncId". Note that not each item result from getSync will contain a Summer Cart ID as an item may not be imported yet and thus do not have a Summer Cart ID. Alternatively, an item may not have a SyncId if the item was created in Summer Cart directly and not imported from your system.

Sync Transactions

One limitation of the getSync method is that it requires you to send all the items of the same type in one service call. This may cause problems if you have a large number of items of the same type (e.g. 50,000 products in your store), as you will need to modify maximum script and message sizes.

Summer Cart supports one alternative to the getSync method that does the same thing, but using a transaction, so you can add a large amount of items on multiple service calls.

Here's how it works:

1. Start a transaction using the **startSyncTransaction** method. It returns a unique transaction ID.

2. Call the **addSyncInfo** method as many times as you need, passing your sync items. This method is much similar to the getSync method as to what you send (your sync items), but it does not return any results.

3. When you're done adding sync items, call the **performSyncTransaction** method. This would make Summer Cart to perform the transaction. Note that if you don't call the performSyncTransaction method over 60 minutes after your last addSyncInfo request, the transaction would be automatically deleted.

4. Finally, get the transaction results using the **getSyncTransactionResults** method. It returns the same type of results that the getSync method returns.

Here is an example in pseudo code that shows synchronization of products using transactions.

```
scSoapApiRequestHeader header = CreateRequestHeader();
int syncTransactionId;

// Load the products from your data source
Product[] products = DataSource.GetProducts();

{
  // Start transaction
  scSoapApiRequestHeader header = CreateRequestHeader();
  scSoapApiStartSyncTransactionRequest request = new scSoap…Request();
  scSoapApiStartSyncTransactionResponse response;
  service.startSyncTransaction(header, request, out response);
  syncTransactionId = response.SyncTransactionId;
}

{
  // Prepare your sync requests
  scSoapApiAddSyncInfoRequest request = new scSoapApiAddSyncInfoRequest();
  request.SyncTransactionId = syncTransactionId;

  foreach (Product product in products)
  {
    scSoapApiCategorySyncInfo syncInfo = new scSoapApiCategorySyncInfo();
    syncInfo.CategorySyncId = product.Id;
    syncInfo.HashCode = product.GetHashCode();
    request.ProductSyncInfo.Add(syncInfo);
  }

  // Split your request into multiple parts
  scSoapApiAddSyncInfoRequest[] requests = SplitRequest(request);
  scSoapApiAddSyncInfoResponse response;

  // Add the requests to the transaction
  for (int i = 0; i < requests.Length; i++)
  {
    requests[i].SyncTransactionId = syncTransactionId;
```

```
    service.addSyncInfo(header, requests[i], out response);
  }
}

int resultPages;

{
  // Perform the transaction
  scSoapApiPerformSyncTransactionRequest request = new scSoapApi…Request();
  request.SyncTransactionId = syncTransactionId;
  request.ReturnNonSynced = false;

  scSoapApiPerformSyncTransactionResponse response;
  service.performSyncTransaction(header, request, out response);

  // Get the number of operations and split them into pages of any size
  resultPages = response.OperationsCount / 1000;
}

for (int page = 1; page <= resultPages; page++)
{
  // For each page with results, get the actual results
  scSoapApiGetSyncTransactionResultsRequest request = new scSoap…Request();
  request.SyncTransactionId = syncTransactionId;
  request.OperationsPerPage = 1000;
  request.PageNumber = page;

  scSoapApiGetSyncTransactionResultsResponse response;
  service.getSyncTransactionResults(header, request, out response);

  // Apply the results to your catalog
  ApplySyncResponse(catalog, response);

  // In each result Summer Cart returns the Id and HashCode of your item,
  // as well as what operation should be performed with the item.
  // You should mark your items accordingly so you can later build
  // your modify() request.
}
```

The **modify** Method

Once you know what items you should send to Summer Cart for synchronization (they are returned by getSync you should send them through the **modify** method. It will perform the requested operations in a single transaction and return a list of results for each operation. You would typically want to display or log eventual problems, which in most cases will be due to incorrect or invalid data passed from your system.

Here is an example in pseudo code:

```
// We are using the getSync response from the previous example
scSoapApiSyncResponse getSyncResponse;
```

```
scSoapApiRequestHeader header = CreateRequestHeader();
scSoapApiModifyRequest request = new scSoapApiModifyRequest();
scSoapApiModifyResponse response;

foreach (scSoapApiCategorySyncInfoResult result in
getSyncResponse.CategorySyncInfoResult)
{
  scSoapApiCategoryOperation operation = new scSoapApiCategoryOperation();
  operation.Operation = result.Operation;
  operation.Category = FindCategoryById(result.CategorySyncId);
  request.CategoryOperation.Add(operation);
}

service.modify(header, request, out response);

foreach (scSoapApiCategoryOperationResult result in
response.CategoryOperationResult)
{
  // Log or display modify results.
  // result.CategoryId - The ID of the category in Summer Cart
  // result.CategorySyncId - The ID of the category as sent from your system
  // result.Operation - The operation that should be performed with the item
  // result.OperationStatusCode - SUCCESS or EXCEPTION
  // result.OperationStatusMessage – Exception message if OperationStatusCode
  // is EXCEPTION
}
```

