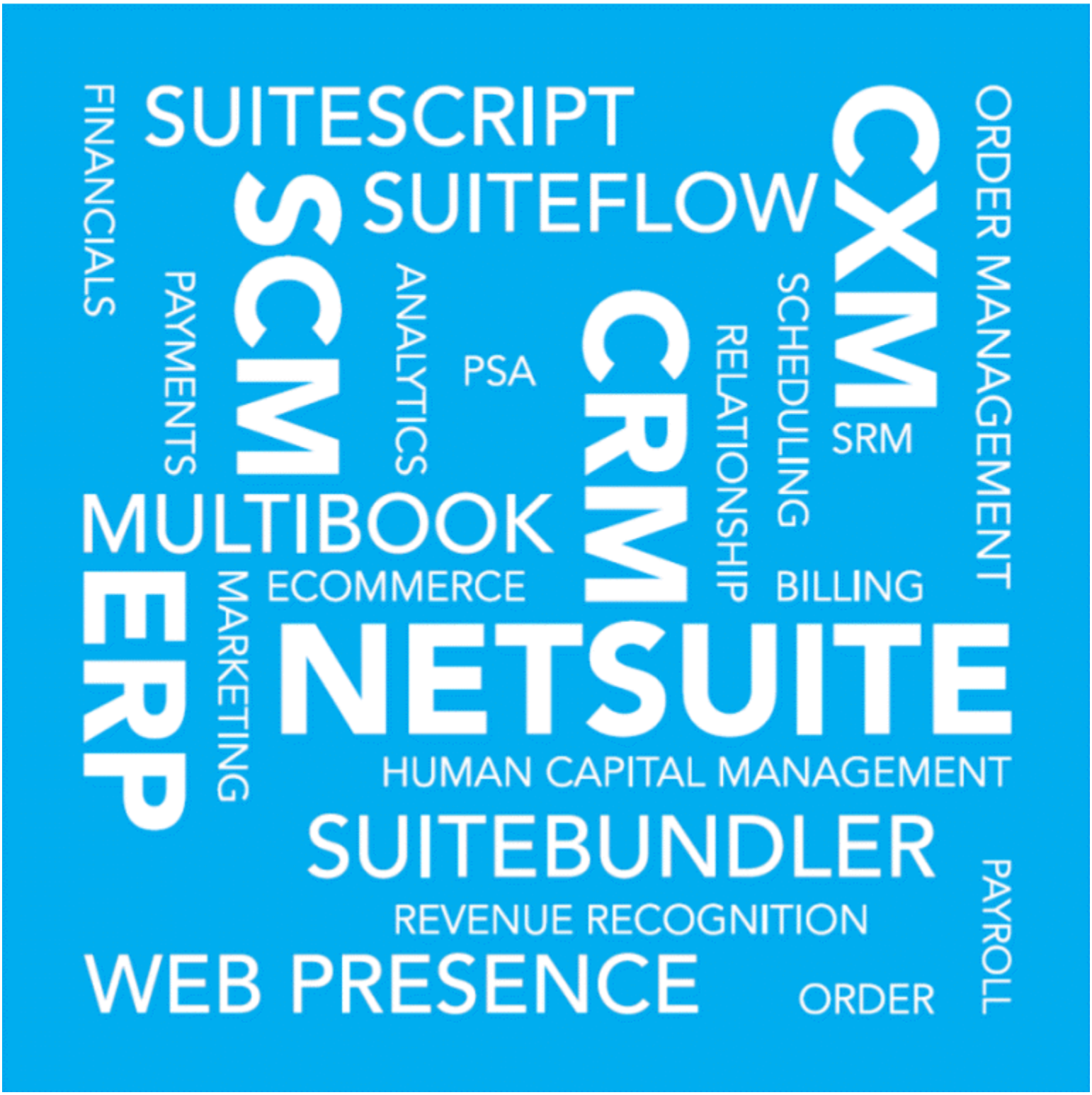
# SuiteCommerce Advanced & Reference Implementations (pre-Denali)



Copyright © 2005, 2017, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

If this document is in public or private pre-General Availability status:

This documentation is in pre-General Availability status and is intended for demonstration and preliminary use only. It may not be specific to the hardware on which you are using the software. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to this documentation and will not be responsible for any loss, costs, or damages incurred due to the use of this documentation.

If this document is in private pre-General Availability status:

The information contained in this document is for informational sharing purposes only and should be considered in your capacity as a customer advisory board member or pursuant to your pre-General Availability trial agreement only. It is not a commitment to deliver any material, code, or functionality,

and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle Master Agreement, Oracle License and Services Agreement, Oracle PartnerNetwork Agreement, Oracle distribution agreement, or other license agreement which has been executed by you and Oracle and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

**Sample Code**

Oracle may provide sample code in SuiteAnswers, the Help Center, User Guides, or elsewhere through help links. All such sample code is provided "as is" and “as available”, for use only with an authorized NetSuite Service account, and is made available as a SuiteCloud Technology subject to the SuiteCloud Terms of Service at www.netsuite.com/tos.

Oracle may modify or remove sample code at any time without notice.

**No Excessive Use of the Service**

As the Service is a multi-tenant service offering on shared databases, Customer may not use the Service in excess of limits or thresholds that Oracle considers commercially reasonable for the Service. If Oracle reasonably concludes that a Customer’s use is excessive and/or will cause immediate or ongoing performance issues for one or more of Oracle’s other customers, Oracle may slow down or throttle Customer’s excess use until such time that Customer’s use stays within reasonable limits. If Customer’s particular usage pattern requires a higher limit or threshold, then the Customer should procure a subscription to the Service that accommodates a higher limit and/or threshold that more effectively aligns with the Customer’s actual usage pattern.

**Beta Features**

Oracle may make available to Customer certain features that are labeled “beta” that are not yet generally available. To use such features, Customer acknowledges and agrees that such beta features are subject to the terms and conditions accepted by Customer upon activation of the feature, or in the absence of such terms, subject to the limitations for the feature described in the User Guide and as follows: The beta feature is a prototype or beta version only and is not error or bug free and Customer agrees that it will use the beta feature carefully and will not use it in any way which might result in any loss, corruption or unauthorized access of or to its or any third party’s property or information. Customer must promptly report to Oracle any defects, errors or other problems in beta features to support@netsuite.com or other designated contact for the specific beta feature. Oracle cannot guarantee the continued availability of such beta features and may substantially modify or cease providing such beta features without entitling Customer to any refund, credit, or other compensation. Oracle makes no representations or warranties regarding functionality or use of beta features and Oracle shall have no liability for any lost data, incomplete data, re-run time, inaccurate input, work delay, lost profits or adverse effect on the performance of the Service resulting from the use of beta features. Oracle’s standard service levels, warranties and related commitments regarding the Service shall not apply to beta features and they may not be fully supported by Oracle’s customer support. These limitations and exclusions shall apply until the date that Oracle at its sole option makes a beta feature generally available to its customers and partners as part of the Service without a “beta” label.

# Table of Contents


SuiteCommerce Advanced Overview ... 1
- Using SuiteCommerce Advanced ... 1

Working with a SuiteCommerce Advanced Web Site ... 4
- Understanding the SuiteCommerce Advanced Template Combiner ... 4
- Web Site Customization with SuiteCommerce Advanced ... 5
  - Using Precedence Overlays for Web Site Customization ... 6
  - Derivative Hosting with SuiteCommerce Advanced ... 8

Reference Implementations Overview ... 18
- Prerequisites for SuiteCommerce Reference Implementations ... 18
- Getting Started with the Reference Implementations ... 19
  - Managing Reference Implementation Installations ... 19
  - Minimum Setup Requirements ... 26
  - Touch Points Available for Reference Implementations ... 36
  - Working with Combination Files ... 36
  - Configuring Backend Objects Using the backend.configuration.js File ... 37
- Customizing Implementations ... 39
  - Using the Custom Folder ... 40
  - Using Template and Macro Files ... 41
  - Debugging and Making a Change to a Site ... 42
- Technology and Framework ... 47
  - Overview ... 47
  - Libraries ... 51
  - SuiteCommerce Reference Implementation Web Assets ... 52
  - Setting Up Your Development Environment ... 55
- Google Analytics for Reference Implementations ... 56
  - Configuring Your Google Analytics Account ... 61
  - Creating Custom Analytics Trackers for 3rd Party Services ... 62
- Handling Localization in Reference Implementations ... 63
  - Setting Up Localization for Your Web Site ... 64
  - Understanding the _.translate() Function ... 65
  - Localized Content from the Content Delivery Service ... 66
- Correlated & Related Items ... 67
  - Prerequisites ... 69
  - Specifying the Number of Related Records to Display ... 69
- List Headers ... 70
  - Setup ... 71
  - Upgrading from a Previous Version ... 74
  - Customizing List Headers ... 74
- Site Management Tools Templates ... 74
  - Page Header and Footer Templates ... 75
  - Home Page Template ... 76
  - Facet Search Page Template ... 76
  - Item Details Template ... 77
  - Customizing Template Files ... 78
- Bronto ... 79
- Reference SuiteApp FAQ's ... 80
- Secure Shopping Domain (pre-Denali) ... 81
  - Migration Tasks ... 81

SuiteCommerce Services ... 98
- Product Merchandising ... 98
  - Understanding Product Merchandising ... 99
  - Creating a Product Merchandising Rule ... 99
  - Preparing Pages for Product Merchandising Zones ... 103

Content Delivery ..... 106
Understanding Content Delivery ..... 106
Using Content Delivery ..... 106
Preparing Pages for Content Delivery ..... 111
Reference ShopFlow ..... 114
Quick Start ..... 115
Working with Faceted Navigation ..... 117
Configuring Facet Fields ..... 117
Customizing the Facet Navigation Panel ..... 122
Customizing and Extending Shop Flow Components ..... 124
Customizing Item URLs for SEO ..... 124
Customizing the View By Area ..... 125
Product Reviews ..... 127
Using the Product Reviews Record ..... 128
Defining Product Review Attributes ..... 129
Modifying Product Review Configuration Defaults ..... 129
Save For Later ..... 131
User Experience ..... 131
Setup ..... 131
Upgrading from Previous Version ..... 134
Customizing Save For Later ..... 134
Social Sharing ..... 134
Pinterest ..... 134
Twitter Product Cards ..... 136
Reference ShopFlow Templates ..... 137
Home Page ..... 138
Product Display Page ..... 139
Product Details Page ..... 141
Product Reviews Pages ..... 142
Shopping Cart Page ..... 146
Reference My Account ..... 151
Working with Permissions ..... 151
Product Lists ..... 154
Product List Custom Records ..... 155
Configuring Product Lists ..... 157
Product List Experiences & Wireframes ..... 158
Billing ..... 165
Billing Experiences & Wireframes ..... 166
Returns ..... 173
User Experience ..... 173
Setup ..... 175
Upgrading from Previous Version ..... 178
Customizing Returns ..... 178
Case Management ..... 180
User Experience ..... 181
Setup ..... 182
Upgrading from a Previous Version ..... 185
Customizing Case Management ..... 185
Quotes ..... 185
User Experience ..... 185
Setup ..... 186
Upgrading from a Previous Version ..... 188
Customizing Quotes ..... 188
Reference My Account Templates ..... 188
Overview Page ..... 189

Orders ..... 190
Product Lists ..... 196
Billing ..... 199
Profile Information ..... 202
Address Book Page ..... 203
Credit Cards Page ..... 204
Email Preferences Page ..... 205
Reference Checkout ..... 207
Reference Checkout Flow ..... 207
Checkout Steps Configuration Wizard ..... 208
Modifying the Reference Checkout Configuration ..... 210
Setting Up a One-Page Checkout Experience ..... 214
Multiple Ship To ..... 217
User Experience ..... 217
Setup ..... 218
Feature Architecture ..... 218
Known Limitations ..... 220
Reference Checkout Template Reference ..... 221
Sign In ..... 221
Shipping Page ..... 223
Payment Page ..... 226
Review & Place Order ..... 228

# SuiteCommerce Advanced Overview

**Important:** This document describes all versions of SuiteCommerce Advanced with Reference Implementations prior to the SuiteCommerce Advanced Denali release only. That includes **ShopFlow 1.06, My Account 1.05, and Checkout 2.04** or earlier. If your site is implemented on SCA Denali or later, see the help topic SuiteCommerce Advanced.

