ChannelUnity Connector Kit API Documentation

Overview

This document describes the functionality of the **ChannelUnity Connector Kit API**. This API allows an online store (for example Magento) to be connected to a ChannelUnity account. When connecting using this API, you don't necessarily need an online store hosted on a web server. You can use any or all of the following supported functionality:

Communication into the ChannelUnity account:

- Validate a merchant account
- Get subscriptions and user names currently present on a merchant account
- Upload product data including prices and quantities
- Upload details of stores and product categories

Callbacks to the API:

- Orders from customers placed on the connected marketplaces
- Request from ChannelUnity for a list of all SKUs
- Request from ChannelUnity to provide product data

The API works via an endpoint through which XML messages are posted. The XML messages are uploaded via HTTPS. For PHP or C# example code please visit the links below.

Sample code

CUSDK - C# Visual Studio Sample
This is a Visual Studio solution written in C# which demonstrates how to call the key ChannelUnity APIs, as well as a listener to accept notifications. There are three projects within the solution. The code which calls into the ChannelUnity API, a test console app which makes calls to a ChannelUnity account, and a listener project to accept order notifications and so on.

https://dl.dropboxusercontent.com/u/29948910/Projects/CUSDK.zip

SOAP / WSDL Web service
A WSDL service description is provided at the following link which can be used to access the same functionality as the XML over HTTPS API.

https://api.channelunity.com/?wsdl

For further information including samples in PHP, visit our support portal at http://support.channelunity.com.

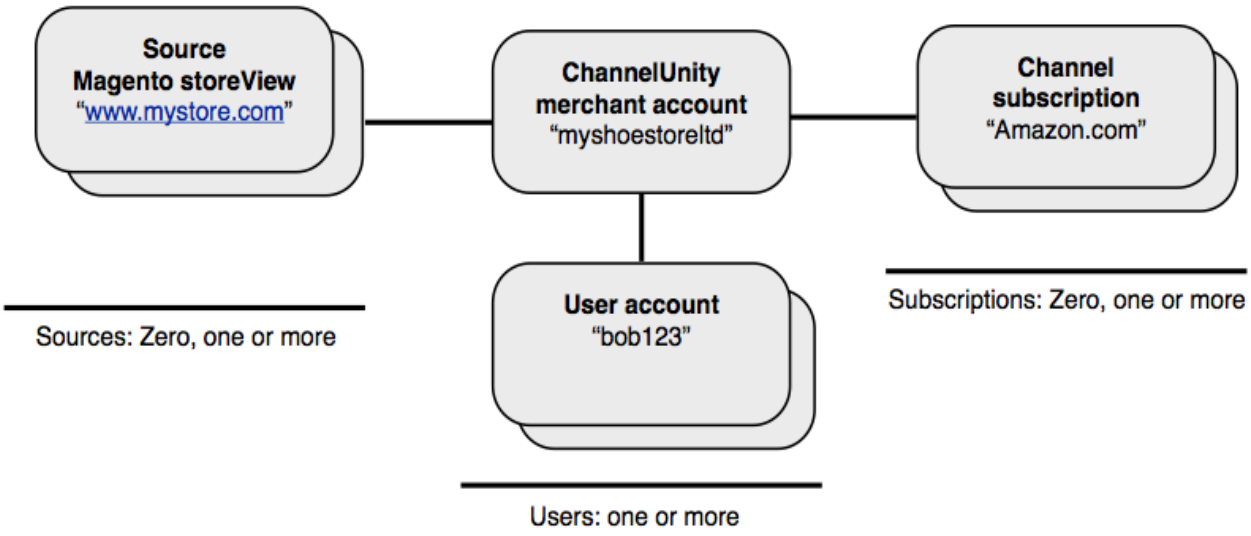
Object Model

At the top level of the API is a “merchant”. Each merchant has a unique API key which is required when accessing the API.

Each merchant has one or more user accounts. A “main user” is created when a merchant account is created, and further users can be added with full access or restricted roles. Users are identified by the merchant name and user name in combination (i.e. a merchant could have user names also in use by other merchants).

Merchants have a list of zero or more subscriptions associated with them. Each subscription is a link to a shopping channel such as Amazon.com. Each subscription has an ID and an SKU (to uniquely indicate the exact channel in use).