The **get** Method

The **get** method allows you to select items by their Summer Cart ID. This is useful not only for synchronization, but in many other scenarios, for example in reporting. With a single method call you can select items of different types (e.g. categories, customers, products) using multiple data selectors. Each selector is a simple array of integers specifying ID's of items to return. You can also search for items based on their references to other items. For example, products have ProductClassId, which links to their product class. You can easily find all the products in a certain product class by specifying the ProductClassId in your product selector.

Here is an example in pseudo code that returns some categories and products:

```
scSoapApiRequestHeader header = CreateRequestHeader();
scSoapApiGetRequest request = new scSoapApiGetRequest();
scSoapApiGetResponse response;

request.CategorySelector = new int[] { 1, 8, 13, 44 };
request.ProductSelector = new int[] { 5, 7, 10, 12, 15 };

service.get(header, request, out response);

foreach (scSoapApiCategory category in response.Category)
```

```
{
  // Do something with returned categories
}

foreach (scSoapApiProduct product in response.Product)
{
  // Do something with returned products
}
```

Synchronizing Orders

Orders, unlike other business data types, are not imported from your system to Summer Cart, but created in Summer Cart and then, if you need to, imported into your system. The only field you can update on Summer Cart orders is the status field. For this reason orders don't have SyncInfo and are not handled by the getSync method. They are, however, updatable through the modify method, but only in regards to their status.

The synchronization process for orders is as follows:

1. Call the **getOperationsLog** method. It will return the operations log (a list of operations performed) for orders. For example, if only one order was placed in Summer Cart, which was then updated 3 times in the admin area, getOperationsLog will return 4 operations – one insert and 3 updates – referencing this order by ID.
2. Call the **get** method with all the different Order ID's returned by getOperationsLog.
3. Insert the orders that are new, update the orders that were modified in Summer Cart;
4. Call the **markOperationsLogSynced** method with the operations you have performed in your database (*mark them synced*) so Summer Cart does not send them again.

The following example in pseudo code illustrates the process:

```
scSoapApiRequestHeader header = CreateRequestHeader();

int[] orderIds;

{
    // Call the getOperationsLog method
    scSoapApiGetOperationsLogRequest request = new scSoapApi…Request();
    request.GetOrderOperations = true;
    scSoapApiOperationsLogOrder[] response;
    service.getOperationsLog(header, request, out response);

    // Read the distinct order ID's from the response
    orderIds = GetDistinctOrderIds(response);
}
```

```
scSoapApiOrder[] orders;

{
    // Call the get method with the order ID's we have from getOperationsLog
    scSoapApiGetRequest request = new scSoapApiGetRequest();
    request.OrderSelector = orderIds;
    scSoapApiGetResponse response;
    service.get(header, request, out response);

    // Read the orders from the response
    orders = response.Order;
}

// Insert/update orders in your database
scSoapApiOperationsLogSyncedOrder[] results = DataSource.SaveOrders(orders);

{
    // Finally tell Summer Cart that you have processed the operations
    scSoapApiOperationsLogSyncedOrder[] request = results;
    scSoapApiMarkOperationsLogSyncedResponse response;
    service.markOperationsLogSynced(header, request, out response);
}
```

Data Types Reference

Data Types Overview

This section explains in detail the different types of data you might want to synchronize with your Summer Cart web store. When you sync your items you would want to do it in the same order as this section lists them. For example, as products have categories, you would want to first synchronize your categories, and then your products.

There are several things that are common across different data types:

- All data types have a property of type scSoapA*Xxx*SyncInfo (e.g. manufacturers have property ManufacturerSyncInfo of type scSoapApiManufacturerSyncInfo) that holds the ID of the data item in your system, and also the HashCode created by your system.

- All data types have two separate ID's, one generated from your system and one created by Summer Cart. The Summer Cart ID is typically called *Xxx*Id (e.g. ManufacturerId), while the ID from your system is called *Xxx*SyncId and is found in the *Xxx*SyncInfo property (e.g. ManufacturerSyncInfo has a property called ManufacturerSyncId).

- All data types have a property called *Xxx*CustomData (e.g. ManufacturerCustomData), which under normal conditions is ignored by the synchronization service. This property will typically be used to hold custom data if you are using a customized synchronization service that extends the standard Summer Cart functionality.

- Some data types, such as manufacturers, categories and products, have properties of type scSoapApiMultilanguageText[]. Unlike regular strings, these properties can hold values in several different languages all at once. Each scSoapApiMultilanguageText has a value (a regular string) and a property called "lang", which specifies the language of the value, e.g. "en", "bg", etc.

