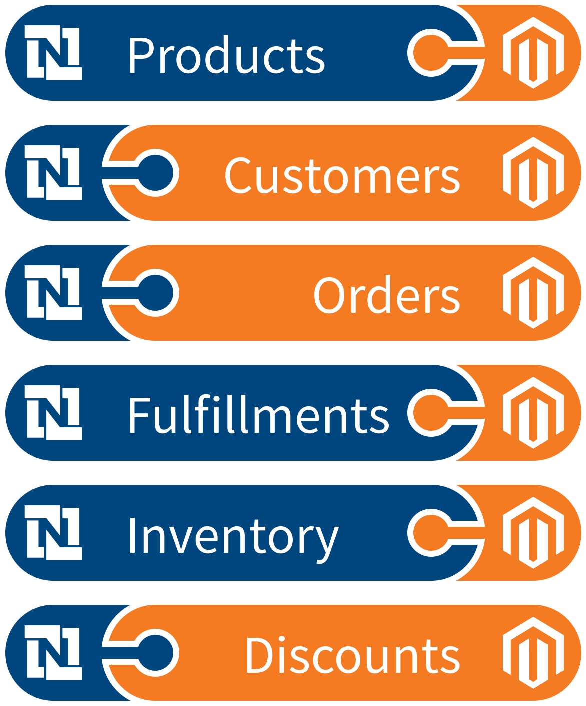
NetSuite Connector For Magento

Join the growing number of eCommerce retailers who are eliminating costly errors and wasted time maintaining their Magento store and NetSuite ERPs.

NetSuite Connector for Magento bridges the gap between your NetSuite ERP and your Magento store.

No longer will you need to manually update orders, products, or customers in two systems.

With **NetSuite Connector**, orders, products, inventory, and even customer information are synced on your schedule giving you the freedom to concentrate on what matters.



Product data is managed in NetSuite and then pushed to Magento.

New and updated **customer information** in Magento flows into NetSuite.

New orders are seamlessly pushed from Magento to NetSuite.

Fulfillments with order status updates push from NetSuite to Magento.

Inventory levels are synchronized between Magento and NetSuite.

Discounts in Magento are passed into NetSuite.

Saves Time

With the time saved allowing NetSuite to communicate and sync with Magento, you can focus more time on your business's success.

100% Code Ownership

The NetSuite Connector code sits on your server, giving you full control and freedom to customize for your needs. The code is yours.

Low Cost

NetSuite Connector for Magento has a fixed cost of $9,000. There are no recurring costs. If you don't need any changes, you owe nothing.

Time to Delivery

Time is money. From signed agreement to Go Live, NetSuite Connector for Magento can be fully implemented in 4 weeks.

Communication Queue

Our communication queue makes your frontend fast and reliable. NetSuite does not need to be running for your site to function properly.

ROCKET WEB

www.rocketweb.com/netsuite-connector-for-magento

General architecture

Used technologies

The NetSuite connector is implemented as a Magento extension, 100% open source. It does not use any external service other than the NetSuite instance it communicates with.

The communication with NetSuite happens via SuiteTalk , which is a SOAP service. It makes use of the PHP Toolkit , a light wrapper over SOAP provided and maintained by NetSuite

Possible Flows

In all cases, NetSuite is assumed to be the "master".

Product synchronization: Product data is imported from NetSuite, it is assumed the catalog is managed there. However, it is possible to have specific product fields unmapped, allowing you to change information directly in Magento, i.e. you may decide to do category association directly in Magento

Stock synchronization: Stocks are always pushed from NetSuite to Magento, never the other way around.