Each merchant also has a set of “sources”. Each source is a shopping cart which can be the main retail website running Magento for example. A source provides product data and category data to ChannelUnity which is used to synchronize with the shopping channels. Each source has a Base URL, and in the case of Magento sources, a store view ID to link to a specific store. The diagram below illustrates this model.



Notes:

- Access to the API itself is inclusive within the main feature set of ChannelUnity.
- It is possible to link to multiple instances of the same channel, for example, multiple Amazon.com accounts.

API Usage

A typical request sent to the ChannelUnity Connector Kit API works as follows.

1. Create an XML message according to the specific API being called. All requests require the XML elements in bold.

```
<ChannelUnity>
    <MerchantName>mytoolsltd</MerchantName>
    <Authorization>12341234</Authorization>
    <ApiKey>ac69d65ca5d466</ApiKey>
    <RequestType>ProductData</RequestType>
    <Payload></Payload>
</ChannelUnity>
```

2. Create a URL encoded version of this XML message and prepend "**message=**". The complete text string will constitute the data for a form upload.

3. Create a POST request to the URL "https://my.channelunity.com/event.php". Set the Content-Type as application/x-www-form-urlencoded. Upload the data generated in step 2.

4. The response is an XML document, the contents of which will depend on the API which was called. However most calls have a message status, for example:

```
<ChannelUnity>
    <Status>OK</Status>
</ChannelUnity>
```

The <Status> element in the returned XML message indicates whether the operation was a success or an error. To detect error conditions, the status will begin with the string "Error". Specific error messages you may encounter are shown for each API call.

Request Elements

In the XML request, the **MerchantName** element contains the merchant name which is associated with the ChannelUnity account. The **ApiKey** element is available from the ValidateUser API, and once obtained it can be stored as it will never change. The **RequestType** element identifies which API is being called. The **Payload** element contains data specific to the API call in use.

This leaves the **Authorization** element. This allows ChannelUnity to validate your username and

password and grant API access.
To generate the value to be used in the Authorization element, follow these steps:

1. Treating all strings as UTF-8 strings, create a SHA-256 hash of the password.
2. Convert this hash into a string by encoding the bytes as a hexadecimal string. For example, the byte value 255 would be encoded as the string “ff” (please use lower case hexadecimal).
3. Now concatenate the user name, the colon symbol “:” and the string from step 2.
4. Create a Base64 encoded string of the result of step 3.

API Calls

This section contains all the available API calls in the ChannelUnity Connector Kit API.

ValidateUser

This call returns your API Key and a list of users and subscriptions for the merchant account.

Input

```
<ChannelUnity>
    <MerchantName>mymerchant</MerchantName>
    <Authorization>ZWdp......=</Authorization>
    <RequestType>ValidateUser</RequestType>
</ChannelUnity>
```

Output

```
<ChannelUnity>
    <Status>OK</Status>
    <MerchantName>mymerchant</MerchantName>
    <ApiKey>361426524563637373</ApiKey>
    <DatabaseLocation>channelunity.com</DatabaseLocation>
    <CreatedDate>2012-08-06 15:40:03</CreatedDate>
    <VersionId>0.7.0.1</VersionId>
    <SubscriptionId>8503</SubscriptionId>
    <Users>
        <User>
            <UserName>egis01</UserName>
            <Name><![CDATA[Egidijus Egidijus]]></Name>
            <Company><![CDATA[egis01]]></Company>
            <MobileNumber>0</MobileNumber>
            <EmailAddress>3526266@camiloo.co.uk</EmailAddress>
            <Country></Country>
        </User>
[  <User> element repeated for each user  ]

    </Users>
    <Subscriptions>
        <Subscription>
            <DateAdded>2011-10-22 09:32:08</DateAdded>
            <ServiceSku>CU_AMZ_UK</ServiceSku>
```

```
            <SettingsLinkId>107</SettingsLinkId>
            <TrialExpiryWarned></TrialExpiryWarned>
            <ServiceStatus>Paid until 24 Dec 2013</ServiceStatus>
            <ServiceStatusInd>2</ServiceStatusInd>
            <PaidTo>2013-12-24</PaidTo>
        </Subscription>
```
[<Subscription> element repeated for each subscription]

```
    </Subscriptions>
</ChannelUnity>
```

The subscriptions each return a **ServiceSku** element. This identifies the type of channel that the subscription refers to. Please see Appendix for the possible values.

