salesorder.com

Cart Connect API

Version 1.00

Cart Connect API

Integrating your Shopping Cart with salesorder.com

---

by Salesorder Ltd

This document gives details on how to integrate Shopping Carts with salesorder.com using the Cart Connect API.

It is intended for developers familiar with PHP, XML and salesorder.com

Cart Connect API

Printed: January 2011

Table of Contents

1 Introduction

This guide is intended for software developers wishing to integrate Shopping Carts with the salesorder.com small business ERP system. Integration is achieved via a freely available PHP API, *Cart Connect*, which can be downloaded from **here**. The API consists of a set of PHP function stubs that, when implemented, will facilitate the synchronization of data between salesorder.com and the Cart. Typical functionality available via the API includes :-

FROM CART TO SO

- Automatic importing of Orders
- Automatic updating of Customer details
- Automatic updating of Item details
- Selective importing of Items
- Selective importing of Orders
- Selective importing of Customers

FROM SO TO CART

- Real time stock level updates from salesorder.com to the Cart
- Automatic update of Customer details
- Automatic update of Item details
- Selective export of Items
- Selective export of Customers

Skills Required
You will need to be competent in PHP and XML, and of course have a thorough knowledge of the Cart you intent to integrate with. You will also need to be familiar with salesorder.com.

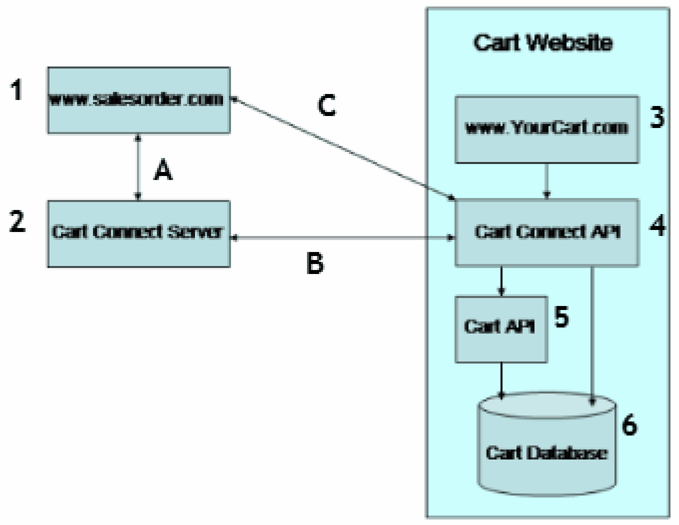
You will need to be familiar with the Shopping Cart functionality within salesorder.com. More information can be found in the salesorder.com help file **here**.

Testing your implementation
Your implementation can be tested using a standard version of salesorder.com provided the *Shopping Cart* Plugin has been enabled. Contact salesorder.com Customer Services for more information.

Reference implementation
You can download a reference implementation for the PrestaShop Cart from **here**.

2 Architecture

The Cart Connect architecture is straightforward, the diagram below shows the various components.



The components are numbered 1-6, the various communication paths are lettered A,B and C. We explain each below.

1 This is the salesorder.com website which corresponds to a particular instance of the salesorder. com web application.

2 This is the *Cart Connect* server which polls the Shopping Cart for new orders, updated customer details, updated item details etc.

3 This is the Web Server of the Cart you wish to integrate.

4 This is the Cart Connect API code

5 The is the Cart's API (if it has one). The Cart Connect API may use either the Carts API or connect directly to the Carts Database (or indeed both), depending what is most convenient.

6 This is the Cart's database.

A/B The Cart Connect Server polls the Shopping Cart at regular (typically 10 minute) intervals. When it finds new orders it pulls them from the Cart and sends them to salesorder.com (using XML). Similarly it polls the Cart for Customer/Item updates.

C Cart Connect allows you to selectively import and export Customers,Orders and Items to and from your Cart. This link does not use the Cart Connect Server.

3 Steps to Integration

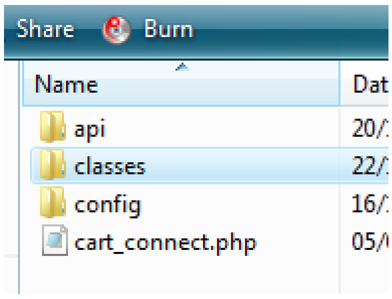
This section shows the sequence of steps needed to integrate your Cart with *Cart Connect.*

Step 1
Download the latest version (zip file) of the the Cart Connect API from **here**.

Step 2
Download an example implementation (for PrestaShop) from **here**. This will show you in detail how the integration is achieved.

Step 3
Unzip and install the Cart Connect API directory in a suitable place on your Cart's Web Server. For example, the root directory *cartconnect* could be placed under the actual Cart's root directory. The file *cart_connect.php* must be publicly accessible from the web server.