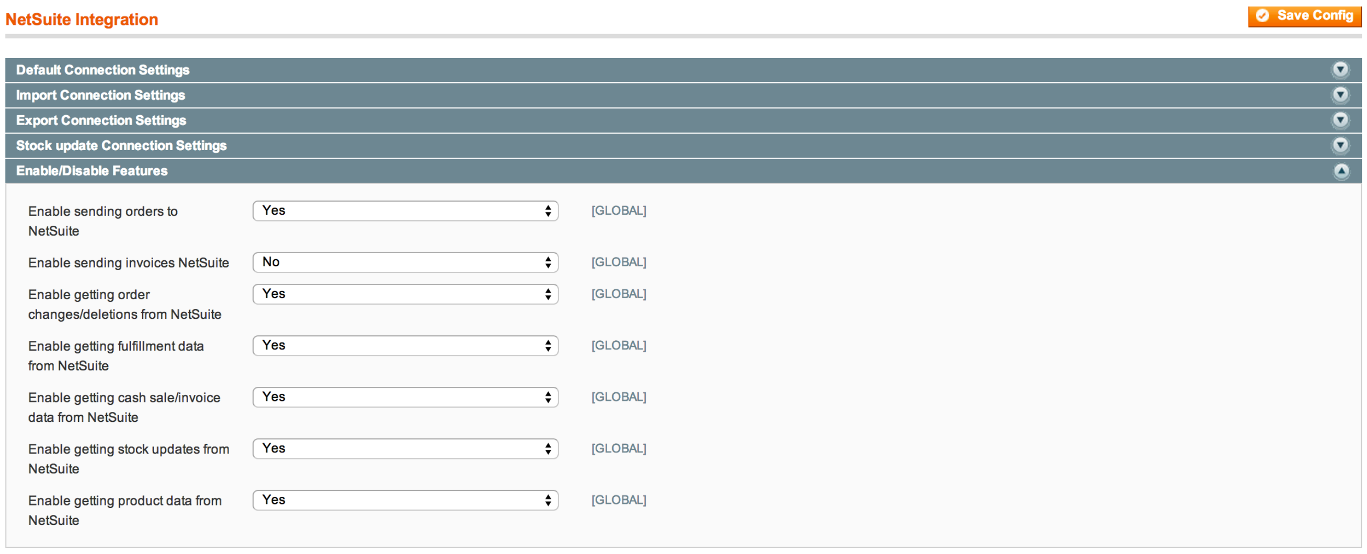
Order synchronization: A new order is pushed from Magento to NetSuite. Further changes to orders in NetSuite are synchronized back in Magento (status changes, order edits)

Invoice synchronization: Any invoice created in Magento is pushed to NetSuite as a cash sale or invoice. Any cash sale/invoice created in NetSuite is pushed back to Magento, considering that the parent order came from Magento

Fulfillment synchronization: Any fulfillment in NetSuite done on an order that came from Magento is pushed to Magento as a shipment. Magento-created shipments are never pushed to NetSuite

Disabling Flows

In case you don't need all flows above, they can be disabled in the connector's configuration screen:


NetSuite Integration
Save Config
Default Connection Settings
Import Connection Settings
Export Connection Settings
Stock update Connection Settings
Enable/Disable Features

Enable sending orders to NetSuite	Yes	[GLOBAL]
Enable sending invoices NetSuite	No	[GLOBAL]
Enable getting order changes/deletions from NetSuite	Yes	[GLOBAL]
Enable getting fulfillment data from NetSuite	Yes	[GLOBAL]
Enable getting cash sale/invoice data from NetSuite	Yes	[GLOBAL]
Enable getting stock updates from NetSuite	Yes	[GLOBAL]
Enable getting product data from NetSuite	Yes	[GLOBAL]

Queueing

All NetSuite operations are queued provides 2 advantages:

- since no NetSuite communication happens synchronously, there is no slowness or down time for the website in case the NetSuite instance is slow or down
- in case an error occurs, the item (i.e. order, invoice) is kept in the queue. An error message is triggered and you get the chance to investigate, fix then push the item without losing it

There are 3 different queues:

- import
- export
- delete (contains delete operations you may be doing in NetSuite)

The queues can be visualized at NetSuite->Import Status/Export Status/Delete Status. If you see an item in any of the queues, it means that it was not processed yet.