StoreData

This call allows you to specify one or more stores which act as the source of product data to be synchronized with the ChannelUnity account.

Input

```
<ChannelUnity>
    <MerchantName>merc</MerchantName>
    <Authorization>ZWd....DI=</Authorization>
    <ApiKey>123123</ApiKey>
    <RequestType>StoreData</RequestType>
    <Payload>
        <StoreList>
            <Store>
                <FriendlyName>My Store</FriendlyName>
                <URL>http://xxxx</URL>
                <MainCountry>UK</MainCountry>
                <FrameworkType>Custom</FrameworkType>
                <WebsiteId>1</WebsiteId>
                <StoreId>1</StoreId>
                <StoreviewId>1</StoreviewId>
            </Store>
[  <Store> element repeated for each store  ]

        </StoreList>
    </Payload>
</ChannelUnity>
```

The FriendlyName is a human-readable name for the store, and is shown in the ChannelUnity interface under Setup -> Online Stores. The URL identifies the location of the store, and is important to be correct if you wish to use the callbacks in this API. The MainCountry is a two letter country code to identify the store location (not currently used). The FrameworkType identifies common shopping cart types, when recognised, this shows an appropriate icon in the ChannelUnity interface. The WebsiteId, StoreId, and StoreviewId are optional, and were designed primarily for Magento support. These can be used to hold any values which are then passed back to the URL given when callbacks take place. WebsiteId, StoreId, and StoreviewId each must contain an integer value of 1 or greater.

Output

```
<ChannelUnity>
    <CreatedStoreId>1</CreatedStoreId>

[  <CreatedStoreId> element repeated for each store  ]

    <Status>OK</Status>
</ChannelUnity>
```

The response of the StoreData call is a list of Store ID numbers which uniquely identify each store. When a store is attempted to be created with a URL which already exists, no further stores are created, and the existing store is updated.

CategoryData

The CategoryData API allows you to specify the product categories available.

Input

```
<ChannelUnity>
    <MerchantName>merch</MerchantName>
    <Authorization>T...==</Authorization>
    <ApiKey>123123123</ApiKey>
    <RequestType>CategoryData</RequestType>
<Payload>
  <CategoryList>
    <URL>http://xxxx</URL>
    <FrameworkType>Custom</FrameworkType>
    <WebsiteId>1</WebsiteId>
    <StoreId>1</StoreId>
    <StoreviewId>1</StoreviewId>
    <Category>
        <ID>1</ID>
        <Name>Category1</Name>
        <Position>1</Position>
        <CategoryPath>1</CategoryPath>
        <ParentID>0</ParentID>
        <Level>1</Level>
    </Category>

[  <Category> element repeated for each category  ]

  </CategoryList>
</Payload>
</ChannelUnity>
```

The CategoryList elements URL, FrameworkType, WebsiteId, StoreId, and StoreviewId must match those previously given to the StoreData call.

The values to be specified for each category are as follows.
ID - An integer value 2 or greater to identify your category or category folder. These ID values must be unique. (The value 1 is reserved for the Root category).
Name - Any name that you wish to use to identify your category or category folder.
Position - The sequence within the same folder of categories. An integer value 1 or greater.
CategoryPath - Specifies the tree structure for this category. There is always the Root category specified in this path. For example if you have a Category folder with ID 2 and a leaf category of

ID 3, the CategoryPath for category ID 3 is “1/2/3”.
ParentID - The category ID of the parent. This is usually the penultimate category ID in the category path.
Level - The indentation level for the current category in its category tree. This is the number of category IDs in the category path excluding the Root.

Output

```
<ChannelUnity>
<Status>OK</Status>
</ChannelUnity>
```

ProductAttributes

This call allows custom product attributes (fields) to be specified along with a type and name.

Input

```
<ChannelUnity>
    <MerchantName>merc</MerchantName>
    <Authorization>T.....==</Authorization>
    <ApiKey>123123123</ApiKey>
    <RequestType>ProductAttributes</RequestType>
    <Payload>
        <ProductAttributes>
            <Attribute>
                <Name>my_attr_code</Name>
                <Type>int</Type>
                <FriendlyName>My attribute code</FriendlyName>
            </Attribute>

[  <Attribute> element repeated for each attribute  ]

        </ProductAttributes>
    </Payload>
</ChannelUnity>
```

Output

```
<ChannelUnity>
    <Status>OK</Status>
</ChannelUnity>
```

It is not strictly necessary to call ProductAttributes but is useful for when a lot of custom attributes are uploaded in product data. The ProductAttributes call associates an attribute code

with a friendly name which is displayed in the ChannelUnity user interface at Setup -> Describe Product Fields.
Each attribute is given the following information.
Name - The attribute code as used in the ProductData messages. This code can contain only: a-z, 0-9 or underscore. The maximum length is 60 characters.
Type - The type of the attribute - for information purposes - one of int, decimal, text.
FriendlyName - The friendly name of this attribute to be displayed to the user.