With SuiteCommerce Advanced, you can use technologies such as Ajax, jQuery, and JSON, together with NetSuite proprietary technologies such as SSP applications, Commerce API and Item Search API to create a custom e-commerce solution.

SuiteCommerce Advanced offers a different approach to web site customization from Site Builder. On Site Builder, web site pages are generated server-side. Web Site Tags can be used to pull site theme HTML or item attributes such as name, price, and product description for inclusion in the web site page.

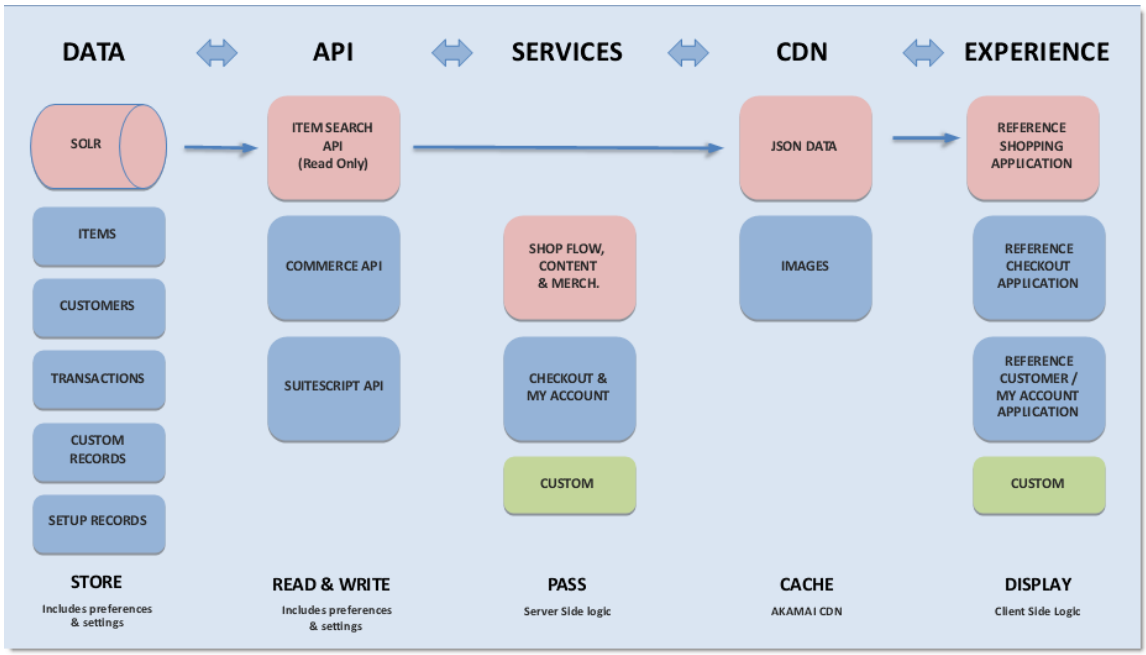
SuiteCommerce Advanced uses a services-based architecture, where the page is essentially static. Item attributes are pulled from a search service, and then output as JSON. Web developers can use the Item Search API, HTML markup and JavaScript to create an e-commerce web site hosted in the file cabinet.

## Using SuiteCommerce Advanced

To begin using SuiteCommerce Advanced, contact your NetSuite account representative for provisioning. When working with a Web Store implemented on pre-Denali releases of SuiteCommerce Advanced, in addition to using this guide refer to the following topics for information on setting up and managing your site:

- Creating a Website
- Item Configuration for Web Stores
- SCA Item Search Settings
- Web Site Setup for Domains
- Selecting Supported Touch Points
- SCA Product Images
- SCA Web Site Preferences
- Configure CDN Caching in SuiteCommerce Advanced
- Item Search API

The graphic shows each layer of functionality included with SuiteCommerce Advanced:




## Data

The **Data** layer in the graphic above, shows the NetSuite records you can use with SuiteCommerce Advanced. Solr is the search server used to index item record information. To learn more about the Solr search index process, see the help topic The Item Search Index Process.

For more information about preparing item records for Solr indexing, see the help topic Item Configuration for Web Stores.

With SuiteCommerce Advanced the Web Site Setup page includes preferences for configuring the Item Search API, and tools for setting up custom shopping domains. A tool for image resizing with SuiteCommerce Advanced is also included. For more information, refer to the following sections:

- SCA Item Search Settings
- Web Site Setup for Domains
- SCA Product Images

## API

The SuiteCloud platform technologies listed below are intended for web developers who want to extend the NetSuite web site capabilities to deliver an innovative user experience and additional functionality:

- SSP applications
- Commerce APIs
- Item Search APIs

SuiteScript Server Pages (SSP) is similar to JSP, PHP, or ASP with the added capability of accessing NetSuite data using SuiteScript, web site objects with the Commerce APIs, and item information using the Item Search API. Touch points on SSP applications determine where the web site visitor interacts with SSP application pages. For more information, see the help topic SSP Application Overview.

For more information about the Commerce API, see the help topic Commerce API Overview.

Web and mobile app developers can use the Item Search API, included with SuiteCommerce Advanced, to access item attributes. This is a REST API with JSON data objects . The Item Search API supports product presentation, faceted navigation, and item search. For more information, see the help topic Item Search API Overview.

## Services

SuiteCommerce Advanced supports SuiteApps (or bundles) built on the platform, and available from NetSuite.

Web site administrators and developers can use SuiteCommerce **Services** for applying business logic to e-commerce web sites. Service files work in the browser making backend calls to access data stored in your NetSuite account. Services for managing content delivery and product merchandising are delivered in SuiteApps you can install. For more information, see SuiteCommerce Services.

## CDN

The **CDN** enables a fast, smart e-commerce site by caching JSON data and image data sent by the Content Delivery and Product Merchandising services. For more information, see the help topic Configure CDN Caching in SuiteCommerce Advanced.

## Experience

You can choose one of the Reference Implementations to use as a template for creating a unique e-commerce shopping **experience**. Because services and SSP applications are built on the NetSuite platform, web developers can install a Reference Implementation, and then extend it to create their own custom solutions.

A Reference Implementation is a documented and supported SSP application delivered in a SuiteApp that includes a focused set of core features and functionality defining a standard best-in-class starting point for a an e-commerce web site. In addition, the application provides a platform that enables customers, developers, partners and agencies to extend and customize the experience to fit their individual goals and needs.

For more information, see Reference Implementations Overview.

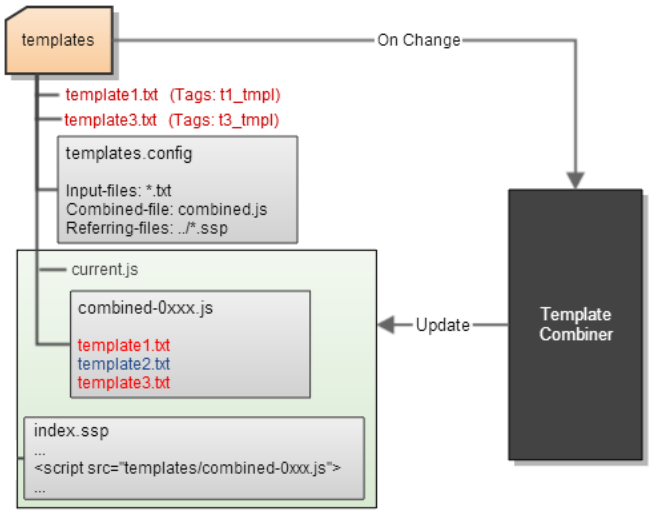
# Working with a SuiteCommerce Advanced Web Site

This section provides detailed information about functionality included with SuiteCommerce Advanced. Use this information to help you effectively maintain your web site and make changes.

- Understanding the SuiteCommerce Advanced Template Combiner
- Using Precedence Overlays for Web Site Customization
- Derivative Hosting with SuiteCommerce Advanced

## Understanding the SuiteCommerce Advanced Template Combiner

The template combiner is a file cabinet mechanism related to SSP applications and SuiteCommerce Advanced site templates that is not visible in the UI, but plays an important role in SuiteCommerce Advanced web site development.



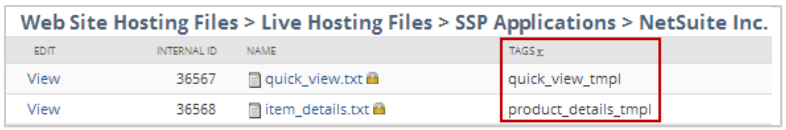

The template combiner preserves data stored in Tags when you work with template files installed by reference SuiteApps, and when you develop web sites using SuiteCommerce Advanced. For more information, see Preserving Metadata.

For more information about the configuration files associated with the template combiner is also important in terms of the reference SuiteApps (ShopFlow and My Account). For more information, see Working with Combination Files.

### Preserving Metadata

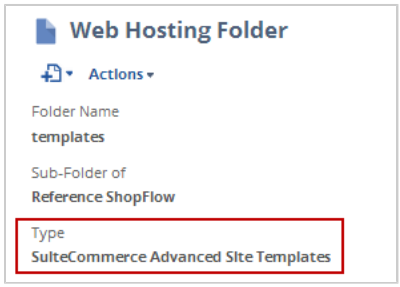
You can use SuiteCommerce Advanced to extend NetSuite's e-commerce reference implementations and to create your own e-commerce solutions for distribution to other NetSuite accounts. To this end,

the template combiner preserves metadata used in site templates. For example, note the templates installed by the Reference ShopFlow SuiteApp.




Template files installed by reference SuiteApps include metadata displayed in the **Tag** field on file cabinet records.