Object Mapping

Magento object class name	**NetSuite object class name, as defined by SuiteTalk**
Mage_Catalog_Model_Product	InventoryItem
Mage_Sales_Model_Order	SalesOrder
Mage_Sales_Model_Order_Invoice	Invoice or CashSale (configuration option)
Mage_Sales_Model_Order_Shipment	ItemFulfillment

Product synchronization

Field mapping

The core of the product synchronization is the map defined at **NetSuite->Settings->Product synchronization->Field Mapping .** This mapping defines the link between Magento attributes and Inventory Items custom/standard fields. Each mapping line contains 3 elements:

- **NetSuite field name**. Either the custom field id from NetSuite or a standard field name, as defined in the NetSuite SuiteTalk documentation
- **NetSuite settings**. Various options explained in the table below.
- **Magento** . A drop-down where you can pick up a Magento attribute. If the Magento attribute is a dropdown or multi-select, then the options will be created automatically. I.e. if you import colors, you only need to create the "color" attribute in Magento and map it, no need to populate it with all possible colors before hand

Setting	Description	Extra parameters and description
Standard Field	A InventoryItem standard field. Please refer to the SuiteTalk documentation for a full list	none
Record Field	A standard field that points to a standard record	**Search class name**: For standard records, SuiteTalk defines search classes, i.e. SiteCategorySearch. You need to find the class name in the documentation and add it here. **Name field**: the standard record searches return objects of specific types, i.e. SiteCategory. They are documented in SuiteTalk reference. The value you put in this field must be the object property that is a good textual representation for import, i.e. name
Custom Field - simple	A custom field defined in NetSuite that does not represent a list	none
Custom Field - list	A custom field defined in NetSuite that represents a list	**NetSuite list id** - the internal NetSuite id for the list
Custom Field - checkbox	A custom record defined in NetSuite that has the checkbox type	**Value** - define the value that is to be imported in Magento when the checkbox is checked in NetSuite, i.e. "1" or "y"
Custom Field - custom record	A custom field defined in NetSuite that represents a custom record	**NetSuite custom record id** - the internal NetSuite id for the custom record

You don't need to map all attributes. If you want to handle an attribute directly in Magento, then don't create any mapping for it so it will never be overwritten by NetSuite data. **However, you do have to map all the attributes that are used in the creation of configurable products.**

One to many mappings

If you want to map multiple NetSuite fields into a single Magento attribute, you can add all the NetSuite fields in the "Netsuite field name" map item, comma delimited. By default the values will be merged by adding space between them, but a custom event ("netsuite_product_fields_merge") is also thrown, allowing you to write code to do a custom merge.

Product type mapping

NetSuite Type	Magento type
InvetoryItem matrix	Configurable product

InventoryItem matrix child	Simple product
KitItem	Bundle product
AssemblyItem	Bundle product

Import / reimport scripts

Importing all products

Script at MAGE_ROOT/shell/netsuite/importAllProducts.php

Param name	Possible values	Description
all	none, just add as param	Import all products, i.e. will rewrite exiting products
add-only	none, just add as param	The counterpart of "all". If mode is add-only, existing products will not be updated
wipe-existing	none, just add the param	Start by deleting all the products imported from NetSuite
from-date	YYYY-MM-DD date	Import only the products that are modified after the specified date
verbose	none, just add param	Display progress information
resume-at	integer	Products are grabbed in pages. If you want to resume an import at a specific page, add the page number here
type	simple, configurable, bundle, assembly, all	allows to import only a specific product type

Example:

```
nohup php importAllProducts.php --all --verbose --type simple --resume-at 2 &>
/tmp/netsuite_log &
```

The above will import simple products, displaying output and starting with page 2. It will also update existing products. The script will be running detached from the current shell.

Import a single product

Script at MAGE_ROOT/shell/netsuite/importSingleProduct.php

Param name	Possible values	Description
id	netsuite internal id	The netsuite internal id
delete-existing	none	whether the product should be deleted first if it exists in Magento

Stock synchronization

Stock synchronization is done by default once in 6 hours. At this time, all stocks will be grabbed from NetSuite and imported in Magento.