ProductData

Allows you to send information on a product or products to the ChannelUnity account.

Input

```
<ChannelUnity>
    <MerchantName>merch</MerchantName>
    <Authorization>TWF0dFR...==</Authorization>
    <ApiKey>123123123</ApiKey>
    <RequestType>ProductData</RequestType>
<Payload>
<Products>
    <Product>
        <RemoteId>1</RemoteId>
        <SourceURL>http://xxxx</SourceURL>
        <ProductType>Default</ProductType>
        <Title>My widget</Title>
        <Description>This does bla bla</Description>
        <SKU>D123</SKU>
        <Price>11.99</Price>
        <Quantity>10</Quantity>
        <Category>2</Category>
        <CategoryName>Cat123</CategoryName>
        <Image>http://www.site.com/files/2012/12/prod.jpg</Image>
    </Product>
</Products>
</Payload>
</ChannelUnity>
```

Output

```
<ChannelUnity>
    <Status>OK</Status>
</ChannelUnity>
```

Multiple products can be given in each Product message. Aim for product message less than 1MB each. The example above is the simplest example i.e. a standalone product with no variations or custom attributes. It shows the minimum product information:

- **RemoteId** - The unique integer ID of the product in your own system. This is a value 1 or greater.
- **SourceURL** - The same URL as provided to earlier API calls.
- **ProductType** - Set this as “Default”.
- **Title** - The title or name of the product.
- **Description** - The main description body text to describe the product. HTML data is

allowed (which can be placed within CDATA tags). NB marketplaces vary in the level of HTML that is allowed on listings.

- **SKU** - Stock keeping unit. This is the field that ChannelUnity uses to uniquely identify a product.
- **Price** - The price that the product is to be sold for. NB. custom fields can be used to specify different prices on different marketplaces. Also the currency for prices is set in ChannelUnity under Setup -> Describe Product Fields.
- **Quantity** - The quantity on hand of this product.
- **Category** - The category ID of this product as matching the ID value of the CategoryData call. Multiple categories are given via a comma separated list, e.g. 4,14,29.
- **CategoryName** - The of the category in which this product belongs. (optional).
- **Image** - URL to the main image for this product. Must be a publicly accessible URL.

The next example demonstrates a parent product with two child products. This a shoe that comes in two sizes. The XML message for this is given below. Note that all the related products do not need to be specified in the same message. However, the child products should be supplied first, and then the parent product such that the child products can be correctly associated with the parent.