If you choose to create a web site for distribution to another NetSuite account, note that metadata (the value in the Tag field) is stored in the *manifest.txt* file. The *manifest.txt* file is generated automatically when you download a folder that contains SuiteCommerce Advanced site templates or any folder that has the Type set to **SuiteCommerce Advanced Site Templates**.




When the ZIP file is uploaded in the target NetSuite account, the *manifest.txt* file can be found in the root of the ZIP file. The manifest file can also be uploaded directly to SuiteCommerce Advanced Site Templates folder

Note that the manifest file must include a defined structure for Tags to be imported successfully.

Metadata in the manifest file is processed when you:

- add *manifest.txt* to the file cabinet.
- upload a ZIP file that includes *manifest.txt* using the Advanced Add button, and then choose the Unzip option.

> **Note:** When modifying the manifest.txt file, ensure that a closing "/" is used in folder paths. For example, `directory1/directory2/ foldertype="site_templates"`. Also, folders defined in the manifest.txt file must contain at least one file.

# Web Site Customization with SuiteCommerce Advanced

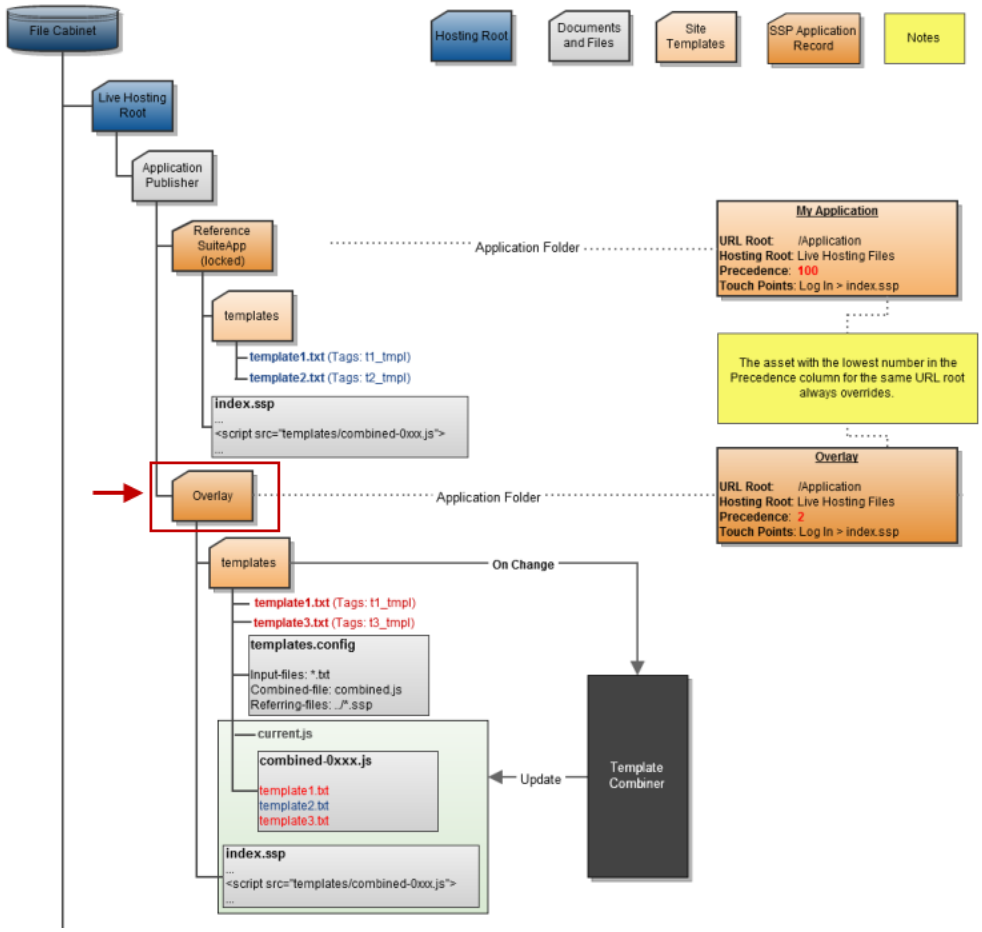
Using an overlay is the recommended approach for making changes to templates in a reference implementation. You can also use overlays with SSP applications that you create. For more information, see Using Precedence Overlays for Web Site Customization.

To make deeper changes, such as changes to SSP applications, scripts, or if you are redesigning your entire web site, use the branch hosting capabilities available with SuiteCommerce Advanced. For more information, see Derivative Hosting with SuiteCommerce Advanced.

# Using Precedence Overlays for Web Site Customization

You can use a precedence overly to make a change to the look-and-feel or functionality of your site. This approach begins with creating an empty SSP application that shares a URL root and hosting root with the application you want to customize.

You can use a precedence overlay to make a change to a locked SSP application. The diagram below shows three template files in a locked SSP application. A precedence overlay is used to modify *template1.txt*, and add *template3.txt*.

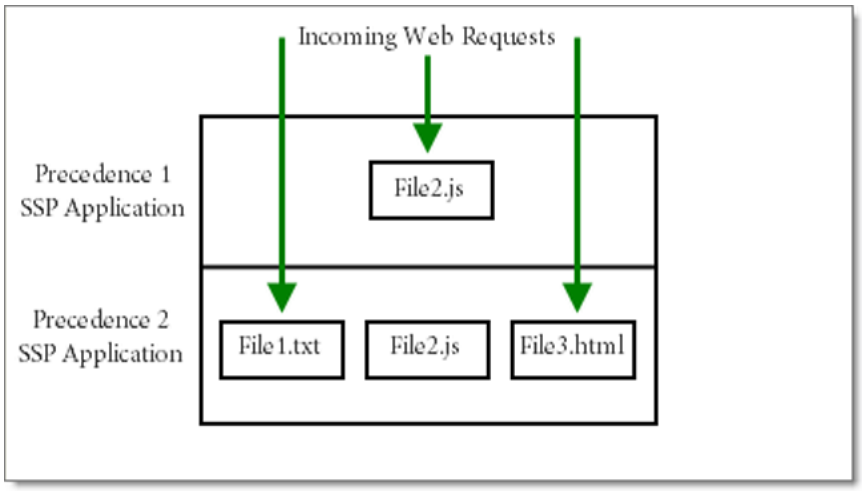



After you set SSP application touch points on a domain, the changes you make to files in the overlay affect the site immediately. In this way you are changing the site for everyone viewing the web site when the SSP application is attached to a live domain.

## Modifying Locked Files in SSP Applications

When SSP applications in your account share a URL root and hosting root, the file in the SSP application folder with the lowest number in the Precedence column will be served to incoming web requests.

For more information about SSP application precedence see the help topic Changing SSP Application Precedence.

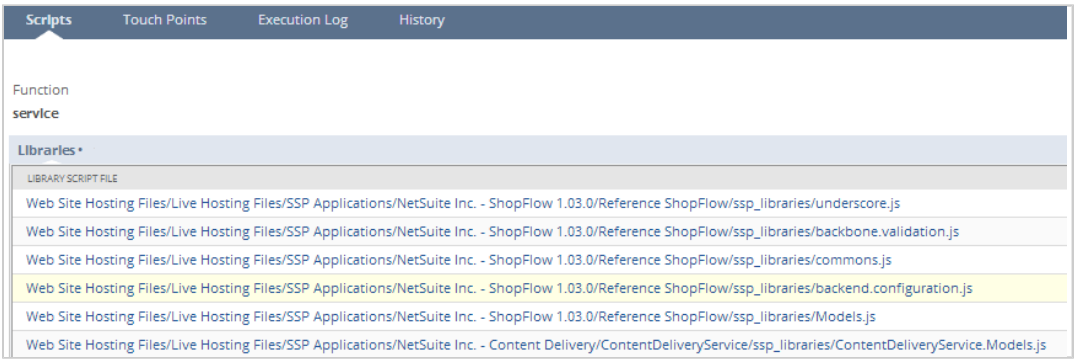



## To create a precedence overlay:

1. Copy the files you want to change from the original application folder into a new application folder with the same URL root. If you have installed a reference implementation, such as ShopFlow or My Account, use the Custom application folder installed by the SuiteApp.

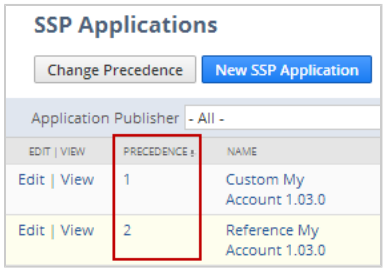
   For more detailed information on using precedence overlays for Reference Implementations see Using the Custom Folder.

2. Associate the proper library scripts with your customized SSP application.




3. Note that a system of precedence determines the SSP application folder serving assets to your site.

   For example, in the figure below, assets in the **Custom** folder are served first. For an explanation of how precedence works, see



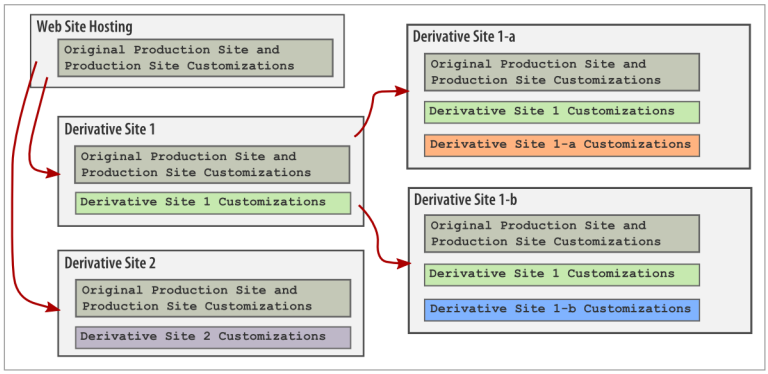

For more information about making changes to a reference implementation, see Customizing Implementations.

# Derivative Hosting with SuiteCommerce Advanced

Web developers can use derivative hosting to customize a website instead of using a precedence overlay. Use an overlay to make changes to the site that display immediately. Use a derivative hosting root to create another site based on the original, and to make and preview changes without affecting the original site. Files in the derivative hosting root override files from the parent hosting root while maintaining operation of the original site.

For example, to design seasonal themes for a web site, a web developer can create multiple derivative hosting root folders, one for each season. Each hosting root contains the minimum required set of files from the Reference Implementation plus copies of the files that need to change. The web developer can point a test domain (summer.test.com, spring.test.com) at each derivative hosting root. You can then preview each seasonal theme by navigating to each test domain. When it is time to go live with a new theme, the hosting root of the live domain can be changed to the appropriate derivative hosting root. The seasonal theme is then viewable on the site.

By using a derivative hosting root, a web developer can have multiple sites that operate from the same base code, but each site can be updated independently. Additionally, you can cascade code through multiple variations of a site by creating derivatives of a derivative. This is a powerful way to maintain multiple versions of a site while minimizing replication of code. The following image shows how code changes in the original site propagate to all of the derivative sites. Changes specific to a derivative site are not reflected in the original site.