The directory structure of *cartconnect* is shown below :-



The directories are as follows.

api
The PHP api function that **you** must complete

classes
The PHP classes used by Cart Connect. **Do not modify these files**.

config
Configuration for both the Cart's database access and Cart Connect's password etc.

cart_connect.php
The entry point for all calls to Cart Connect. **Do not modify this file**.

Step 4
In your salesorder.com system create a Custom Shopping Cart (from the Explorer go to *Setup->Plugins->Shopping Carts* and from the Action drop-down click *New Custom Cart*). This Shopping Cart can be used to test your new integration. For information on how to use the Shopping Cart within salesorder.com see the help information **here**.

To complete the integration you must implement the function stubs included in the *api* directory. You must also include any configuration information, for the database etc, in the *config* directory.

By far the easiest way to do this is to look at the existing *PrestaShop* example (downloaded in Step 2), in conjunction with the Function Stubs information.

Step 5
Once you have implemented both the PHP function stubs and configuration file you should be able to connect and use Cart Connect with a suitably enabled salesorder.com instance.

4 Function Stubs

This section lists the various function stubs that you need to implement.

```
//
//   Description:
//   Performs any tests to make sure the Cart Connect installation is
//   OK, including checking the Cart Connect Password. This function also
//   returns a list of the Currencies used in the Cart. If there is a mismatch
//   salesorder.com will inform the user which currencies need to be added.
//
//   Throws an SOException if the test fails for any reason.
//
//   Inputs:
//   Cart Connect password.
//
//   Output:
//   XML of type SOCartTest
//
function   TestCart($cartConnectPassword)
{}

//
//   Description:
//   Returns a list of Orders with the specified unique
//   Order Numbers. The Order Number is usually the auto
//   incremented primary key on the Cart's Order table.
//
//   Inputs:
//    $xmlOrderNumbers
//   List of values with the values being the unique order ids.
//   List is of the form <List><V v="id1"/><V v="id2">....</List>
//
//   Output:
//    List of SOTransaction in the form <Orders>...SOTransaction...SOTranaction..<Orders>
//
function   GetOrdersIn($xmlOrderNumbers)
{}

//
//   Description:
//   Returns a list of Orders from, and including the specified
//   date/time
//
//   Inputs:
//    $fromDateTime
//   Date-time from which Orders are retrieved. The Date-time is in the
//   format YYYY/MM/DD HH:MM:SS
```

```
//
//  Output:
//  List of SOTransaction of the form
//    <Orders dateTime='YYYY/MM/DD HH:MM:SS'>...SOTransaction...SOTransaction...</Orders>
//  Note the from date-time value is included as an attribute in the
//  format YYYY/MM/DD HH:MM:SS.
//
function   GetOrdersFrom($fromDateTime)
{}

//
//  Description:
//  Return a list of Order summaries within the specified time period.
//
//  Inputs:
//  $fromDate
//  Date-time from which Orders are retrieved. The date is in the
//  format YYYY/MM/DD
//  $toDate
//  Date-time from which Orders are retrieved. The date is in the
//  format YYYY/MM/DD
//
//  Output:
//  List of SOTransactions
//
function   GetOrdersSummary($fromDate,$toDate)
{}

//
//  Description:
//  Returns a list of Customers with the specified unique
//  Customer Numbers. The Customer Number is usually the auto
//  incremented primary key on the Cart's Customer table.
//
//  Inputs:
//  List of values with the values being the unique Customer ids.
//  List is of the form <List><V v="id1"/><V v="id2">....</List>
//
//  Output:
//  List of SOCustomer
//
function   GetCustomersIn($xmlCustomerNumbers)
{}

//
//  Description:
//  Return a list of Customer given the first name,
//  last name and email address.
//
//  Inputs:
//  $firstName
//  Customer first name (with possible wild card)
//  $lastName
//  Customer last name (with possible wild card)
//
//  Output:
//  List of SOCustomer
//
function   GetCustomers($firstName,$lastName)
{}
```

```
//
//   Description:
//   Get a list of the Customers updated since the specified
//   from date/time.
//
//   Inputs:
//   Date-time from which Customers have been updated. The Date-time is in the
//   format YYYY/MM/DD HH:MM:SS.
//
//   NOTE: The update time must be greater than $fromDateTime.
//
//   Output:
//   List of SOCustomer
//
function   GetUpdatedCustomers($fromDateTime)
{}

//
//   Description:
//   Get a list of the Items updated since the specified
//   from date/time.
//
//   Inputs:
//   Date-time from which Items have been updated. The Date-time is in the
//   format YYYY/MM/DD HH:MM:SS.
//
//   NOTE: The update time must be greater than $fromDateTime.
//
//   Output:
//   List of SOItem
//