Input

```
<ChannelUnity>
    <MerchantName>merch</MerchantName>
    <Authorization>TWF0dFR...==</Authorization>
    <ApiKey>123123123</ApiKey>
    <RequestType>ProductData</RequestType>
<Payload>
<Products>
    <Product>
        <RemoteId>11</RemoteId>
        <SourceURL>http://YOUR_URL/</SourceURL>
        <StoreViewId>0</StoreViewId>
        <ProductType>Default</ProductType>
        <Title>Nike Trainers</Title>
        <Description>Great style for the new season.</Description>
        <SKU>NTPARNT-8</SKU>
        <Price>45.99</Price>
        <Quantity>10</Quantity>
        <Category>3</Category>
        <CategoryName>Shoes</CategoryName>
        <Image>http://bit.ly/16qYx5H</Image>
        <Custom>
            <colour>White</colour>
            <size>8</size>
        </Custom>
```

```
    </Product>
    <Product>
        <RemoteId>12</RemoteId>
        <SourceURL>http://YOUR_URL/</SourceURL>
        <StoreViewId>0</StoreViewId>
        <ProductType>Default</ProductType>
        <Title>Nike Trainers</Title>
        <Description>Great style for the new season.</Description>
        <SKU>NTPARNT-9</SKU>
        <Price>45.99</Price>
        <Quantity>10</Quantity>
        <Category>3</Category>
        <CategoryName>Shoes</CategoryName>
        <Image>http://bit.ly/16qYx5H</Image>
        <Custom>
            <colour>White</colour>
            <size>9</size>
        </Custom>
    </Product>
    <Product>
        <RemoteId>10</RemoteId>
        <SourceURL>http://YOUR_URL/</SourceURL>
        <StoreViewId>0</StoreViewId>
        <ProductType>Default</ProductType>
        <Title>Nike Trainers</Title>
        <Description>Great style for the new season.</Description>
        <SKU>NTPARNT</SKU>
        <Price>45.99</Price>
        <Quantity>10</Quantity>
        <Category>3</Category>
        <CategoryName>Shoes</CategoryName>
        <Image>http://bit.ly/16qYx5H</Image>
        <RelatedSKUs>
            <SKU>NTPARNT-8</SKU>
            <SKU>NTPARNT-9</SKU>
        </RelatedSKUs>
        <Variations>
            <Variation>colour</Variation>
        </Variations>
        <Custom>
            <colour>White</colour>
        </Custom>
    </Product>
    <RangeNext>0</RangeNext>
    <TotalProducts>3</TotalProducts>
</Products>
</Payload>
</ChannelUnity>
```

ProductDataUpdate

Allows you to send information on a product or products to the ChannelUnity account.
This creates the product if the given SKU doesn't exist or updates the product if the SKU does exist. Note that only the SKU and SourceURL elements are required here. Include further elements for the fields that you wish to update only. Unlike ProductData where unspecified fields are erased, the previous values will remain when there is an unspecified field in ProductDataUpdate. The format of the Payload is the same as with ProductData.

Input

```
<ChannelUnity>
    <MerchantName>merch</MerchantName>
    <Authorization>TWF0dFR...==</Authorization>
    <ApiKey>123123123</ApiKey>
    <RequestType>ProductDataUpdate</RequestType>
<Payload>
<Products>
    <Product>
        <SourceURL>http://xxxx</SourceURL>
        <SKU>D123</SKU>
        <Price>11.99</Price>
        <Quantity>10</Quantity>
        <Image>http://www.site.com/files/2013/12/prodnew.jpg</Image>
    </Product>
</Products>
</Payload>
</ChannelUnity>
```

Output

```
<ChannelUnity>
    <Status>OK</Status>
</ChannelUnity>
```

ProductDataLite

Allows you to send quantity and price for a given SKU. Many products' data can be uploaded at once. 10000 entries per upload maximum. Note that in the upload data, the given quantity applies to the standard Quantity of the product, and the given price applies to the standard Price value for the product. No other attributes can be affected by this call.

Input

```
<ChannelUnity>
    <MerchantName>merch</MerchantName>
    <Authorization>TWF0dFR...==</Authorization>
    <ApiKey>123123123</ApiKey>
    <RequestType>ProductDataLite</RequestType>
    <Payload>
        <Products>
            <SourceURL>http://YOUR_URL/</SourceURL>
            <StoreViewId>0</StoreViewId>
            <Data><![CDATA[ * ]]></Data>
        </Products>
    </Payload>
</ChannelUnity>
```

The above message is a template, with the asterisk denoting where the SKUs, quantities and prices are to be provided.

An example Data message looks like the following:

```
XUA918,1,2.99*
CQI924,91,13.99*
AHO382,110,160.50*
```

This shows three SKUs, with quantities and prices in that order. Generally each product is provided as follows:

[SKU] , [Qty] , [Price] * \n

Output

```
<ChannelUnity>
    <Info>SKU WGW252 HAS CHANGED</Info>
    <Info>SKU WYB352 not changed</Info>
    <Length>1298</Length>
    <Status>OK</Status>
</ChannelUnity>
```

ProductDelete

Allows you to remove a product by SKU from the connected ChannelUnity account. This action also removes the product from all connected marketplaces if it is listed there.

Input

```
<ChannelUnity>
    <MerchantName>merch</MerchantName>
    <Authorization>TWF0dFR...==</Authorization>
    <ApiKey>123123123</ApiKey>
    <RequestType>ProductDelete</RequestType>
    <Data>YOUR_SKU</Data>
</ChannelUnity>
```

Output

```
<ChannelUnity>
    <Status>OK</Status>
</ChannelUnity>
```

VerifyNotification

Allows you to check that a notification you receive originates from ChannelUnity and decrypts the data given to it.