## Setting Up Derivative Hosting with SuiteCommerce Advanced

When a web developer sets up a derivative hosting root, assets are served in order of precedence from both the derivative hosting root and the parent hosting root. The parent hosting root contains the SSP application intended for customization. The derivative hosting root contains the SSP application under development. Changes to the site can only be viewed on a domain pointed to the derivative hosting root folder.

The table below describes each task required to set up derivative hosting for web site development.

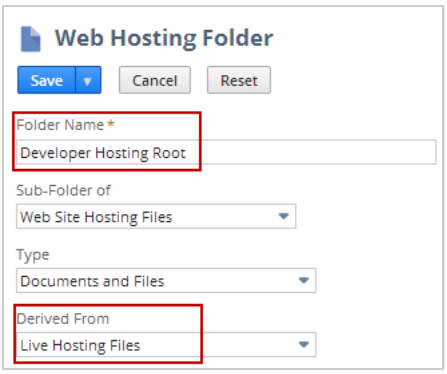
| Task | Before you start | Goal |
|---|---|---|
| Step 1: Setting Up a Derivative Hosting Root Folder S. | You should have a functional web site built using SuiteCommerce Advanced. | To set up a web site hosting folder that contains an SSP application and associated source files for extending or customizing an existing site. |
| Step 2: Creating an SSP Application for Derivative Hosting. | Make sure the target folder for your SSP application is derived from your production site. | To create a new SSP application for linking asset files. |
| Step 3: Copying Required Assets to the Derived Hosting Root. | Make sure the SSP application in the derivative hosting folder has the same URL root as the parent SSP application. | To copy required source files and assets into the derivative hosting root folder. |
| Step 4: Configuring the SSP Application. | Make sure that Step 2 above is complete before attempting to configure the SSP application. | To attach required JavaScript libraries and set touch points. |
| Step 5: Creating a Domain for Viewing a Site Under Development | Determine whether you want to setup a live domain, or you want to modify your local hosts file to use a domain that only exists on your computer. | To view the new site as it is being developed. |
| Step 6: Pointing a Domain to the Derivative Hosting Root | Domain is set up in hosts file on your computer or configure DNS with your domain provider. | To view the site under development by navigating to a domain. |
| Step 7: Copying Files for Customization of the Derivative Site. | Note that other than the required assets, you only need to copy the files you intend to change into the derived SSP application folder. | To customize an existing site. |

## Setting Up a Derivative Hosting Root Folder

The first step in setting up derivative hosting in SuiteCommerce Advanced, is to setup a derivative hosting root folder. You must create a new hosting files folder derived from the hosting folder that serves assets to your production web site.

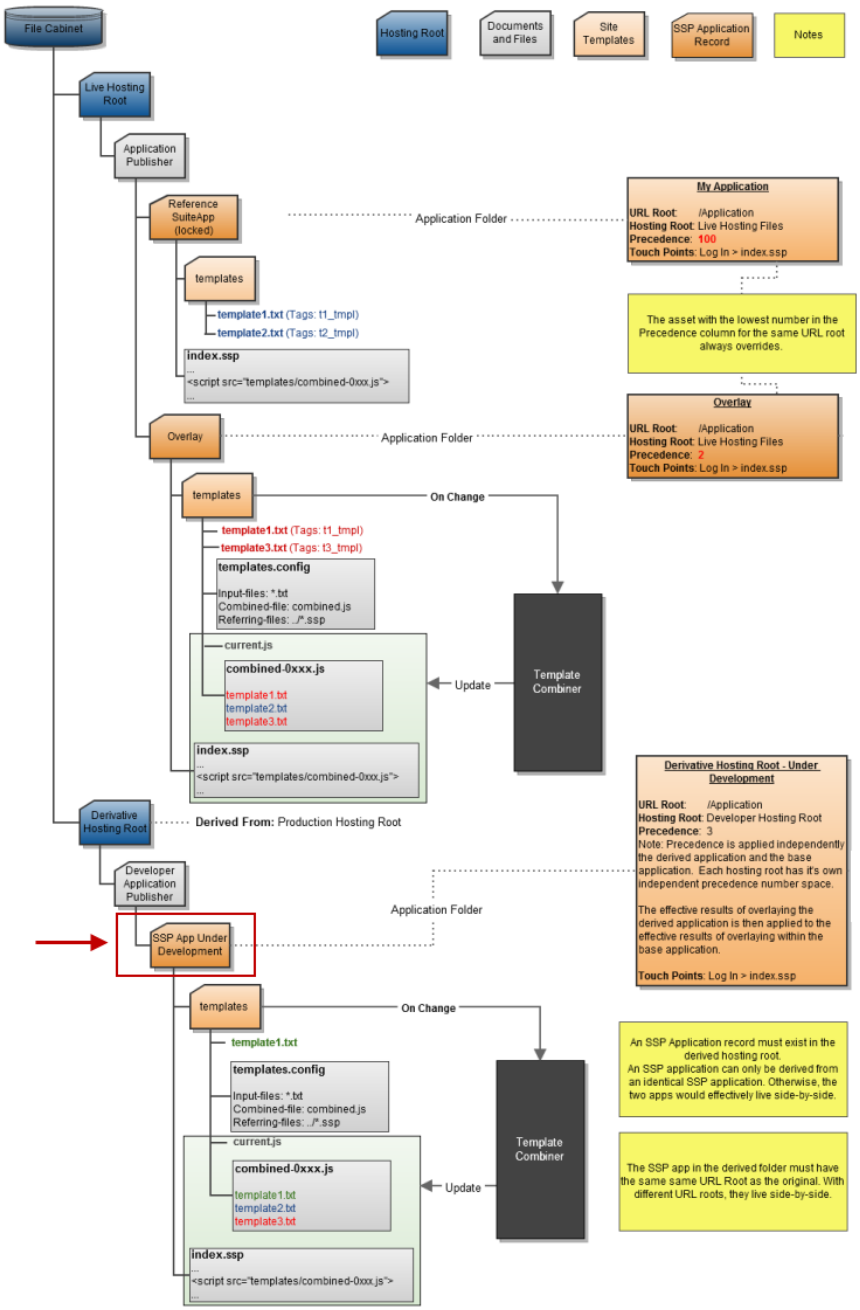
### To set up a derivative hosting root in SuiteCommerce Advanced:

1. Go to Setup > SuiteCommerce Advanced > SuiteCommerce Advanced > Web Site Hosting Files.
2. Click **New Folder**.
3. Enter a name for the folder, and select the hosting folder your derivative hosting root is derived from.



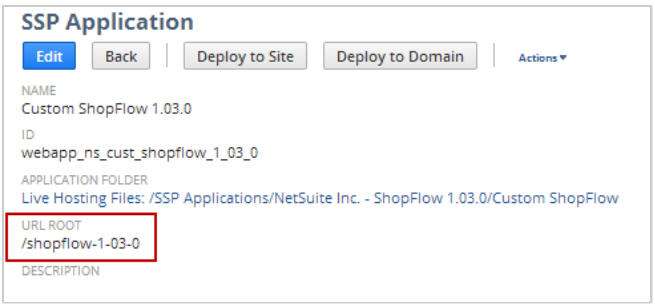

## Creating an SSP Application for Derivative Hosting

When creating a Derivative hosting site, an SSP Application folder is needed to store minimum set of required assets.




## To create an SSP application for development in a derivative hosting root:

1. Create a new SSP application. Go to Web Site hosting Files > Developer hosting root.
2. Click **New SSP application.**
    1. Enter a name for the publisher.
    2. Enter a name for the application.
    3. Click **Continue**.
3. Enter a value for the URL Root that is the same as the SSP application in the parent hosting root folder.




To make any changes to an SSP application (modifying templates for example) a copy of the SSP application must exist in the derivative hosting folder.

## Copying Required Assets to the Derived Hosting Root

After you create an SSP application record in the derived hosting folder, copy the required assets from the corresponding Custom Application folder to the derived hosting folder you created. The minimum set of files required reflect the same set of files that are shipped in the Custom folder for the Reference application. This includes any folders that contain combiner files, the SSP application files, and any additional files needed for your customization.

> **Note:** Depending on work already completed by your organization, the Custom folder may contain folders and files in addition to the minimum set shipped with the product. For the derived hosting site, only copy the minimum required set. Customization work completed in the Custom folder will automatically propagate to the derived hosting site.

For example, if you want to create a spring theme version of the Reference ShopFlow application, you might have created the following structure for the derived hosting root, *Spring Season : Spring Application Publisher : Spring ShopFlow*.

### To copy required assets:

1. Go to Live Site > SSP Applications > NetSuite Inc. > [Custom Folder of the Implementation].
2. Copy the following SSP applications to your derivative hosting folder:
   - index.ssp
   - sc.environment.ssp
   - sc.user.environment.ssp

   > **Note:** By default, the SSP files above are not in the Custom folder for a Reference Implementation. They are normally copied there when initial customization is completed. If they are not in the Custom folder you will need to copy these files from the Reference folder.
3. Copy all folders that include combiner files to your derivative hosting folder:
   - js
   - skins
   - templates

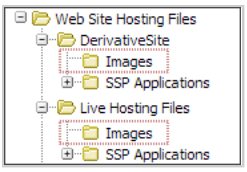
> **Important:** Copy the SSP application files over FIRST as described above. This ensures that when the combiner files run, the SSP application is available to update.

4. Copy the images and languages folder to your derivative hosting folder:
   - img
   - languages

   > **Note:** If the Custom Folder does not contain copies of the image and language files within these folders, you'll need to copy these folders directly from the Reference Folder.

5. Copy the Image Folder that contains your product image assets to the derivative root folder.

   This is the Image Folder as defined at Setup > SuiteCommerce Advanced > Set Up Web Site on the Advanced tab. The folder will be in your Web Site Hosting folder.




## Configuring the SSP Application

To view your site online, you must attach the JavaScript libraries to your SSP application and set a touchpoint.

### To configure the SSP application:

1. Go to Setup > SuiteCommerce Advanced > SSP Applications.
2. Click **Edit** next to the SSP Application in the derived hosting root.

   This is the derived hosting application you created earlier. Notice the **Libraries** subtab under the **Scripts** tab.
3. In the **Libraries** subtab, add the following JavaScript files from the Reference folder of the implementation.
   - /underscore.js
   - /backbone.validation.js
   - /commons.js
   - /backend.configuration.js
   - /Models.js

   > **Note:** If these libraries have been customized and exist in the Custom folder of the implementation you should add them from there to propagate all customizations to your derived hosting site. Also, ensure that the list of JavaScript files are listed on the SSP Application in the derived hosting root in the **exact** same order as they are on the original SSP application.

4. In the **Touch Points** tab define the Touch Points.
   1. Select **View Homepage**.
   2. Select /index.ssp for the Entry page.

5. Click **Save**.

## Creating a Domain for Viewing a Site Under Development

To view a site that is served from a derivative hosting folder, you must use a domain name. The process of setting up a domain name for viewing a web site is done outside of the NetSuite application either by modifying the hosts file on your computer, or by contacting your domain provider to set up a wildcard DNS record.