The relevant stock settings can be found at **NetSuite->Settings->Stock Options:**

Parameter name	Example value	Description
Location internal id	1	NetSuite supports multiple stock locations (warehouses). Magento does not, so it is necessary to specify the location from where Magento will grab stocks. If you need to grab from multiple locations, you can create a fake location in NetSuite (i.e. "Web"), then write a SuiteScript to combine stocks in there (i.e. add stocks from all other locations). This way Magento can still function by grabbing stocks from a single location.
Saved Stock search internal id	12	See below
Page size for the saved search query	500	The stock information is paged when grabbed. This value defines how many stock values are grabbed at once. 500 is the maximum NetSuite will allow
Update the stocks every N hours	6	This value defines the frequency of the updates. Based on your catalog size, you can tweak this up or down
Stock stored at location level?	Yes	While by default the stock information is present at location level, some users may choose to expose it as a custom field at product level. This flag instructs the connector on where to look for stock information
Quantity field name	custitem_web_inventory	The exact field name that contains the stock number. For fields at product level, only custom fields are accepted. For fields at location level you can specify either custom or standard fields. A new setting will appear, "Quantity field type", where you can define whether the field is standard or custom

Saved stock search

Since stock update is a frequent operation on all products, we wanted to make it as fast as possible. Instead of querying NetSuite for all product information then extracting the stock, a saved search must be created so to grab only the fields we care about. The saved search must contain only 2 fields (it can contain more but they will be ignored):

- product internal id
- the field that contains the stock number at was defined in the "Quantity field name" setting

If the "Quantity Field Name" is at location level, then you will be getting a row for each location. Mitigate this by adding a filter criteria to the search to list only the location you are grabbing from (the one that you defined in "Location internal id")

If you don't use stocks for specific product types (i.e. matrixes do not have stocks, only their subitems have), filter them out with criteria conditions. This way you will prevent unnecessary data processing.

Note that it is required to create the saved search as described above and put its internal NetSuite id inside the "Saved Stock search internal id" setting, stocks will not be imported otherwise.

Customer Export

Exported customer data

- email address
- full name
- address book

Note that for states to be exported properly, they must be defined in NetSuite accordingly. While this is not an issue for US, it may be one for other countries. If you find out that specific customers cannot be exported due to states not being defined, make sure the NetSuite information at **Setup->Company->States/Provinces/Countries** matches the one in Magento.

Customer identifier

In Magento, the customer identifier is the email address and website id, i.e. multiple customers with the same email may exist on different websites if configured. To prevent integrity constraints, we use **email_websiteId** for the **externalId** field in NetSuite, i.e. **john@example.com_1** . While the site suffix may be unnecessary for some merchants, it is there to fully support all Magento configurations. Note that the email without suffix is still stored in the **email** field, on which NetSuite does not impose a unique constraint.

Customer export & sync time

To honor the concept of "customer" in NetSuite, customers are not pushed from Magento when registering, but when placing their first order. After this point, customer data will be pushed again when:

- placing another order
- changing account information

Since customer data can be edited in the Magento front-end, it will always be pushed to NetSuite when changed. This means that any manual edits for the customer in NetSuite will be lost, if affecting the address book, name, or email.

Order export

Orders are queued for export once they are placed.

Customer data

The customer is pushed with the order, as described here. Note that even if an order is placed anonymously, a customer is still created in NetSuite, since it has no concept of anonymous orders

Shipping Address

The shipping address is pushed with the order. As in Magento, the shipping address on the order is a separate entity from the customer's shipping address, so that even if the customer changes it in the future, the one on the order stays the same.

Billing Address

The billing address is pushed with the order. As in Magento, the billing address on the order is a separate entity from the customer's billing address, so that even if the customer changes it in the future, the one on the order stays the same.