Input

```
<ChannelUnity>
    <MerchantName>merch</MerchantName>
    <Authorization>TWF0dFR...==</Authorization>
    <ApiKey>123123123</ApiKey>
    <RequestType>VerifyNotification</RequestType>
    <Payload>your_encrypted_data</Payload>
</ChannelUnity>
```

Output

```
<ChannelUnity>
    <Status>OK</Status>
[  Decrypted data follows  ]
</ChannelUnity>
```

OrderStatusUpdate

Allows an order's status to be updated. Either the order can be shipped (with or without tracking details), or the order can be cancelled. The corresponding marketplace channel is updated accordingly. To mark an order as shipped use OrderStatus "Complete". To cancel an order use OrderStatus "Cancelled".

Input

```
<ChannelUnity>
    <MerchantName>merch</MerchantName>
    <Authorization>TWF0dFR...==</Authorization>
    <ApiKey>123123123</ApiKey>
    <RequestType>OrderStatusUpdate</RequestType>
    <Payload>
        <SubscriptionID>12</SubscriptionID>
        <OrderID>1122512-5522114-655625</OrderID>
        <OrderStatus>Complete</OrderStatus>
[  Start of optional shipment data  ]
        <ShipmentDate>2013-01-02 16:11:33</ShipmentDate>
        <CarrierName>Royal Mail</CarrierName>
        <ShipmentMethod>First Class</ShipmentMethod>
        <TrackingNumber>GB0000456465</TrackingNumber>
[  End of optional shipment data  ]
    </Payload>
</ChannelUnity>
```

Output

```
<ChannelUnity>
    <Status>OK</Status>
</ChannelUnity>
```

DoPartialAction

Performs a partial refund of an order, or a partial shipment of an order. The example below illustrates one partial refund and one partial shipment in a single message. i.e. Each DoPartialAction message can contain more than one PartialActionItem.

Input

```
<ChannelUnity>
    <MerchantName>merch</MerchantName>
```

```
    <Authorization>TWF0dFR...==</Authorization>
    <ApiKey>123123123</ApiKey>
    <RequestType>DoPartialAction</RequestType>
    <Payload>
        <SubscriptionID>12</SubscriptionID>
        <OrderID>1122512-5522114-655625</OrderID>
        <PartialActionDate>2013-01-02 16:11:33</PartialActionDate>
        <PartialActionItem>
            <Action>Ship</Action>
            <SKU>AI022</SKU>
            <QtyAffected>1</QtyAffected>
[  Start of optional shipment data  ]
            <ShipCarrier>DHL</ShipCarrier>
            <ShipTracking>YU7822341</ShipTracking>
[  End of optional shipment data  ]
        </PartialActionItem>
        <PartialActionItem>
            <Action>Refund</Action>
            <Amount>1.92</Amount>
            <SKU>AI082</SKU>
            <QtyAffected>1</QtyAffected>
        </PartialActionItem>
[  <PartialActionItem> element repeated as required  ]
    </Payload>
</ChannelUnity>
```

Output

```
<ChannelUnity>
    <Status>OK</Status>
</ChannelUnity>
```

Callbacks

This section contains all the available callbacks that the ChannelUnity Connector Kit implementation can expect to receive.
The type of notification is always indicated in the Type element which is a child of the Notification element.

Ping

This request is sent from ChannelUnity to the cart to ensure the credentials provided within the cart installation are adequate to make API calls.
When receiving a Ping, the cart should attempt a ValidateUser request and return whether or not this call was successful.

Input

```
<ChannelUnity>
    <Notification>
        <ID>50f9de15b9363</ID>
        <Timestamp>18-01-2013 23:43:17</Timestamp>
        <Type>Ping</Type>
        <Payload></Payload>
    </Notification>
</ChannelUnity>
```

Output

```
<ChannelUnity>
    <Status>OK</Status>
</ChannelUnity>
```

GetAllSKUs

When the connector kit receives this notification it should respond with a list of all SKUs currently available in the product catalog.