- Some data types, such as manufacturers, categories and products, have properties of type scSoapApiImage. This represents an image that can be either specified by an URI (in this case Summer Cart will download and resize the image) or by the image content (raw bytes) directly. If you are referencing an image by URI, fill the scSoapApiImage.ImageUri property. If you are referencing an image by its raw content bytes, encode them in Base64 and fill the scSoapApiImage.ImageEncoded property. Do not fill both properties at the same time.

- Data types that reference other data types, such as products referencing their product classes, have properties of type scSoapApi*Xxx*Key (e.g. scSoapApiProductClassKey) that allow you to reference an item by your system's ID (in this case ProductClassSyncId) or the Summer Cart's ID (in this case ProductClassId). There will be scenarios where you only know the ID of an item in Summer Cart (such as with items that still exist in Summer Cart, but were deleted from your system), and there will be other scenarios where you only know the ID of an item in your system (such as with items not yet imported into Summer Cart). Remember that only one of the two must be set for a key, otherwise Summer Cart will throw an exception.
- Unless otherwise stated, all strings can be up to 255 characters long, integers are 32 bit unsigned, and decimals are (10,2).

Manufacturers

These are the manufacturers of your products. Each of your products may or may not have an associated manufacturer.

Class **scSoapApiManufacturer**

Property	Description
ManufacturerId Nullable Int	The ID of the manufacturer in Summer Cart
ManufacturerSyncInfo scSoapApiManufacturerSyncInfo	Contains ManufacturerSyncId (the ID of the manufacturer in your system) and HashCode (a hash code created by your system)
ManufacturerName scSoapApiMultilanguageText[]	The name of the manufacturer
ManufacturerDescription scSoapApiMultilanguageText[]	Description of the manufacturer
ManufacturerImage scSoapApiImage	Optional image of the manufacturer
ManufacturerSort Int (optional)	Specifies a sort index. In Summer Cart manufacturers will be sorted by this index in ascending order
ManufacturerCustomData String	Optional custom data

Categories

Categories can be organized into a hierarchical structure. Each product is assigned a category, and end-users may browse products by categories.

Class **scSoapApiCategory**

Property	Description
CategoryId Nullable Int	The ID of the category in Summer Cart
CategorySyncInfo scSoapApiCategorySyncInfo	Contains CategorySyncId (the ID of the Category in your system) and HashCode (a hash code created by your system)
CategoryParentKey scSoapApiCategoryKey	The ID of the parent category if this is not a root category
CategoryName scSoapApiMultilanguageText[]	The name of the category
CategoryDescription scSoapApiMultilanguageText[]	Description of the category
CategoryImage scSoapApiImage	Optional image of the category
CategorySort Nullable int	Specifies a sort index. In Summer Cart categories will be sorted by this index in ascending order
CategoryCustomData String	Optional custom data

Customers

The customers of your web store. They can either register directly from the web store, or be imported from your business system. Each customer may have one or more physical addresses. Note that the email of a customer is a unique string and a primary key in Summer Cart. If you have a customer with certain email address in your system and a different customer with the same email address already registered in Summer Cart, when you send the ID and hash code of your customer Summer Cart will ask you to insert it, however insert will then fail because the email address is already in use. So when inserting customers, search for their email addresses in Summer Cart first (use the get method with a selector by email) and if any of the customers is found, change the insert operation to an update operation.

Class **scSoapApiCustomer**

Property	Description
CustomerId Nullable Int	The ID of the customer in Summer Cart
CustomerSyncInfo scSoapApiCustomerSyncInfo	Contains CustomerSyncId (the ID of the Customer in your system) and HashCode (a hash code created by your system)
CustomerGroupId Nullable int	The ID of the customer's group in Summer Cart. Optional.
CustomerReferrerKey scSoapApiCustomerKey	The ID of the customer's referrer customer. Optional.
CustomerEmail String	The email of the customer. Must be a unique string.
CustomerPassword String	The password of the customer's Summer Cart account. Passwords are not returned by the web service. You can modify them with an update operation.
CustomerPoints Int	The points that might have been earned by the customer. Points are returned from the web service, but you can't modify them with an update operation.
CustomerStatus scSoapApiCustomerStatus	The status of the customer, ACTIVE or DISABLED.
CustomerCustomData String	Optional custom data

Customer Addresses

Represents an address of a customer. Each customer may have one or more addresses, with different identities (first name, last name, company, etc.) on each address. One of the addresses must be marked as default for billing, and one as default for shipping (these can be the same address).

Class **scSoapApiCustomerAddress**

Property	Description
CustomerAddressId Nullable Int	The ID of the customer address in Summer Cart
CustomerAddressSyncInfo scSoapApiCustomerAddressSyncInfo	Contains CustomerAddressSyncId (the ID of the customer address in your system) and HashCode (a hash code created by your system)
CustomerKey scSoapApiCustomerKey	The ID of the customer
CustomerAddressFirstName String	The first name of the customer on this address.
CustomerAddressLastName String	The last name of the customer on this address.
CustomerAddressPhone String	The phone of the customer on this address.
CustomerAddressFax String	The fax of the customer on this address.
CustomerAddressCompany String	The company of the customer on this address.
CustomerAddressLine1 String	The first line of the address.
CustomerAddressLine2 String	The second line of the address.
CustomerAddressCity String	The city of the customer address.