You have two options for creating a domain for viewing a site that is under development:

- Modifying the Hosts File on your Computer
- Creating a Live Domain Using a Wildcard DNS Record

### Modifying the Hosts File on your Computer

The *hosts* file, included with the Operating System (OS) on your computer, allows you to create mappings between domain names and IP addresses. In this way, you can navigate to a domain mapped to a NetSuite shopping server, without involving any changes to the DNS setup with your domain provider.

> **Note:** When you modify the hosts file on your computer, the domain is only valid on your computer. Anyone else who wants to view the site must modify the hosts file on their own computer to map the IP to the domain you set up.

For more information, follow the steps for *Editing the Hosts File*.

### Creating a Live Domain Using a Wildcard DNS Record

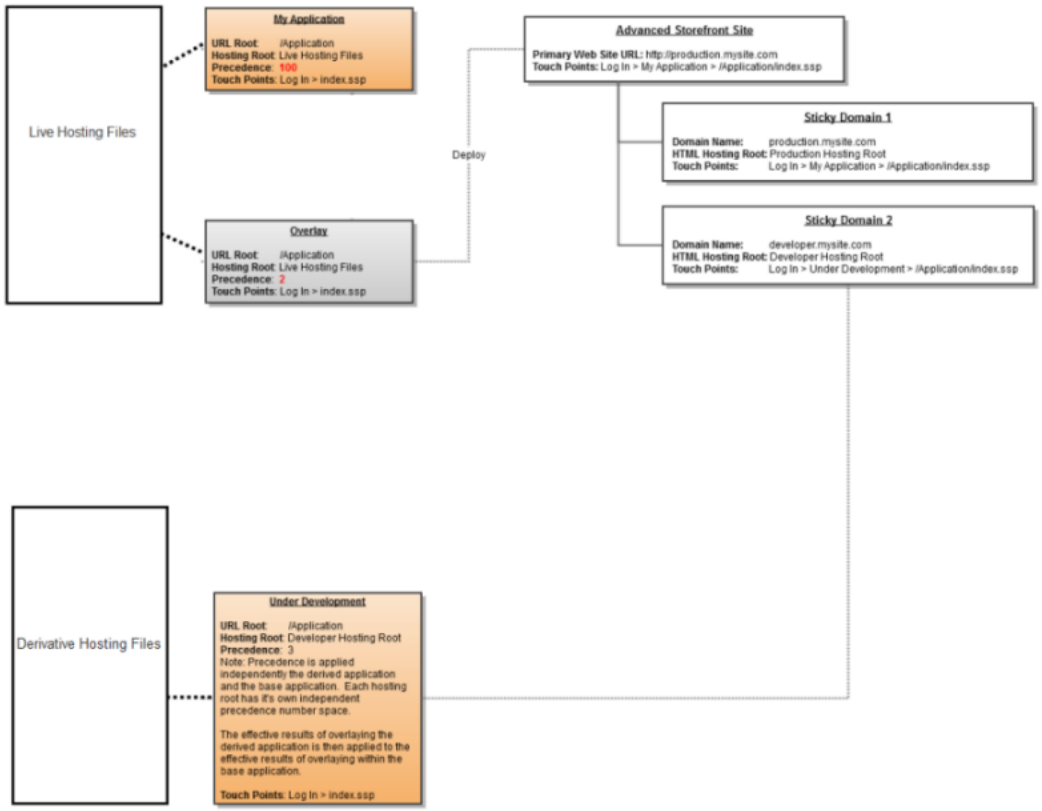
A wildcard DNS record will allow any name to be placed in front of a domain name.

> **Note:** For example, you can create a number of subdomain names on your domain name, such as *name1.mysite.com, name2.mysite.com*, and *name3.mysite.com*. Anyone who wants to view the site, can visit the domain you tell them, without any configuration on their computer.

You must contact your domain provider to set this up, as there may be some cost and configuration changes involved with creating multiple subdomains.

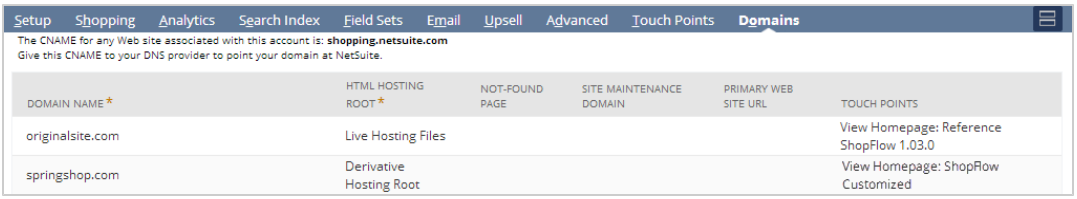
## Pointing a Domain to the Derivative Hosting Root

Once you have a domain pointing at the derivative hosting root, you can view the site under development. Visit the domain to view the output of combined files from the site under development and files from the live hosting root that may have been modified using a precedence overlay.




## To point a domain to the derivative hosting root:

1. On the Web Site Setup page, click the **Domains** subtab.
2. In the **Domain Name** field, enter the domain name you created for viewing the web site under development.
3. In the **HTML Hosting Root** field, select the hosting folder that is derived from the hosting folder you use for your production site.
4. In the **Touch Points** field select the touch points where the site will interact with the SSP application.




> **Note:** To use touch points, a copy of the SSP application must exist in the derivative hosting folder.

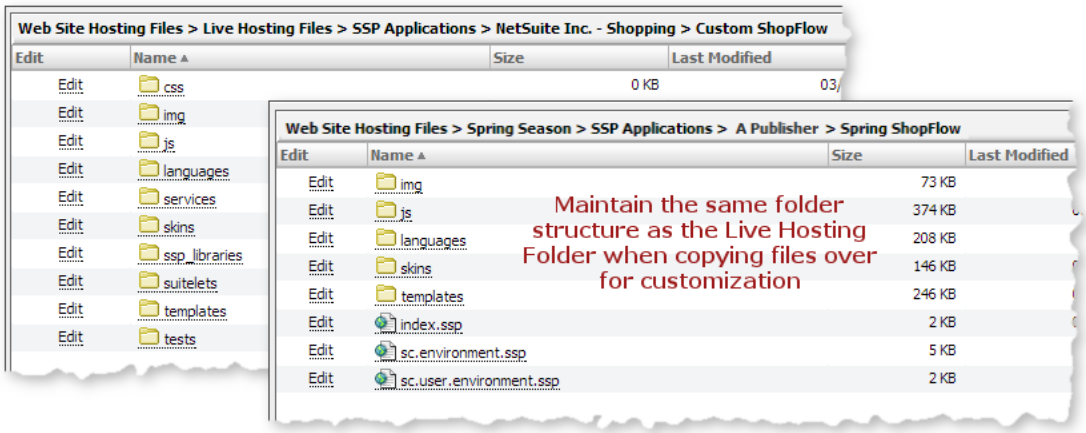
5. Click **Save**.

View the web site under development by launching a browser, and then visiting the domain you set up for previewing the site.

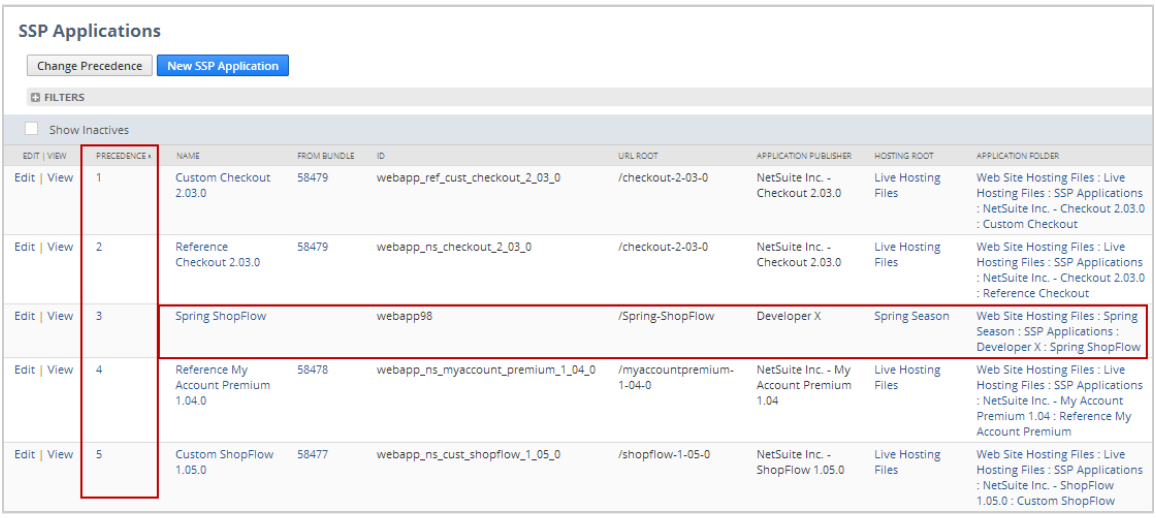
## Copying Files for Customization of the Derivative Site

To customize an SSP application included in a reference implementation, you must copy each of the files you intend to modify from the reference implementation into the SSP application folder you created in the derived hosting root.

The folder structure you use for your derived site (where you will make your changes) must be the same as the folder structure for the SSP application in the Live Hosting Files folder.




**Note:** A file in a derived hosting root folder always takes precedence over the same file in the parent hosting root. Each hosting root has its own independent precedence number space. So, if you had already used a precedence overlay to modify a file in the parent hosting folder, that modification is considered in the output from the site in the derived hosting root.




| EDIT \| VIEW | PRECEDENCE | NAME | FROM BUNDLE | ID | URL ROOT | APPLICATION PUBLISHER | HOSTING ROOT | APPLICATION FOLDER |
|---|---|---|---|---|---|---|---|---|
| Edit \| View | 1 | Custom Checkout 2.03.0 | 58479 | webapp_ref_cust_checkout_2_03_0 | /checkout-2-03-0 | NetSuite Inc. - Checkout 2.03.0 | Live Hosting Files | Web Site Hosting Files : Live Hosting Files : SSP Applications : NetSuite Inc. - Checkout 2.03.0 : Custom Checkout |
| Edit \| View | 2 | Reference Checkout 2.03.0 | 58479 | webapp_ns_checkout_2_03_0 | /checkout-2-03-0 | NetSuite Inc. - Checkout 2.03.0 | Live Hosting Files | Web Site Hosting Files : Live Hosting Files : SSP Applications : NetSuite Inc. - Checkout 2.03.0 : Reference Checkout |
| Edit \| View | 3 | Spring ShopFlow | | webapp98 | /Spring-ShopFlow | Developer X | Spring Season | Web Site Hosting Files : Spring Season : SSP Applications : Developer X : Spring ShopFlow |
| Edit \| View | 4 | Reference My Account Premium 1.04.0 | 58478 | webapp_ns_myaccount_premium_1_04_0 | /myaccountpremium-1-04-0 | NetSuite Inc. - My Account Premium 1.04 | Live Hosting Files | Web Site Hosting Files : Live Hosting Files : SSP Applications : NetSuite Inc. - My Account Premium 1.04 : Reference My Account Premium |
| Edit \| View | 5 | Custom ShopFlow 1.05.0 | 58477 | webapp_ns_cust_shopflow_1_05_0 | /shopflow-1-05-0 | NetSuite Inc. - ShopFlow 1.05.0 | Live Hosting Files | Web Site Hosting Files : Live Hosting Files : SSP Applications : NetSuite Inc. - ShopFlow 1.05.0 : Custom ShopFlow |

After you make changes to files in the derived hosting root, precedence is applied independently in the SSP application in the derivative hosting root and the base SSP application (in the parent hosting folder).