Shipping Method

For the connector to work properly, a few settings must be defined at **NetSuite->Settings->Shipping options**:

Setting	Example value	Description
Mapping between Magento and NetSuite shipping methods	Free-"-2 UPS ground-"-5	A mapping between the active shipping methods in Magento and the defined shipping items in Magento. Each line has 3 options: • Shipping method name - a drop down from where you can select any shipping method defined & activated in Magento • Shipping Description - not used currently, leave it empty • NetSuite internal id - the internal id of a shipping item defined in NetSuite at **Lists->Accounting->Shipping Items** Note that the shipping costs will be passed from Magento and they will override the values defined in NetSuite. So this means that you don't necessarily have to have a one to one mapping between Magento and NetSuite, just make it granular enough so that it suits your needs.
If none of the rules above match, use this NetSuite shipping item internal id:	2	As described, a "catch all" for shipping methods that are not mapped. NetSuite won't take in an order with a shipping method that is not mapped, so we still need to pass in a value
Mapping between Magento Tracking number types and NetSuite shipping methods	UPS - 2 DHL - 5	NetSuite will pass back tracking information. Magento supports a pre-defined number of carriers for auto tracking (i.e. clicking a tracking number will take you to the carrier page for the parcel). This map ensures that if you use one of the pre-defined carriers (DHL, UPS, USPS, Fedex), automatic tracking checking will work. If you don't use any of the carriers Magento supports, you can ignore this map.
Default tracking number carrier	DHL	A catch all for if the above map does not match. It defaults to "Custom", which means that the customer will still see the tracking number but without the ability to track the order.
Send tracking information when receiving tracking numbers from NetSuite	Yes	Whether the customer is announced about the tracking number addition/change. This will be done via the standard shipping update mail in Magento. If you choose not to use this feature, the customer can still see the tracking number in his account, but there will be no notice when one gets added or changed.

Payment method

Payment method mapping

As with shipping methods, you need to define a one to one map between the Magento payment methods and the NetSuite ones. Each line contains:

- **Magento payment method** - pick the Magento payment method from the drop down. All active methods are listed
- **Credit card** - it is common to have a single payment method in Magento (i.e. "Cybersource"), but to want to map it to a different NetSuite payment methods in NetSuite based on the credit card type. If in this case, pick a specific credit card type from this drop down. Otherwise, choose "All"
- **NetSuite internal id** - the internal id of a payment method defined in NetSuite at **Setup->Accounting->Accounting Lists, "Payment Method" view**

Payment processor mapping

The payment processor mapping allows you to define the fields and behavior for the passed payment method. Each line item contains:

- **Website**. For credit card processing, you may choose to use a different merchant account for each website. This setting allows to define the mapping for a specific website or for "All"
- **Payment method** - Drop down from where you can select any active Magento payment method
- **NetSuite Internal Id** - the internal id of the credit card processor defined in NetSuite at **Setup->Accounting->Credit Card Processing**. If the specific method does not require credit card processing, you can leave this empty
- **Payment Processor helper class** - defines a few extra fields that are to be passed per payment method. Options are:
    - **Simple** - no extra information is passed other than the mapped payment method. Can be used for most "general" payment methods
    - **Check** - used for check/money order. Will also pass the check/money order number
    - **PayPal / PayPal Express** - supports passing PayPal related information
    - **Cybersource** - use for Cybersource in normal (no tokenization) mode
    - **Cybersourcetokenize** - use with Cybersource in tokenization mode

Make sure you map all the used payment methods. A common omission is "No Payment Information required", which appears when the whole order is covered by user points/store credit. If this is a valid use case for your store, make sure you also map it.

Authorize & Capture mode

If both authorization and capturing is done in Magento, then the integration with NetSuite will be smooth. There is no payment information that needs to be passed except the method name.

Authorize only mode

If you choose to authorize in Magento but capture in NetSuite, then your only option is Cybersource. You also need to:

- make sure the same NetSuite payment processor is enabled in Magento and in NetSuite
- make sure Allow Request ID to Meet Payment Card Field Requirements is enabled in NetSuite's credit card processor settings

Cybersource and multiple shipments in authorize-only mode

If you need to do multiple charges for the same order (i.e. charge on shipment while doing split shipments), there are two options:

CyberSource split shipments

This is the least flexible method, described at https://support.cybersource.com/cybskb/index?page=content&id=C1108&actp=LIST . Basically, if your payment processor allows to, you can charge the same authorization multiple times with the condition that your total charge never goes over the initial authorization. Contact CyberSource to check if your setup supports this.