CustomerAddressStateCode String	The state code of the customer address.
CustomerAddressStateName String	The state name of the customer address. State names are returned by the web service, but should not be modified (set the state code instead).
CustomerAddressCountryCode String	The ISO 3166-1 Alpha2 country code of the customer address.
CustomerAddressCountryName String	The country name of the customer address. Country names are returned by the web service, but should not be modified (set the country code instead).
CustomerAddressVatNumber String	The VAT number of the customer on this address.
CustomerAddressIsDefaultBilling Boolean	Specifies whether this is the default billing address.
CustomerAddressIsDefaultShipping Boolean	Specifies whether this is the default shipping address.
CustomerAddressCustomData String	Optional custom data

Product Classes

Each product in Summer Cart must be associated with a product class. Any custom product attributes you define in your Summer Cart admin area are defined within a product class and then inherited by any product of this class. For example, as all hard drives have attributes such as Volume, Interface (SATA/PATA) and RPM, you may create a product class called Hard Drives and define these 3 attributes there. Then you can have many products of this product class, and they all will have these 3 attributes inherited from the product class.

Class **scSoapApiProductClass**

Property	Description
ProductClassId Nullable Int	The ID of the product class in Summer Cart
ProductClassSyncInfo scSoapApiProductClassSyncInfo	Contains ProductClassSyncId (the ID of the product class in your system) and HashCode (a hash code created by your system)
ProductClassName scSoapApiMultilanguageText[]	The name of the product class
ProductClassSort Nullable int	Specifies a sort index. In Summer Cart product classes will be sorted by this index in ascending order
ProductClassCustomData String	Optional custom data

Product Class Attributes

Each product class contains one or more attributes, and all products linked to that product class inherit these attributes.

Class **scSoapApiProductClassAttribute**

Property	Description
ProductClassAttributeId Nullable Int	The ID of the attribute in Summer Cart
ProductClassAttributeSyncInfo scSoapApiProductClassAttributeSyncInfo	Contains ProductClassAttributeSyncId (the ID of the attribute in your system) and HashCode (a hash code created by your system)
ProductClassKey scSoapApiProductClassKey	The ID of the parent product class
ProductClassAttributeName scSoapApiMultilanguageText[]	The name of the attribute
ProductClassAttributeType Nullable scSoapApiProductClassAttributeType	The type of the attribute. One of TEXT (regular text attribute), SELECT (attribute where the user selects one of several given options) and MULTIPLE_SELECT (attribute where the user selects one or more of several given options). Default is TEXT.
ProductClassAttributeValidator Nullable scSoapApiProductClassAttributeValidator	An optional validator for the attribute if it is of type TEXT. One of NONE (no validation) or NUMBER (text must be numeric). Default is NONE.
ProductClassAttributeUnitName String	An optional value that specifies in what unit you measure the value of the attribute. For example “m”, “l”, “kg” etc.
ProductClassAttributeSort Nullable int	Specifies a sort index. In Summer Cart attributes will be sorted by this index in ascending order
ProductClassAttributeCustomData String	Optional custom data

Product Class Attributes Values

Attributes of types SELECT and MULTIPLE_SELECT have a list of accepted values, and these values are referred to as Product Class Attributes values.

Class **scSoapApiProductClassAttributeValue**

Property	Description
ProductClassAttributeValueId Nullable Int	The ID of the attribute value in Summer Cart
ProductClassAttributeValueSyncInfo scSoapApiProductClassAttributeValueSyncInfo	Contains ProductClassAttributeValueSyncId (the ID of the attribute value in your system) and HashCode (a hash code created by your system)
ProductClassAttributeKey scSoapApiProductClassAttributeKey	The ID of the parent attribute
ProductClassAttributeValueML scSoapApiMultilanguageText[]	The attribute value display text
ProductClassAttributeValueSort Nullable int	Specifies a sort index. In Summer Cart attribute values will be sorted by this index in ascending order
ProductClassAttributeValueCustomData String	Optional custom data

Products

These are the products you sell in your web store. Each product falls into a certain category and certain product class, and has a number of built-in properties, such as code, name, price, weight and quantity. In addition, you can define various custom attributes and place them into product classes, so similar products can be put within the same product class and thus inherit the same attributes.