For more information on precedence, see Using Precedence Overlays for Web Site Customization.

### To copy the files into the SSP application folder in the derived hosting root:

1. Navigate to the derivative hosting root you created in the file cabinet.

2. If the folders do not already exist. Create new folders for site assets.
3. Go to Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Shopping > Custom [Implementation].
4. Click **Copy Files**.
5. Check the box next to the files and folders you want to copy to the SSP application in the derivative hosting folder.

   > **Note:** If customization has not already been done for the file needed, that file may not reside in the Custom [Implementation] folder. You may need to copy files from the Reference [Implementation] folder.

6. Copy files to the target folders you created in the derived hosting root folder.

Note that you can use a precedence overlay to change a template file in the Live Hosting Files folder to update the original site. This action will also update all sites pointed at derivative hosting roots, based on the sequence of precedence between the derived site and the original site.

# Reference Implementations Overview

**Important:** When upgrading to a new bundle version, refer to the Denali Release of SuiteCommerce Advanced for information on enhancements and fixes in that bundle.

The reference implementations are a set of documented and supported SSP bundle applications that include core features defining a standard best-in-class starting point for an e-commerce website. NetSuite currently offers the following Reference Implementations to get you started using the SuiteCommerce Advanced feature:

- Reference ShopFlow
- Reference My Account
- Reference Checkout
- Content Delivery
- Product Merchandising

Each application provides source files and assets, so that web developers, partners and agencies can use features included with SuiteCommerce Advanced, and extend the reference implementations to fit their individual goals and needs. These applications are generally designed to work together to provide a complete website shopping experience. Reference ShopFlow provides a robust shopping frontend and leverages the Content Delivery and Product Merchandising SuiteApps to delivery custom content areas while Reference My Account provides the account details, order tracking, and preferences for logged in users.

Read the following sections to learn more about how to setup, install, and get started with Reference Implementations:

- Prerequisites for SuiteCommerce Reference Implementations
- Getting Started with the Reference Implementations
- Customizing Implementations
- Technology and Framework
- Google Analytics for Reference Implementations
- Handling Localization in Reference Implementations
- Reference SuiteApp FAQ's

## Prerequisites for SuiteCommerce Reference Implementations

To install SuiteCommerce Advanced reference implementations, you must have the following features enabled:

1. Click the SuiteCloud subtab, enable the following features:
   - Client SuiteScript
   - Server SuiteScript
   - SuiteScript Server Pages
2. Click the Web Presence subtab, enable the following features:
   - Web Site
   - Web Store

- Advanced Site Customization
- Descriptive URLs
- Host HTML Files
- SuiteCommerce Advanced

**Note:** Reference My Account and Reference Checkout can also be implemented on a Site Builder website. For these implementations, the SuiteCommerce Advanced feature is NOT required.

SuiteCommerce Reference Implementations were developed in such a way that you can install them, deploy the touch points, customize CSS files to match the look and feel of your website, and then start using them right away. Using SuiteCommerce Advanced, you can install a reference implementation, and then extend the functionality to meet your company's specific ecommerce needs.

**Important:** Some implementations require additional customization before the site is functional. For example, Reference ShopFlow requires set up of specific field sets and Item fields. See the section specific to the implementation you are interested in for additional setup requirements.

# Getting Started with the Reference Implementations

The following information is common to all reference implementations:

- Managing Reference Implementation Installations
- Minimum Setup Requirements
- Using a Setup Script to Define Field Sets
- Touch Points Available for Reference Implementations
- Working with Combination Files
- Configuring Backend Objects Using the backend.configuration.js File
- Customizing Implementations
- Technology and Framework
- SuiteCommerce Reference Implementation Web Assets
- Google Analytics for Reference Implementations
- Handling Localization in Reference Implementations
- Reference SuiteApp FAQ's

Managing Content and Product Merchandising:

- Content Delivery
- Product Merchandising

## Managing Reference Implementation Installations

See the following sections for instructions on managing your **Reference Implementation** bundles. For more general information on working with bundles, see the help topic SuiteApp Installation and Update.

- Understanding Reference Implementation Versioning

- Installing New Reference Implementations
- Testing Reference Implementation Installations
- Updating Reference Implementations
- Uninstalling Reference Implementations

## Understanding Reference Implementation Versioning

**Important:** When upgrading to a new bundle version, refer to the Denali Release of SuiteCommerce Advanced for information on enhancements and fixes in that bundle.

Reference Implementation bundles are generally available as unique bundles, with **unique bundle IDs**, unless explicitly stated otherwise. Each bundle has a version number appended to the folder name. You can install multiple versions of the same bundle into your account.

There are three different types of Reference Implementation bundle updates that can occur. Bundle updates use the following naming conventions:

- **X.01.0**: These updates include major updates, new features, and enhancements. These bundles have a unique IDs and generally correspond with other NetSuite core releases.
- **1.0X.0**: These updates include bug fixes and small enhancements. These bundles will have a unique ID and be released on a frequent basis.
- **1.01.X**: These updates include critical bug fixes. A new bundle ID is not created for these fixes. Upgrading existing bundles with these releases should have little to no impact on existing customizations.

**Important:** Reference Implementation bundles are **unmanaged**. With unmanaged bundles, updates are not automatically pushed to installed bundles. You must explicitly update each bundle to receive the latest changes to that bundle. This is important to ensure that existing implementations with customizations are not impacted unexpectedly by an update.

### Reference My Account Editions

Currently there are two available editions for Reference My Account:

- **Reference My Account**: This edition provides all of the core functionality needed for a complete B2C experience and is available for all Site Builder subscriptions. The same functionality is available for SuiteCommerce Advanced subscriptions using the Premium edition.

  **Important:** For SuiteCommerce Advanced subscriptions you **must** use the Reference My Account Premium edition.

- **Reference My Account Premium**: This edition provides all of the functionality offered in Reference My Account plus additional advanced features for an enhanced B2B experience. This edition is available by default with all SuiteCommerce Advanced subscriptions. It is also available for Site Builder subscriptions for an additional fee.

The following table outlines the feature set available with each edition.

| Feature | Reference My Account | Reference My Account Premium |
|---|---|---|
| Logon/Signup/Forgot Password | X | X |
| Profile Information | X | X |
| Email Preferences | | |

| Feature | Reference My Account | Reference My Account Premium |
|---|---|---|
| Address Book Management<br>Credit Card Management | | |
| Email Preferences | X | X |
| Address Book Management | X | X |
| Credit Cards Management | X | X |
| Order History & Order Details | X | X |
| Reorder Items | X | X |
| Returns | X | X |
| Receipt History & Details | X | X |
| Case Management | X | X |
| Product Lists | | X |
| Quotes | | X |
| Billing | | X |
| Account Balance<br>Print A Statement<br>Make A Payment<br>Transaction History & Details<br>Credit Memos/Deposits | | |

## Installing New Reference Implementations

When installing a Reference Implementation for the first time you must install the most recent version of the bundle.

If you have previously installed a Reference Implementation see Testing Reference Implementation Installations for information on installing a new version.

### To install a new Reference Implementation:

1. Click Customization > SiteBundler > Search & Install Bundles.
2. In the **Keyword** field, enter the Bundle ID of the Reference Implementation you want to install.

   **Important:** There is a separate Bundle ID for each Reference Implementation. For **shared** bundles, contact your NetSuite Representative if you do not have access to the bundle.

| Reference Implementation | Bundle ID |
|---|---|
| Reference ShopFlow 1.07 | 96113 (this is a **shared** bundle) |
| Reference My Account 1.06 | 96209<br>**Note:** For more information on the Reference My Account Editions, see Reference My Account Editions. |
| Reference My Account Premium 1.05 | 96162(this is a **shared** bundle) |

| Reference Implementation | Bundle ID |
|---|---|
| | **Note:** For more information on the Reference My Account Editions, see Reference My Account Editions. |
| Reference Checkout 2.04 | 96111 |
| Content Delivery | 33437 |
| Product Merchandising | 35075 |
| Reference Cart & One Page Checkout | 3519604<br>**Note:** This is a **shared** bundle and will be deprecated. If you are implementing a new Checkout process for your store, use the latest Reference Checkout implementation. |
| Reference Product Lists Records | 53051<br>This bundle is required when using Product Lists with Reference ShopFlow and Reference My Account. |
| Reference Product Review Records | 53053<br>This bundle is required when using Product Reviews with Reference ShopFlow. |

3. Click **Search**.
4. Verify that the correct Reference Implementation is returned in the search and select it.
5. Review the Reference Implementation details and then click **Install**.
6. Deploy the Reference Implementation to touch points.

> **Note:** If this is an upgrade, do not deploy touch points to your production site until the new installation has been thoroughly tested. For details, see Testing Reference Implementation Installations.

If you are installing a new bundle, after installing the bundle, deploy the Reference Implementation to touch points. Note that the Reference Implementations do not operate in a site until you deploy to its Touch Point. You can set multiple Touch Points for a certain reference implementation and you can set touch points for each implementation for multiple websites. See Minimum Setup Requirements for details on the minimum Touch Point required.

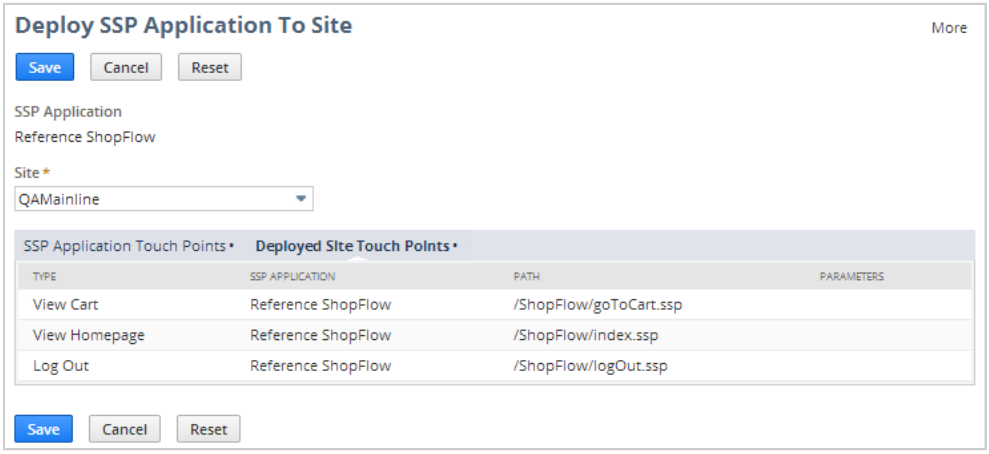
> **Note:** Content Delivery and Product Merchandising bundles do NOT need to be deployed to a touch point.