Input

```
<ChannelUnity>
    <Notification>
        <ID>50f9de15b9363</ID>
        <Timestamp>18-01-2013 23:43:17</Timestamp>
        <Type>GetAllSKUs</Type>
        <Payload></Payload>
    </Notification>
</ChannelUnity>
```

Output

```
<ChannelUnity>
    <RequestType>GetAllSKUs</RequestType>
    <Payload>
    <SKU>D2484</SKU>
    <SKU>T22584</SKU>
    <SKU>D2411</SKU>
    <SKU>P2004</SKU>
[  <SKU> element repeated for each SKU  ]

    </Payload>
</ChannelUnity>
```

ProductData

This notification is sent when ChannelUnity requires to obtain product data for the account. Use the VerifyNotification call to decrypt the payload.

Input

```
<ChannelUnity>
    <Notification>
        <ID>50f9de15f1e84</ID>
        <Timestamp>18-01-2013 23:43:17</Timestamp>
        <Type>ProductData</Type>
        <Payload>P+V....HbM=</Payload>
    </Notification>
</ChannelUnity>
```

Input payload example

```
<MerchantName>matttest123</MerchantName>
<RangeFrom>750</RangeFrom>
<SourceId>1</SourceId>
<FriendlyName>My Store</FriendlyName>
<URL>http://94.1.1.158</URL>
<MainCountry></MainCountry>
<FrameworkType>Custom</FrameworkType>
<WebsiteId>1</WebsiteId>
<StoreId>1</StoreId>
<StoreviewId>1</StoreviewId>
```

Output

```
<ChannelUnity>
    <RequestType>ProductData</RequestType>
    <Payload>
    <Products>
    <Product>
        <RemoteId>751</RemoteId>
        <SourceURL>http://94.193.82.158:80/</SourceURL>
        <StoreViewId>0</StoreViewId>
        <ProductType>Default</ProductType>
        <Title>My widget</Title>
        <Description>This does bla bla</Description>
        <SKU>D123</SKU>
        <Price>11.99</Price>
        <Quantity>10</Quantity>
```

```
        <Category>2</Category>
        <CategoryName>Cat123</CategoryName>
        <Image>http://www.aaa/12/badge-industry-bronze.jpg</Image>
    </Product>
    <RangeNext>0</RangeNext>
    <TotalProducts>1</TotalProducts>
    </Products>
    </Payload>
</ChannelUnity>
```

In the returned product data is the total number of products available. The **RangeNext** should contain an ID which is passed back to the connector kit from ChannelUnity when the next range of products is needed. This allows the connector kit to provide products to ChannelUnity in batches for when there is a large amount of products.

CartDataRequest

This notification is sent to the cart connector when the ChannelUnity account is ready to receive the list of stores, categories and attributes. The implementation should be to simply invoke each of the StoreData, CategoryData and ProductAttributes calls and return the statuses for each.

Input

```
<ChannelUnity>
    <Notification>
        <Type>CartDataRequest</Type>
        <ID>507257</ID>
        <URL>http://94.193.1.2/channelunity/api/index</URL>
        <Timestamp>2013-01-21 09:45:14</Timestamp>
    </Notification>
</ChannelUnity>
```

Output

```
<ChannelUnity>
    <StoreStatus><Status>OK</Status></StoreStatus>
    <CategoryStatus>OK</CategoryStatus>
    <ProductAttributeStatus>OK</ProductAttributeStatus>
</ChannelUnity>
```

OrderNotification

This notification is sent to the cart when an order is placed on one of the marketplace channels connected to the CU account. This is also sent when the status of an order changes. Valid order statuses are in the appendix.

Input

```
<ChannelUnity>
    <Notification>
    <ID>50fdd17aeefdf</ID>
    <Timestamp>21-01-2013</Timestamp>
    <Type>OrderNotification</Type>
    <Payload>P+V...abg==</Payload>
    </Notification>
</ChannelUnity>
```