Class **scSoapApiProduct**

Property	Description
ProductId Nullable Int	The ID of the product in Summer Cart
ProductSyncInfo scSoapApiProductSyncInfo	Contains ProductSyncId (the ID of the attribute in your system) and HashCode (a hash code created by your system)
CategoryKey scSoapApiCategoryKey	The ID of the category this product belongs to
ManufacturerKey scSoapApiManufacturerKey	The ID of the product manufacturer
ProductClassKey scSoapApiProductClassKey	The ID of the product class this product belongs to
ProductCode String	The code of the product. Must be a unique string.
ProductCreatedTimestamp Nullable DateTime	The date and time the product was created. You can specify any value in this field, and it does not necessarily need to match the actual date and time the product was created in your system. For example, you way want to specify a different date so the products appears as “New!” in the web store
ProductName scSoapApiMultilanguageText[]	The name of the product
ProductDescription scSoapApiMultilanguageText[]	Description of the product

ProductDetailedDescription scSoapApiMultilanguageText[]	Detailed description of the product. Long text up to 65536 chars
ProductPrice Nullable Decimal	The price of the product
ProductWeight Nullable Decimal	The weight of the product
ProductQuantity Nullable Int	The quantity of the product you have in stock
ProductIsActive Nullable Boolean	Specifies whether the product is active and thus visible in your web store
ProductPageTitle scSoapApiMultilanguageText[]	Text to be used as page title (and thus browser window title) when a user opens the details of the product. If not specified, the name of the product will be used
ProductMetaKeywords scSoapApiMultilanguageText[]	Meta keywords about the product
ProductMetaDescription scSoapApiMultilanguageText[]	Meta description of the product
ProductCustomData String	Optional custom data

Product Images

Each product in your web store may have one or more images. Each image has many versions: regular image, full size image, thumbnail image, etc. These versions are generated automatically by Summer Cart – you only supply your original image.

Class **scSoapApiProductImage**

Property	Description
ProductImageId Nullable Int	The ID of the product image in Summer Cart
ProductImageSyncInfo scSoapApiProductImageSyncInfo	Contains ProductImageSyncId (the ID of the image in your system) and HashCode (a hash code created by your system)
ProductKey scSoapApiProductKey	The ID of the product
ProductImageOriginal scSoapApiImage	The original image version. This is the only version you can set, and all the other versions are generated by Summer Cart automatically from it.
ProductImage scSoapApiImage	Standard image version used for example in the product details page. Generated automatically by Summer Cart when you set ProductImageOriginal.
ProductImageLarge scSoapApiImage	Large image version used for example if you click the product image on the details page. Generated automatically by Summer Cart when you set ProductImageOriginal.
ProductImageThumb scSoapApiImage	Thumbnail image version used where thumbnails of products are displayed. Generated automatically by Summer Cart when you set ProductImageOriginal.
ProductImageBox scSoapApiImage	Box image version used on boxes and panels. Generated automatically by Summer Cart when you set ProductImageOriginal.
ProductImageIsDefault Boolean	Specifies whether this is the default (primary) image of the product.
ProductImageSort Nullable int	Specifies a sort index. In Summer Cart images will be sorted by this index in ascending order

ProductImageCustomData String	Optional custom data

Product Attributes Values

These are the actual values for the attributes a product have inherited from its product class.

Class **scSoapApiProductAttributeValue**

Property	Description
ProductAttributeValueId Nullable Int	The ID of the product attribute value in Summer Cart
ProductAttributeValueSyncInfo scSoapApiProductAttributeValueSyncInfo	Contains ProductAttributeValueSyncId (the ID of the attribute in your system) and HashCode (a hash code created by your system)
ProductKey scSoapApiProductKey	The ID of the product for which the value is defined
ProductClassAttributeKey scSoapApiProductClassAttributeKey	The ID of the attribute for which the value is defined
ProductAttributeValue scSoapApiProductAttributeValueChoice	The actual value for of the specified attribute for the specified product. scSoapApiProductAttributeValueChoice itself has 4 properties, and only one of them can be set for any given product attribute value: - **ProductAttributeValueText** is used for regular TEXT attributes; - **ProductAttributeValueML** is used for attributes where the text can be in multiple languages; - **ProductClassAttributeValueKey** is used for SELECT attributes, where an attribute has a list of accepted values (see Product Class Attributes Values) and the user must select one value; - **ProductClassAttributeMultiValueKey** is used for MULTIPLE_SELECT attributes where an attribute has a list of accepted values and the user may select multiple values.
ProductAttributeCustomData String	Optional custom data

Bundle Products

A product in Summer Cart that contains one or more sub-products is called a Bundle. These are regular products having one or more scSoapApiBundleProduct entries associated with them that specify their child products.

Class **scSoapApiBundleProduct**

Property	Description
BundleProductId Nullable Int	The ID of the bundle product in Summer Cart
BundleProductSyncInfo scSoapApiBundleProductSyncInfo	Contains BundleProductSyncId (the ID of the bundle product in your system) and HashCode (a hash code created by your system)
ProductKey scSoapApiProductKey	The ID of the parent product
SubProductKey scSoapApiProductKey	The ID of the child product
BundleProductQuantity Nullable int	The quantity of the child product contained in the parent product
BundleProductSort Nullable int	Specifies a sort index. In Summer Cart the child products of a bundle will be sorted by this index in ascending order
BundleProductCustomData String	Optional custom data

Local Option Groups

Some products may have several variations based on some attribute. For example, a customer buying shoes will typically be required to specify a shoe size, and this may or may not affect the price of the product. In this case the Shoe Size is the local option group, and individual shoe sizes are the local options associated with the group.