> **Important:** When using **Reference Checkout** with a Site Builder website, you must use the implementation for the following three checkout touch points: **Log In**, **Register**, and **Proceed to Checkout**. You can use the implementation for the **View Cart** touch point. When using Reference Checkout with a SuiteCommerce Advanced Site, you should use Reference ShopFlow for the View Cart touch point.

1. Go to Setup > SuiteCommerce Advanced > SSP Applications.
2. Click **View** next to the desired Bundle to see the SSP Application record.
3. On the **SSP Application Touch Points** subtab, define the touch point for the currently selected Reference Implementation.

4. Click **Deploy to Site**.
5. On the Deploy to SSP Application To Site page, select the site where you want to deploy the currently selected Reference Implementation. This setting is required.

   On the **SSP Application Touch Points** subtab, verify the **Deploy** box is marked. On the **Deployed Site Touch Points** subtab, review all SSP applications that are deployed to touch points on your website.




## Testing Reference Implementation Installations

Prior to configuring your production site domains to point to the touch points for a Reference Implementation, test the implementation.

The procedures for testing Reference Implementations are different for Reference ShopFlow versus the secure implementations, Reference My Account and Reference Checkout.

- Testing a Reference ShopFlow Implementation
- Testing a Checkout or MyAccount Reference Implementation

### Testing a Reference ShopFlow Implementation

For SuiteCommerce Advanced implementations using Reference ShopFlow, test using a subdomain. For example, if your production site is mystore.netsuite.com, create a dev.mystore.netsuite.com subdomain and configure that subdomain to point to the touch points of the Reference Implementation you installed. If this is a new implementation, complete your site customizations and test against the subdomain before deploying to the production site.

If this is an upgrade to an existing ShopFlow implementation, configure your new installation with the customizations already implemented and then test against the subdomain as described below.

> **Note:** Do not uninstall the old version of the Reference ShopFlow until you have thoroughly tested your site and have deployed the new touch points to your production site.

#### To test the Reference ShopFlow Implementation:

1. Create a subdomain for your website at Setup > SuiteCommerce Advanced > Domains. For more information, see the help topic Web Site Setup for Domains.

   Configure the domain touch points to point to the most recent version of the bundle. This subdomain is used to test your custom application before deploying to your production site.

You can test Reference ShopFlow on your website using the new version of the Reference Implementation without interrupting your production web site.

2. Verify that the Primary Web Site URL is not set.

   If the Primary Web Site URL is set, the subdomain is directed to the primary URL for your website. For more information, see the help topic Selecting a Primary Web Site URL.

3. Test the URL of the new subdomain to verify that the default Reference Implementation is working correctly.
4. Test Reference ShopFlow on your website with any previously implemented customizations.

   Copy the customizations from your existing Custom folder to the custom folder of the new Reference Implementation.

   > **Note:** Copy the files individually rather than copying the entire directory. The directory structure of the new version may be different than previous versions.

   Review the backend.configuration.js file to verify that objects only return fields that are used in the front-end website. For more information, see Configuring Backend Objects Using the backend.configuration.js File.

5. Test your custom application as necessary using the touch points.

   > **Note:** When testing your custom application, disable any analytics modules you have configured to avoid corrupting your analytics data with testing output.

6. After you have verified that your custom application is working correctly, deploy the Reference Implementation to the touch points of your production site. For details, see Step 6 in Installing New Reference Implementations.

## Testing a Checkout or MyAccount Reference Implementation

For both SuiteCommerce Advance and Site Builder sites, test a new Reference My Account or Reference Checkout installation by adding items to the shopping cart and performing the checkout process before deploying the new Reference Implementation to touch points. Test each application component by accessing the URL directly, instead of using touch points.

### To test a Checkout or MyAccount Reference Implementations

1. Add items to the shopping cart.
2. Log in to your website or create a new account.
3. Test the application by accessing the following URLs:

   > **Note:** You must browse each link within the application manually because touch points are not enabled. In the following examples, c.3571948 corresponds to your account number and checkout-2-02-0 corresponds to the new bundle folder installed in your account.

   > **Important:** When testing your application on a secure domain you must ensure that links do not redirect to the older version of the Reference Implementation. As you step through flows, double-check the URL in the browser.

   - **Login**: https://checkout.netsuite.com/c.3571948/checkout-2-02-0/login.ssp?n=8&#login-register
   - **Shopping Cart**: https://checkout.netsuite.com/c.3571948/checkout-2-02-0/cart.ssp?n=8#cart

- **First page of Checkout**: https://checkout.netsuite.com/c.3571948/checkout-2-02-0/index.ssp?n=8

4. Review the backend.configuration.js file to verify that objects only return fields that are used in the front-end website. For more information, see Configuring Backend Objects Using the backend.configuration.js File.
5. Deploy the new Reference Implementation to touch points. For details, see Step 6 in Installing New Reference Implementations.

## Updating Reference Implementations

**Important:** When updating to a new bundle version, refer to the Denali Release of SuiteCommerce Advanced for information on enhancements and fixes in that bundle.

When **upgrading** to a new release with a new bundle ID, install the new bundle and then merge existing customizations to the new bundle before moving your live site to the new bundle. For details, see Installing New Reference Implementations.

**Important:** Reference Implementation bundles are **unmanaged**. With unmanaged bundles, updates are not automatically pushed to installed bundles. You must explicitly update each bundle to receive the latest changes to that bundle. This is important to ensure that existing implementations with customizations are not impacted unexpectedly by an update.

### To update a Reference Implementation:

**Important:** The only case where you want to *update* an existing bundle is when there is a critical bug fix release (as noted in ). After installing a new bundle, keep the existing bundle in place until after you finish testing customizations on the new bundle.

1. Click Customization > SiteBundler > Search & Install Bundles, and then click **List**.

   A list of bundles currently installed is displayed. If an update is available, the Version number displays a blue exclamation mark icon next to it.
2. Click on the wrench icon next to the bundle you wish to update and select **Update**.
3. If updating a Reference Implementation, the Preview Bundle Update page is displayed which describes all scripts and files installed by the bundle. Click **Update Bundle** and then **OK** to continue.

   The update begins and the Installed Bundles page displays again with the status of the update. Click **Refresh** to update the status.

   **Important:** When updating or uninstalling a Reference Implementation bundle, any custom changes made in the Custom folder for the implementation are retained as well as any custom SSP applications you have created.

**Note:** After updating an existing ShopFlow Reference implementation you may experience caching issues that prevent the page from loading. Try navigating to the domain using the shopflow url, **http://[domain.com]/shopflow**.

## Uninstalling Reference Implementations

### To uninstall a Reference Implementation:

1. Click Customization > SiteBundler > Search & Install Bundles and then click **List**.

A list of bundles currently installed is displayed. If an update is available the Version number displays a blue exclamation mark icon next to it.

2. Click on the wrench icon next to the bundle you wish to uninstall and select **Uninstall**.
3. Click **OK** at the prompt. The Reference Implementation will be uninstalled. This may take a few moments.

   If you are uninstalling to upgrade from a BETA version of the bundle, do not remove your customization files. These files are retained during the uninstall process.

   When using a Reference Implementation, if you do not want to retain any customizations, delete the customized folders from the file cabinet. The names and locations of these folders may vary depending on your original installation and the Reference Implementation you are uninstalling. For example, delete all files and folders at Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - ShopFlow.

> **Important:** When uninstalling a Reference Implementation bundle, any custom changes made in the Custom folder for the implementation are retained as well as any custom SSP applications you have created.

## Minimum Setup Requirements

Following are the minimum requirements for a complete SuiteCommerce Advanced website using all four reference implementations together.

Set the following preferences:

- On the Setup tab, in the Home Page field, select Hosting Files : Live Hosting Files
- On the Shopping tab, you must select at least one currency and one language to be online and default.
- On the Search Index tab, under Facet Fields, select at least one facet selected to show facets in the front end (for example, Online Price)
- On the Touch Points tab, set the values as described in the following table.

| Name | SSP Application | Path |
|---|---|---|
| View Cart | Reference ShopFlow | goToCart.ssp |
| View Homepage | Reference ShopFlow | index.ssp |
| Log Out | Reference ShopFlow | logOut.ss |

- On the Domains tab, enter one domain. For more information, see the help topic Web Site Setup for Domains. At least one domain is required.
- On the Field Sets tab, select fieldsets needed as described in the following table. Fields not indicated as required are used inside the Reference ShopFlow as an example.

## Field Sets

**Note:** See Using a Setup Script to Define Field Sets for instructions on running a script to populate your website with these default Field Set requirements.

**Important:** The Field Set to return item details is named "details". If a different name is used, you **must** configure that change in the backend.configuration.js file as described in items_fields_advanced_name.

| Name | Id | Fields | Required/Details |
|---|---|---|---|
| **Details** | **details** | | |
| | | custitem_ns_pr_count: NS PR Count (Custom) | |
| | | custitem_ns_pr_item_attributes: NS PR Attributes (Custom) | |
| | | custitem_ns_pr_rating: NS PR Rating (Custom) | |
| | | custitem_ns_pr_rating_by_rate: NS PR Rating By Rate (Custom) | |
| | | defaultcategory_detail: Default Category (Detail) | |
| | | itemimages_detail: Item Images (Detail) | |
| | | itemoptions_detail: Item Options (Detail) | Required for matrix items |
| | | matrixchilditems_detail: Matrix Child Items (Detail) | Required for matrix items |
| | | onlinecustomerprice_detail: Price for Current Customer (Detail) | |
| | | pagetitle2: Page Title 2 | |
| | | pricelevel1: Retail | |
| | | pricelevel1_formatted: Retail (Formatted) | |
| | | quantityavailable: Available | |
| | | displayname: Display Name | |
| | | featureddescription: Featured Description | |
| | | internalid: Internal ID | Required for Shopping Cart and My Account |
| | | itemtype: Type | Required for Shopping Cart and My Account |
| | | itemid: Name | |

| Name | Id | Fields | Required/Details |
|---|---|---|---|
| | | outofstockmessage: Out Of Stock Message | |
| | | pagetitle: Page Title | |
| | | stockdescription: Stock Description | |
| | | storedetaileddescription: Detailed Description | |
| | | storedescription2: Store Description 2 | |
| | | storedisplayname2: Store Display Name 2 | |
| | | urlcomponent: URL Component | |
| | | isinactive: Inactive | Required for My Account |
| | | isonline: Available Online | Required for My Account |
| | | isinstock: In Stock | Required for Reference ShopFlow |
| | | ispurchasable: Available for Purchase | Required for Reference ShopFlow |
| | | isbackorderable: Available for Backorder | Required for Reference ShopFlow |
| | | showoutofstockmessage: Display Out-of-Stock Message | Required for Reference ShopFlow |
| | | minimumquantity: Minimum Quantity | Required if is set smallest quantity for allow customers to purchase for an item. |
| | | metataghtml: Meta Tag HTML | To be included in the document's head on product detailed page. |
| | | searchkeywords: Search Keywords | Coma separated words to be included in the meta keywords on the product detailed page. |
| **Matrix Child Items** | **matrixchilditems** | | |
| | | onlinecustomerprice_detail: Price for Current Customer (Detail) | |
| | | quantityavailable: Available | |
| | | internalid: Internal ID | Required for Shopping Cart and My Account |