Input payload example

```
<MerchantName>matttest123</MerchantName>
<SourceId></SourceId>
<FriendlyName>My Store</FriendlyName>
<URL>http://94.193.82.158</URL>
<MainCountry></MainCountry>
<FrameworkType>Custom</FrameworkType>
<WebsiteId>1</WebsiteId>
<StoreId>1</StoreId>
<StoreviewId>1</StoreviewId>
<SubscriptionId>13</SubscriptionId>
<SkuAttribute>sku</SkuAttribute>
<Orders>
    <Order>
        <ServiceSku>CU_AMZ_UK</ServiceSku>
        <OrderId>112211-08888-9913813</OrderId>
        <PurchaseDate>2012-11-01 12:12:13</PurchaseDate>
        <Currency>GBP</Currency>
        <OrderFlags></OrderFlags>
        <OrderStatus>Processing</OrderStatus>
        <ShippingInfo>
            <RecipientName><![CDATA[Mr Test]]></RecipientName>
            <Email><![CDATA[m2048@amazon.co.uk]]></Email>
            <Address1><![CDATA[1 High St]]></Address1>
            <Address2><![CDATA[]]></Address2>
            <Address3><![CDATA[]]></Address3>
```

```
            <City><![CDATA[Manchester]]></City>
            <State><![CDATA[Manchester]]></State>
            <PostalCode><![CDATA[M1 1AA]]></PostalCode>
            <Country><![CDATA[GB]]></Country>
            <PhoneNumber><![CDATA[01612202222]]></PhoneNumber>
            <ShippingPrice>1.99</ShippingPrice>
            <ShippingTax>0.00</ShippingTax>
            <Service><![CDATA[]]></Service>
            <DeliveryInstructions><![CDATA[]]></DeliveryInstructions>
            <GiftWrapPrice>0.00</GiftWrapPrice>
            <GiftWrapTax>0.00</GiftWrapTax>
            <GiftWrapType></GiftWrapType>
            <GiftMessage><![CDATA[]]></GiftMessage>
        </ShippingInfo>
        <BillingInfo>
            <Name><![CDATA[Mr Buyer]]></Name>
            <Email><![CDATA[unknown@channelunity.com]]></Email>
            <PhoneNumber><![CDATA[]]></PhoneNumber>
        </BillingInfo>
        <OrderItems>
            <Item>
                <SKU><![CDATA[D532]]></SKU>
                <Name><![CDATA[My widget]]></Name>
                <Quantity>1.000</Quantity>
                <Price>10.00</Price>
                <Tax>0.00</Tax>
            </Item>
[  <Item> element repeated for each unique SKU in the order  ]

        </OrderItems>
    </Order>
[  <Order> element repeated for each order  ]

</Orders>
```

OrderFlags contains information which is marketplace-specific such as whether the order has been fulfilled by Amazon. In this case the value of **OrderFlags** is AMAZON_FBA.

The **Tax** element is only populated when there is a country or state-specific sales tax (e.g. in the United States). European style VAT this is included in the **Price** element. The **Price** is the per item price for each single unit.

Output

Return an **Imported** element for each order which is successfully received, and an **Exception**

and **NotImported** element for each order not successfully received. The value in the element is the remote order ID which identifies the order. If the order is not imported, CU will attempt to send the order again in the next 2 minutes.

```
<ChannelUnity>
      <Imported>11100011-662200-22111</Imported>
      <Imported>11100011-662200-22510</Imported>
      <Exception>Order not stored successfully.</Exception>
      <NotImported>11100011-662200-22510</NotImported>
</ChannelUnity>
```

Appendix

ChannelUnity Channel Type SKUs

Value	Description
Amazon marketplaces	
CU_AMZ_COM	Amazon.com marketplace
CU_AMZ_UK	Amazon.co.uk marketplace
CU_AMZ_DE	Amazon.de marketplace
CU_AMZ_FR	Amazon.fr marketplace
CU_AMZ_ES	Amazon.es marketplace
CU_AMZ_JP	Amazon.co.jp marketplace
CU_AMZ_CN	Amazon.cn marketplace
CU_AMZ_IT	Amazon.it marketplace
CU_AMZ_CA	Amazon.ca marketplace
eBay marketplaces	
CU_EBAY_COM	eBay.com marketplace
CU_EBAY_UK	eBay.co.uk marketplace
CU_EBAY_FR	eBay.fr marketplace
CU_EBAY_DE	eBay.de marketplace
(more to be added)	
Rakuten marketplaces	
CU_RAKU_COM	Rakuten Shopping marketplace (rakuten.com)
CU_PLAY_COM	Play.com marketplace

ChannelUnity Order Statuses

Value	Description
`OnHold`	Order is pending for payment or awaiting stock or other information.
`Processing`	Order is ready to be dispatched to the customer by the retailer. Payment has been taken from the customer.
`Complete`	The order has been shipped to the customer.
`Cancelled`	The order has been cancelled, either by the customer or the retailer.