To use this method, simply choose "Cybersource" as a "Payment Processor helper class" for the payment method to payment processor mapping. The following card-related information will go to NetSuite:

- card type
- transaction id (part of the P/N ref number)

Tokenization

CyberSource tokenization is the more flexible option in that a CC token will allow you to charge multiple times no matter the payment processor and even charge more than the authorized amount. To use this approach:

1. Contact CyberSource to enable tokenization for your account
2. Make sure the Magento extension you are using supports tokenization
3. Make sure the NetSuite payment methods you are using have the "Tokenized" checkbox checked
4. Inside the NetSuite connector config, choose "Cybersourcetokenized" as a "Payment Processor helper class" for the payment method to payment processor mapping

The following information gets sent to NetSuite:

- card type
- card token (note that although this is sent as part of the CC number field, it is not the number but the token)
- card expiration date
- authorization code
- transaction id (part of the P/N ref number)

Line items and discount

For a product to be passed to NetSuite, it must have a NetSuite internal id, i.e. it must be imported from NetSuite or linked as described here. NetSuite will not take in random SKUs, i.e. products created in Magento that it does not know about.

For discounts, a discount item must be defined in NetSuite. Then its NetSuite internal id must be entered at **NetSuite->Settings->Order Settings->Discount item internal id**. All the discounts in Magento will be aggregated under this item. The rate will also be overridden. **In case the rate is not overridden, contact NetSuite support. On some instances this feature is off.**

Order states

A Magento to NetSuite (and vice-versa) status mapping can be defined at **NetSuite->Settings->Order Settings->Status Mapping**. Every status change in NetSuite will update the status in Magento according to this map. Note that this does not work the other way around though: only the initials status in Magento will be pushed to NetSuite according to map, all other status changes in Magento will be ignored, as it is assumed all order processing is done in NetSuite.

Custom fields

Custom fields can also be pushed to NetSuite. You can define a map at **NetSuite->Settings->Order Settings->Custom fields that are to be synched between Magento and NetSuite** . Each line contains:

- **NetSuite field name** - the field name as defined in NetSuite. Must be defined at Sales Order level
- **NetSuite field type**:
  - **custom field - simple** : any field of a simple type, i.e. string, text area, number, date etc
  - **custom field list** : field of type list
  - **standard** : an order standard field. The list of possible standard fields can be found in the SuiteTalk documentation
- **NetSuite list internal id:** the internal id of the list, in case the type of the field is list. Leave it empty otherwise
- **Value type:**
  - **Magento order attribute** - whether the value to be passed is an attribute of the Magento order object
  - **Fixed value:** pass the value exactly as defined in the next field. Usecase: always pass the value of "Magento" to a "channel" field
- **Value**. If value type is fixed, the value here will be passed with the order as-is. If value type is "Magento order attribute", then you need to pass the attribute exactly as it appears as a column name in the sales_flat_order table.

Invoice import/export

Invoices management should be happening in NetSuite, but the initial invoice is pushed to NetSuite in case you make use of an authorize & capture payment method in Magento, or create an invoice using any other mean.

At **NetSuite->Settings->Order Settings->Netsuite equivalent for a Magento invoice**, you can choose whether the invoice is pushed to NetSuite as a cash sale or as an invoice.

Any invoice/cash sale created in NetSuite on a sales order that originated from Magento will be sent to Magento.

Fulfillment import

Order fulfillment should happen in NetSuite. You can fulfill the order one or more times, each fulfillment being imported as a shipment in Magento. Tracking information is also passed back to Magento. See order export for how to configure tracking.

Shipments should never be created manually in Magento if order fulfillment import is on. They will not be exported to NetSuite and cause inconsistencies.

Extending the connector

General considerations

It is not recommended to directly change the connector code directly. If you need to change it, the best approach is to create a module in app/code/local, then either hook into one of the below events or use class rewrites.