Class **scSoapApiLocalOptionGroup**

Property	Description
LocalOptionGroupId Nullable Int	The ID of the local option group in Summer Cart
LocalOptionGroupSyncInfo scSoapApiLocalOptionGroupSyncInfo	Contains LocalOptionGroupSyncId (the ID of the local option group in your system) and HashCode (a hash code created by your system)
ProductKey scSoapApiProductKey	The ID of the associated product
LocalOptionGroupName scSoapApiMultilanguageText[]	The name of the local option group
LocalOptionGroupType scSoapApiLocalOptionGroupType	The type of the local option group: - **PRICE_MODIFIER** for options that affect the price of the product; - **USER_IMAGE** for products where the user is asked to upload one or more image (each local option within the group specifies one image that may be uploaded); - **USER_TEXT** for products where the user is asked to type some text (each local option within the group represents a textbox); - **VARIANT** for options that do not affect the price of the product.
LocalOptionGroupSort Nullable int	Specifies a sort index. In Summer Cart the local option groups within a product will be sorted by this index in ascending order
LocalOptionGroupCustomData String	Optional custom data

Local Options

Some products may have several variations based on some attribute. For example, a customer buying shoes will typically be required to specify a shoe size, and this may or may not affect the price of the product. In this case the Shoe Size is the local option group, and individual shoe sizes are the local options associated with the group.

Class **scSoapApiLocalOption**

Property	Description
LocalOptionId Nullable Int	The ID of the local option in Summer Cart
LocalOptionSyncInfo scSoapApiLocalOptionSyncInfo	Contains LocalOptionSyncId (the ID of the local option in your system) and HashCode (a hash code created by your system)
LocalOptionGroupKey scSoapApiLocalOptionGroupKey	The ID of the parent group
LocalOptionName scSoapApiMultilanguageText[]	The name of the local option
LocalOptionPriceModType scSoapApiLocalOptionPriceModType	The modifier type of the local option: - **NONE** for options that are not in a PRICE_MODIFIER groups; - **AMOUNT** for options in PRICE_MODIFIER groups that affect the price of the product by a fixed amount; - **PERCENT** for options in PRICE_MODIFIER groups that affect the price of the product by a fixed percent;
LocalOptionPriceModValue Nullable decimal	If the modifier type is AMOUNT or PERCENT, here you specify the actual modifier value. If the modifier type is NONE, a NULL value must be used.
LocalOptionRequired Boolean	Specifies whether the user is required to enter a value for the local option. This is used only for options in USER_IMAGE or USER_TEXT groups.
LocalOptionSort Nullable int	Specifies a sort index. In Summer Cart the local options within a group will be sorted by this index in ascending order

LocalOptionCustomData String	Optional custom data

Orders

Orders placed by customers on your web store. As opposed to most other types of business data, these are synchronized from Summer Cart to your system. An order may contain one or more order lines (referencing products) and total lines (with summaries), which, again unlike most other types of business data, are passed to and from the service as part of the order itself.

Class **scSoapApiOrder**

Property	Description
OrderId Nullable Int	The ID of the order in Summer Cart
OrderStatusId Int	The ID of the order status. Summer Cart has a number of predefined order statuses that cannot be deleted, and the user is allowed to define more statuses that are specific to his web store. Typically an application needs to only handle the predefined statuses. These are: - **Unfinished (1)** The order is not yet finished - **Payment Failed (2)** The payment of the order failed - **Payment Pending (3)** The order was placed and processed by the operator, but not yet paid - **New (4)** The order was placed by the customer, but not yet processed - **In Progress (5)** The order is being processed by the operator - **Cancelled (6)** The order was cancelled - **On Hold (7)** The order is on hold - **Delivered (8)** The order was delivered - **Returned (9)** The order was returned by the customer - **Queued (10)** The order was queued for later processing

OrderTimestamp DateTime	The date and time the order was placed
OrderPaymentStatus scSoapApiOrderPaymentStatus	The payment status of the order: - **PAYMENT_STATUS_NONE** No payment has been made - **PAYMENT_STATUS_AT_PROCESSOR** We are waiting for the payment processor - **PAYMENT_STATUS_DECLINED** Payment was declined - **PAYMENT_STATUS_PENDING** Received payment pending from processor - **PAYMENT_STATUS_AUTH** Received authorization confirmation from processor - **PAYMENT_STATUS_CAPTURE** Received capture confirmation from processor - **PAYMENT_STATUS_PARTIAL_REFUND** Payment partially refunded - **PAYMENT_STATUS_REFUND** Payment refunded - **PAYMENT_STATUS_CHARGEBACK** Payment charged back - **PAYMENT_STATUS_OTHER** Unknown payment status received from processor - **PAYMENT_STATUS_OFFLINE** The payment will happen offline - **PAYMENT_STATUS_PARTIAL_CAPTURE** Some of the amount has been captured - **PAYMENT_STATUS_VOID** Payment voided
OrderCustomerInstructions String	Instructions given by the customer
OrderCustomerEmail String	The email of the customer
OrderShippingTrackingNo String	Shipping tracking number, if one is available
OrderShippingModuleName String	The name of the shipping module