function   GetUpdatedItems($fromDateTime)
{}

//
//   Description:
//   Return a list of the Items not currently imported. These are
//   Items that have not yet been flagged by the SetItemReferences()
//   function.
//
//   Inputs:
//   $itemCode
//   Item code (with possible wild card)
//    $itemDescription
//   Item description (with possible wild card)
//
//   Output:
//   XML consisting of a list of SOAttributes giving details of the defined Attribute/values to
//   a list of SOItems.
//
//     XML of the form <ItemsAndAttributes><Attributes>,,SOAttribute..</Attributes><Items>..SOIt
//
function    GetItemsNotImported($itemCode,$itemDescription)
{}

//
//   Description:
//   Return a list of items given the specified item code,
//   and/or description. Both these values may have wildcards.
```

```
//
//   Inputs:
//   $itemCode
//   Item code (with possible wild card)
//    $itemDescription
//   Item description (with possible wild card)
//
//   Output:
//   XML consisting of a list of SOAttributes giving details of the defined Attribute/values to
//   a list of SOItems.
//
//     XML  of  the  form  <ItemsAndAttributes><Attributes>,,SOAttribute..</Attributes><Items>..SOIt
//
function    GetItems($itemCode,$itemDescription)
{}

//
//   Description:
//   Tag the items in the Cart with the associated Item Code
//   in salesorder.com (if this is possible).This facilitates
//   keeping check on which Items have been imported.
//   Note that with some Carts, Matrix combinations do not
//   have there own unique product code/name. In this case
//   the matrix identifiers are sent through as a separate
//   list.
//
//   Inputs:
//   $itemIds
//   List of values with the values being the unique Item codes.
//   List is of the form <List><V v="code1"/><V v="code2">....</List>
//   $matrixIds
//   List of values with the values being the unique Matrix Item codes.
//   List is of the form <List><V v="code1"/><V v="code2">....</List>
//
//   Output:
//   SOStatus::$OK
//
function    SetItemReferences($itemIds,$matrixIds)
{}

//
//   Description:
//   Update the Stock Quantites of specified Items.
//
//   Inputs:
//    $stockQuantities
//   List of stock quantity values. This is of the form,
//
//    <List><Stock  itemCode="xxx"  qty="55.57"  cartId="zz"  cartMatrixId="mm"/>...</List>
//
//   itemCode      - the salesorder.com item code
//   qty           - the quantity as a floating point number
//   cartId        - the Carts unique item id
//   cartMatrixId - the Carts unique matrix id (optional)
//
//   Output:
//   SOK
//
function    UpdateStockQuantities($stockQuantities)
{}
```

```
//
//  Description:
//  Update Items with the specified list.
//
//  Inputs:
//  $items
//  List of SOItem
//
//  Output:
//   SOStatus::$OK
//
function  UpdateItems($items)
{}

//
//  Description:
//  Update Customers with the specified list.
//
//  Inputs:
//  $customers
//  List of SOCustomer
//
//  Output:
//   SOStatus::$OK
//
function  UpdateCustomers($customers)
{}

//
//  Description:
//  Create the specified Items.
//
//  Inputs:
//  $items
//  List of SOItem.
//
//  Output:
//  List of item mappings of the form
//   <List><Item id="xxx" cartId="yyy" cartMatrixId="zzz"/>...</List>
//
//  id           - the salesorder.com Item id
//  cartId       - the associated Cart Item id
//  cartMatrixId - the associated Cart Matrix id
//
function  CreateItems($items)
{}

//
//  Description:
//  Create the specified Customers.
//
//  Inputs:
//  $customers
//  List of SOCustomer.
//
//  Output:
//  List of item mappings of the form
//   <List><Item id="xxx" cartId="yyy"/>...</List>
//
```

```
//   id       - the salesorder.com Customer id
//   cartId   - the associated Cart Customer id
//
function   CreateCustomers($customers)
{}
```

5 Classes

The following classes are used by the API and should not be modified.

CartConnectDB
Handles connections to the Cart database.

SOCartTest
Returns the status,version and currencies for the the current Cart Connect installation.

SOContact
Data representing a Contact.

SOCustomer
Data representing a Customer.

SOItem
Data representing an Item/Product.

SOAttribute
Data representing an Item/Product attribute and value.

SOTransaction
Data representing a Cart Order.

SOLineItem
Data representing a Cart Order line item.

SOStatus
General status class.

SOException
General exception class.

6 Downloads

The following files are available for download.

The Cart Connect API code (zip file)
http://www.salesorder.com/salesorder/downloads/cartconnect/api/cartconnect_api_1.00.zip

Example implementation for PrestaShop Cart (zip file)
http://www.salesorder.com/salesorder/downloads/cartconnect/prestashop/cartconnect.zip

This document
http://www.salesorder.com/salesorder/downloads/cartconnect/api/cartconnect.pdf

Help information
http://www.salesorder.com/salesorder/sohelp/help/Working_with_your_cart.htm