Events list

netsuite_stock_item_save_before

Description: grants a chance to execute custom code right before the stock for an item is saved
Parameters:

- stock_item type cataloginventory/stock_item, the Magento stock item that is to be saved
- item_search_row, type ItemSearchRowBasic, the stock information from Netsuite

Code example

```
//only add 60% of the Netsuite stock in Magento.
$stockItem = $observer->getEvent()->getStockItem();
/* @var ItemSearchRowBasic $itemSearchRow */
$itemSearchRow = $observer->getEvent()->getItemSearchRow();
$qty = $itemSearchRow->locationQuantityAvailable[0]->searchValue;
$qty = round($qty*0.6);
$stockItem->setQty($qty);
if($qty>0) $stockItem->setIsInStock(1);
else $stockItem->setIsInStock(0);
```

netsuite_bill_address_create_before

Description: occurs just before a billing address is sent to Netsuite
Parameters:

- netsuite_address, type BillAddress

Code example

```
//pad phone numbers to be at least 10 chars.
$netsuiteAddess = $observer->getEvent()->getNetsuiteAddress();
if($this->phoneIsInvalid($netsuiteAddess->billPhone)) {
 $netsuiteAddess->billPhone = $this->getPaddedPhone($netsuiteAddess->billPhone);
}
```

netsuite_ship_address_create_before

Description: occurs just before a shipping address is sent to Netsuite
Parameters:

- netsuite_address, type ShipAddress

Code example

```
//pad phone numbers to be at least 10 chars.
$netsuiteAddess = $observer->getEvent()->getNetsuiteAddress();
if($this->phoneIsInvalid($netsuiteAddess->shipPhone)) {
 $netsuiteAddess->shipPhone = $this->getPaddedPhone($netsuiteAddess->shipPhone);
}
```

netsuite_address_create_before

Description: occurs just before a customer (address book) address is sent to Netsuite
Parameters:

- netsuite_address, type CustomerAddressbook

Code example

```
//pad phone numbers to be at least 10 chars.
$netsuiteAddess = $observer->getEvent()->getNetsuiteAddress();
if($this->phoneIsInvalid($netsuiteAddess->phone)) {
 $netsuiteAddess->phone = $this->getPaddedPhone($netsuiteAddess->phone);
}
```

netsuite_customer_send_before

Description: occurs just before sending a customer to Netsuite
Parameters:

- netstuite_customer, type Customer

Code example

```
//phone number must be at least 7 chars long if not empty
$netsuiteCustomer = $observer->getEvent()->getNetsuiteCustomer();
if($this->phoneIsInvalid($netsuiteCustomer->phone))
 $netsuiteCustomer->phone = $this->getPaddedPhone($netsuiteCustomer->phone);
return $this;
```

netsuite_new_order_send_before

Description: occurs just before sending an order to Netsuite
Parameters:

- magento_order type sales/order, the Magento order object
- netsuite_order type SalesOrder, the Netsuite order

Code example

```
//set all Netsuite orders to be printed
$netsuiteOrder = $observer->getEvent()->getNetsuiteOrder();
$netsuiteOrder->toBePrinted = true;
return $this;
```

netsuite_import_request_before

Description: occurs before an import operation starts
Parameters:

- record_type the type of the record to be imported
- search_object the search request that will grab the to-be-imported requests, type TransactionSearchBasic

netsuite_import_product_created_after

Description: Occurs when a product is imported
Parameters:

- magento_product, type catalog/product, the Magento product object
- netsuite_product, type InventoryItem, the Netsuite product object
- product_is_new, type bool, whether the product previously existed in the db or not

Code example

```
//set product categories
$magentoProduct = $observer->getEvent()->getMagentoProduct();
/** @var InventoryItem $inventoryItem */
$inventoryItem = $observer->getEvent()->getNetsuiteProduct();
foreach($inventoryItem->customFieldList->customField as $customField) {
 if($customField->internalId == 'custitem_magento_category_ids') {
  $categoryIds = explode(',',trim($customField->value));
  if(count($categoryIds)) {
   $magentoProduct->setCategoryIds($categoryIds);
  }
 }
}
```

netsuite_inventory_item_is_importable

Description: gives a chance to write custom code in deciding whether a inventory item is importable or not
Parameters:

- inventory_item, type InventoryItem, the Netsuite product
- is_importablebool, whether the product is importable

Code example

```
//items that do not have the custitem_sendtomagento flag set are not to be imported
$inventoryItem = $observer->getEvent()->getInventoryItem();
$isImportable = $observer->getEvent()->getIsImportable();
foreach($inventoryItem->customFieldList->customField as $customField) {
 /* @var CustomFieldRef $customField */
 if($customField->internalId == 'custitem_sendtomagento') {
  if($customField->value == false) {
   $excludeFromImport = true;
  }
 }
}
$isImportable->setFlag(!$excludeFromImport);
```

netsuite_item_fulfillment_import_save_before

Description: occurs just before a Netsuite fulfillment gets imported as a Magento shipping
Parameters:

- netsuite_shipping, type ItemFulfillment, the Nesuite shipment

- magento_shipping the Magento shipment, not yet saved

Code example

```
//import custom tracking info
/* @var Mage_Sales_Model_Order_Shipment $magentoShipment*/
$magentoShipment = $observer->getEvent()->getMagentoShipping();
/* @var ItemFulfillment $netsuiteShipment */
$netsuiteShipment = $observer->getEvent()->getNetsuiteShipping();
foreach($netsuiteShipment->customFieldList->customField as $customField) {
 if($customField->internalId == 'custbody_shipping_carrier') {
  $carrier = strtolower(trim($customField->value));
  $carrierCode = 'custom';
  if($carrier == 'ups') {
   $carrierCode = 'ups';
  }
  if($carrier == 'usps') {
   $carrierCode = 'usps';
  }
  $tracks = $magentoShipment->getTracksCollection();
  foreach($tracks as $track) {
   $track->setCarrierCode($carrierCode);
  }
 }
}
```

product_map_value_extracted

Description: allows writing custom code when field mapping in product import occurs
Parameters:

- product_map_value, type RocketWeb_Netsuite_Model_Product_Map_Value, the mapping

Code example

```
$productMapValue = $observer->getEvent()->getProductMapValue();
if($productMapValue->getMagentoFieldId() == 'batteries_included') {
 $this->prepareBatteriesIncludedField($productMapValue);
}
return $this;
```

Server configuration

Crons

Main cron

The main NetSuite processing cron is located at MAGE_ROOT/shell/netsuite/netsuiteCron.php. The minimum amount of configuration is to set this cron to run every few minutes:

```
*/2 * * * * php /home/mt_dev/public_html/oDaBS2962/shell/netsuite/netsuiteCron.php
--mode all > /dev/null 2>&1
```

This setting will take care of all NetSuite synchronization: grab product changes, grab stocks, export orders, import invoices etc. Since this is single threaded, it can be slow. If you have multiple NetSuite web users, you can speed up the process by adding multiple cron entries with different modes:

- export - takes care of pushing orders and invoices to NetSuite
- import - takes care of importing products, invoices, orders and fulfillments
- stock - takes care of stock updates

The modes can be combined with a comma. Here is an example that runs exports and imports in parallel:

```
*/2 * * * * php /home/mt_dev/public_html/oDaBS2962/shell/netsuite/netsuiteCron.php
--mode import,stock > /dev/null 2>&1
*/2 * * * * php /home/mt_dev/public_html/oDaBS2962/shell/netsuite/netsuiteCron.php
--mode export > /dev/null 2>&1
```

Tax crons

The connector can grab tax rates from NetSuite and automatically create them in Magento. If you choose to do this, it is recommended to refresh the taxes at least once per month:

```
1 1 1 * * php /home/mt_dev/public_html/oDaBS2962/shell/netsuite/importSalesTaxes.php >
/dev/null 2>&1
```