OrderShippingMethodName String	The name of the shipping method
OrderPaymentModuleName String	The name of the payment module
OrderPaymentModuleIntegration String	The name of the payment module integration
OrderPaymentTransactionNo String	Payment transaction number
OrderCheckoutModuleName String	The name of the checkout module
OrderCouponCode String	Coupon code if one is used
OrderTotal Decimal	The total amount of the order
OrderCurrencyCode String	The code of the currency used
OrderCustomerCurrencyTotal Decimal	The total amount of the order in the currency used by the customer
OrderCustomerCurrencyCode String	The code of the currency used by the customer
OrderTotalPayAmount Decimal	Total amount to be paid for the order
OrderTotalPayCurrencyCode String	The code of the total pay amount currency
OrderBillingAddress scSoapApiOrderAddress	The billing address of the customer
OrderShippingAddress scSoapApiOrderAddress	The shipping address of the customer
OrderItems scSoapApiOrderItem[]	The items of the order

OrderTotalLines scSoapApiOrderTotalLine[]	The total lines of the order
OrderCustomData String	Optional custom data

Order Lines

Lines of orders that represent an ordered product with its quantity.

Class **scSoapApiOrderItem**

Property	Description
OrderItemId Nullable Int	The ID of the order item in Summer Cart
OrderItemPrice Decimal	The unit price of the product in the store currency
OrderItemCustomerCurrencyPrice Decimal	The unit price of the product in the currency selected by the customer
OrderItemQty Int	The quantity of the ordered products
OrderItemTotal Decimal	The total price of the ordered products in the store currency
OrderItemCustomerCurrencyTotal Decimal	The total price of the ordered products in the currency selected by the customer
OrderItemDiscount Decimal	The discount per unit for this order line in the store currency
OrderItemCustomerCurrencyDiscount Decimal	The discount per unit for this order line in the currency selected by the customer
OrderItemProductCode String	The code of the product
OrderItemProductName String	The name of the product
OrderItemCategoryName String	The name of the product category
OrderItemBundledItems scSoapApiOrderItem[]	If the ordered product is a bundle, its associated order item will contain sub-items linking to the sub-products of the bundle
OrderItemOptions scSoapApiOrderItemOption[]	The global options selected by the customer

OrderItemLocalOptions scSoapApiOrderItemLocalOption[]	The local options specified by the customers
OrderItemCustomData String	Optional custom data

Order Total Lines

Lines of text displayed at the end of an order that would typically contain total amounts and other summaries.

Class **scSoapApiOrderTotalLine**

Property	Description
OrderTotalLineId Nullable Int	The ID of the order total line in Summer Cart
OrderTotalLineType Int	The type of total line
OrderTotalLineName String	The text displayed on the total line
OrderTotalLineAmount Decimal	The amount of the total line
OrderTotalLineCustomerCurrencyAmount Decimal	The amount of the total line in the currency selected by the customer
OrderTotalLineSort Int	Sort index
OrderTotalCustomData String	Optional custom data

Best Practices

Please consider the following best practices when your build your Summer Cart synchronization application:

- Validate your data before sending it to the service. At database level, use unique primary keys and foreign keys to assure referential integrity. Failing to provide the service with valid primary keys and foreign keys will result in sometimes obscure error messages. It will be much easier to prevent incorrect data to reach the service than analyzing verbose log files to figure out what is wrong.

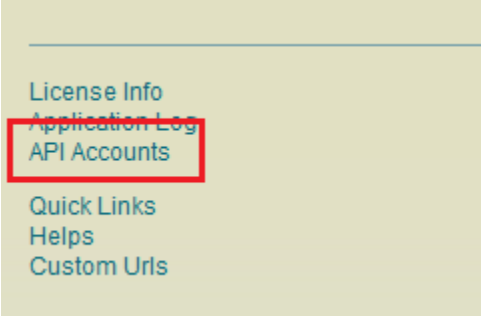
- Use sync transactions if you have more than few thousand items to synchronize with your system. This will result in much lower server load and minimal impact on server performance during synchronization. Alternatively, you can increase the PHP script memory and message size limits on your server.

- Split your modify request in chunks if you are modifying more than few thousand items. The effect is the same as when using sync transactions – lower server load and better response time during synchronization. The recommended chunk size is around 1000 items. If the chunk size is too big you may get error messages stating the max script or message size quota was reached and your service operation will be aborted.

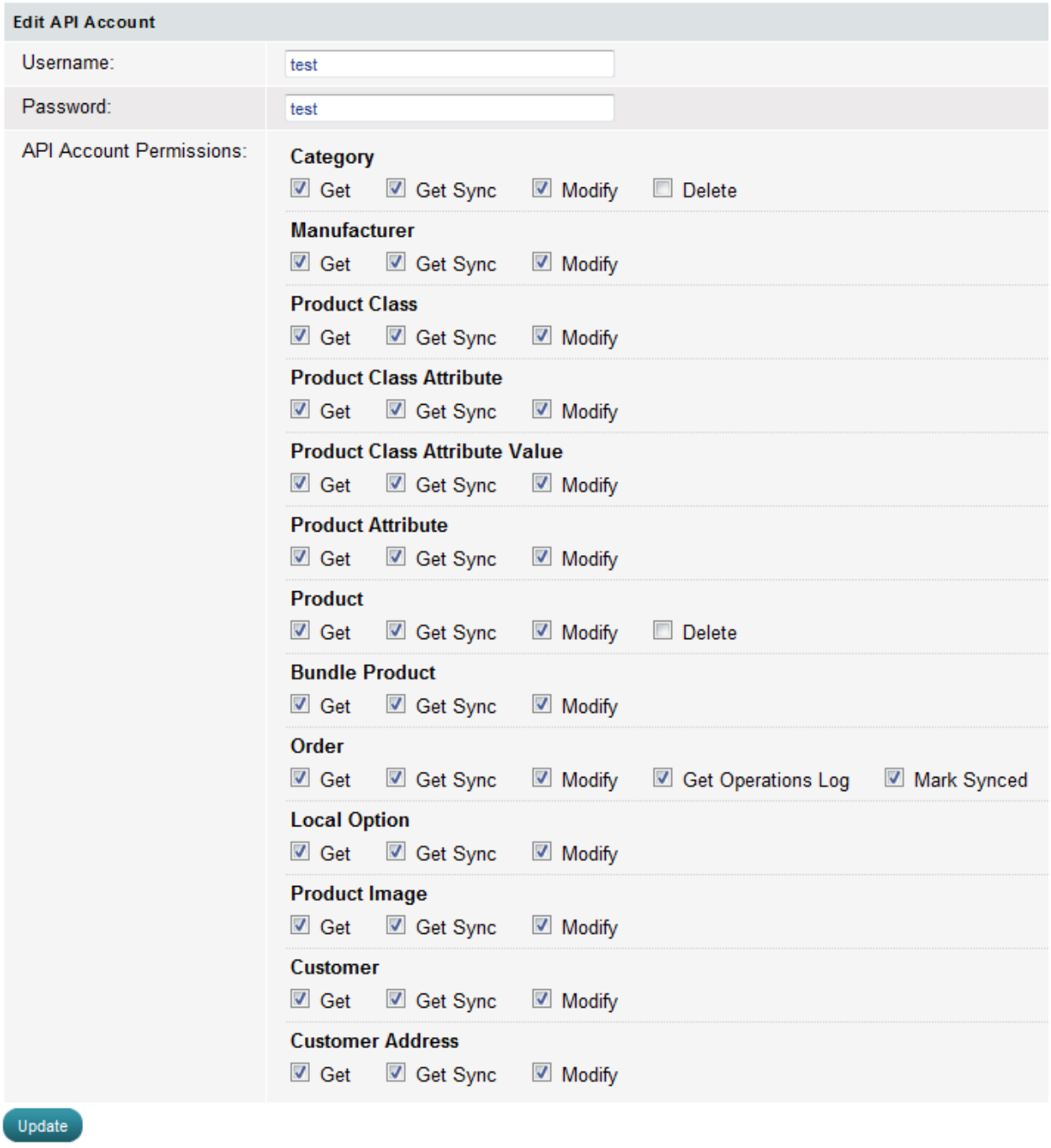
How Do I…

Create an API account in Summer Cart?

1. Open your Summer Cart Admin panel.
2. Navigate to the bottom of the page and click API Accounts



3. Add a new API account (*see next page for screenshot*). For each data type that can be synchronized, you can specify if the API account has access to the get(), getSync() and modify() methods of the service in regards to this data type.


Edit API Account
Username:
test
Password:
test
API Account Permissions:
Category
Get
Get Sync
Modify
Delete
Manufacturer
Get
Get Sync
Modify
Product Class
Get
Get Sync
Modify
Product Class Attribute
Get
Get Sync
Modify
Product Class Attribute Value
Get
Get Sync
Modify
Product Attribute
Get
Get Sync
Modify
Product
Get
Get Sync
Modify
Delete
Bundle Product
Get
Get Sync
Modify
Order
Get
Get Sync
Modify
Get Operations Log
Mark Synced
Local Option
Get
Get Sync
Modify
Product Image
Get
Get Sync
Modify
Customer
Get
Get Sync
Modify
Customer Address
Get
Get Sync
Modify
Update