| Name | Id | Fields | Required/Details |
|---|---|---|---|
| | | minimumquantity: Minimum Quantity | Required to allow customers to purchase for an item. |
| | | isinstock: In Stock | Required for Reference ShopFlow |
| | | ispurchasable: Available for Purchase | Required for Reference ShopFlow |
| | | isbackorderable: Available for Backorder | Required for Reference ShopFlow |
| | | itemid: Name | |
| **Search** | **search** | | Required for Reference ShopFlow |
| | | itemimages_detail: Item Images (Detail) | |
| | | itemoptions_detail: Item Options (Detail) | Required for matrix items |
| | | matrixchilditems_detail: Matrix Child Items (Detail) | Required for matrix items |
| | | onlinecustomerprice_detail: Price for Current Customer (Detail) | |
| | | pricelevel1: Retail | |
| | | pricelevel1_formatted: Retail (Formatted) | |
| | | displayname: Display Name | |
| | | internalid: Internal ID | Required for Shopping Cart and My Account |
| | | itemid: Name | Required for Shopping Cart and My Account |
| | | outofstockmessage: Out Of Stock Message | |
| | | stockdescription: Stock Description | |
| | | storedisplayname2: Store Display Name 2 | |
| | | urlcomponent: URL Component | |
| | | itemtype: Type | |
| | | isinstock: In Stock | Required for Reference ShopFlow |
| | | ispurchasable: Available for Purchase | Required for Reference ShopFlow |
| | | isbackorderable: Available for Backorder | Required for Reference ShopFlow |

| Name | Id | Fields | Required/Details |
|---|---|---|---|
| | | showoutofstockmessag e: Display Out-of-Stock Message | Required for Reference ShopFlow |
| | | custitem_ns_pr_count: Product Reviews Count (Custom) | |
| | | custitem_ns_pr_rating: Product Reviews Rating (Custom) | |
| **Order id** | order | | |
| | | itemimages_detail: Item Images (Detail) | |
| | | itemoptions_detail: Item Options (Detail) | |
| | | onlinecustomerprice_detail: Price for Current Customer (Detail) | |
| | | displayname: Display Name | |
| | | internalid: Internal ID | Required for Shopping Cart and My Account |
| | | itemid: Name | |
| | | storedisplayname2: Store Display Name 2 | |
| | | urlcomponent: URL Component | |
| | | outofstockmessage: Out Of Stock Message | |
| | | showoutofstockmessag e: Display Out-of-Stock Message | |
| | | isinstock: In Stock | Required for Reference MyAccount |
| | | isbackorderable: Available for Backorder | Required for Reference MyAccount |
| | | ispurchasable: Available for Purchase | Required for Reference MyAccount |
| | | pricelevel1: Retail | |
| | | pricelevel1_formatted: Retail (Formatted) | |
| | | stockdescription: Stock Description | |
| | | matrixchilditems_detail: Matrix Child Items (Detail) | Required for My Account Product Lists |

| Name | Id | Fields | Required/Details |
|---|---|---|---|
| | | itemtype: Type | Required for Shopping Cart and My Account |
| | | minimumquantity: Minimum Quantity | |
| | | isonline: Available Online | Required for Reference MyAccount |
| | | isinactive: Inactive | Required for Reference MyAccount |
| Type Ahead | typeahead | | |
| | | itemimages_detail: Item Images (Detail) | |
| | | displayname: Display Name | |
| | | itemid: Name | |
| | | custitem_ns_pr_rating: NS PR Rating (Custom) | Required for Product Reviews |
| | | storedisplayname2: Store Display Name 2 | |
| | | urlcomponent: URL Component | |
| Correlated Items Id | correlateditems | same as search | |
| Related Items Id | relateditems | same as search | |
| Correlated Items Details | correlateditems_details | correlateditems_detail: Correlated Items (Detail) | |
| Related Items Details Id | relateditems_details | relateditems_detail: Related Items (Detail) | |

## Reference ShopFlow SSP Application

Several JavaScript libraries must be referenced from your Reference ShopFlow application. These files integrate the Content Delivery Service bundle, Host Mapping for multiple languages and domains, and the required backbone, underscore, commons, and models JavaScript libraries.

### To verify that the required libraries are added to the application:

1. Edit the Reference ShopFlow Application.

   Go to Setup > SuiteCommerce Advanced > SSP Applications and click **Edit** next to the Reference ShopFlow application.

2. On the **Scripts** tab, verfiy the following references:

   **Note:** The order of these libraries **MUST** match the order listed below.

   - Web Site Hosting Files/Live Hosting Files/SSP Applications/SuiteCommerce/Reference ShopFlow /ssp_libraries/underscore-1.4.2.js

- Web Site Hosting Files/Live Hosting Files/SSP Applications/SuiteCommerce/Reference ShopFlow/ssp_libraries/backbone.validation-0.5.2.js
- Web Site Hosting Files/Live Hosting Files/SSP Applications/SuiteCommerce/ Reference ShopFlow/ssp_libraries/commons.js
- Web Site Hosting Files/Live Hosting Files/SSP Applications/SuiteCommerce/ Reference ShopFlow/ssp_libraries/backend.configuration.js
- Web Site Hosting Files/Live Hosting Files/SSP Applications/SuiteCommerce/Reference ShopFlow /ssp_libraries/Models.js
- Web Site Hosting Files/Live Hosting Files/SSP Applications/NetSuite Inc. - Content Delivery/ ContentDeliveryService/ssp_libraries/ContentDeliveryService.Models.js

  This script is not added during installation of Reference ShopFlow and must be added if you intend to use Content Delivery services. Before adding the script the Content Delivery bundle must be installed in your account. For more information on Content Delivery see Content Delivery.
- Web Site Hosting Files/Live Hosting Files/SSP Applications/NetSuite Inc. - Shopping/Reference ShopFlow/ssp_libraries/hosts.config.js

  This script is not added during installation of Reference ShopFlow and must be added if you intend to define custom host mappings for your website.

3. Click **Save**.

## Required Item Fields

Several fields are required to be set up for each item record before items are returned in the search results sets and displayed correctly in item lists and details. For detailed information on setting up item fields see the help topic Item Search Index.

## Using a Setup Script to Define Field Sets

Following are scripts you can leverage to quickly populate your site with required Field Sets. We recommend that you understand your site setup requirements fully before using this script. For details, refer to Minimum Setup Requirements. You can then modify the scripts to add Field Sets that are critical to your needs. For example, if your account does not define correlated or related items for a certain item, do not include the correlated and related items Field Sets. Your modified script is intended to be used during the initial development phase of your site so that you can quickly replicate a site setup as needed.

> **Note:** To execute these scripts, you need to use the SuiteScript Debugger tool. If you are unfamiliar with this tool, refer to SuiteScript Debugger for detailed instructions. To execute these scripts, you shouldn't need to know all of the debugging capabilities but it is a good idea to familiarize yourself with the tool.

### To use the script:

1. Create your new site, without adding any Field Sets, as described in Getting a Reference ShopFlow Site Running.
2. Using one of the scripts below as a template:
   - (Optionally) Add any Field Set definitions to the script below as needed for your site.

     The set included in these scripts are the required fields and normally provide a good starting point for your website.

> **Important:** The Field Set to return item details is named "details". If a different name is used, you **must** configure that change in the backend.configuration.js file as described in items_fields_advanced_name.

- Edit the siteID variable with the site id you want to use.

  The siteID is displayed in the URL when you navigate to the Web Site Setup page for the desired site. The default value of zero fails since no site is ever assigned an ID of zero.

3. Execute the script in the SuiteScript debugger.
   1. Go to Customization > Scripting > Script Debugger.
   2. Enter the SuiteScript Debugger domain (by clicking the link as indicated) and then re-enter your login credentials if prompted.
   3. Copy and paste the script code into the debugger.
   4. Click **Debug Script** and then hit the **Continue** button.

Once the script has executed, Field Sets are properly populated and can be verified in the Field Sets tab of the Web Site Setup record at Setup > SuiteCommerce Advanced > Set Up Web Site.

## Setup Script

There are two possible scripts you can run. The first, Standard Script, can be used for all sites. The second, Script with Product Reviews, should be executed as described when the product list records bundle is installed in your account **and** you need product reviews.

## Standard Script

```
var siteID = 99999;// change to the site id you want to use
var siteRecord = nlapiLoadRecord('website', siteID);
siteRecord.selectNewLineItem('fieldset');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetname', 'search');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetid', 'search');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetrecordtype', 'ITEM');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetfields', 'urlcomponent,itemimages_detai
l,itemoptions_detail,internalid,onlinecustomerprice,onlinecustomerprice_detail,onlinecustomerpr
ice_formatted,onlinematrixpricerange,onlinematrixpricerange_formatted,quantityavailable,display
name,itemid,outofstockbehavior,outofstockmessage,stockdescription,storedescription,storedisplay
thumbnail,storedisplayname2,isinstock,isbackorderable,ispurchasable,showoutofstockmessage');
siteRecord.commitLineItem('fieldset');
siteRecord.selectNewLineItem('fieldset');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetname', 'details');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetid', 'details');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetrecordtype', 'ITEM');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetfields', 'urlcomponent,itemimages_detai
l,itemoptions_detail,internalid,matrixchilditems_detail,onlinecustomerprice_detail,quantityavai
lable,displayname,itemtype,itemid,outofstockbehavior,outofstockmessage,pagetitle,rate,rate_form
atted,relateditemsdescription,stockdescription,storedetaileddescription,storedisplayimage,store
displayname2,isinstock,isbackorderable,ispurchasable,showoutofstockmessage,metataghtml,minimumq
uantity');
siteRecord.commitLineItem('fieldset');
siteRecord.selectNewLineItem('fieldset');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetname', 'matrixchilditems');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetid', 'matrixchilditems');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetrecordtype', 'ITEM');
```

```
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetfields', 'onlinecustomerprice_detail,in
ternalid,quantityavailable,outofstockbehavior,outofstockmessage,stockdescription,isinstock,isba
ckorderable,ispurchasable,showoutofstockmessage,itemid,minimumquantity');
siteRecord.commitLineItem('fieldset');
siteRecord.selectNewLineItem('fieldset');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetname', 'Correlated Items');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetid', 'correlateditems');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetrecordtype', 'ITEM');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetfields', 'itemimages_detail,itemoptions
_detail,internalid,onlinecustomerprice,onlinecustomerprice_detail,onlinecustomerprice_formatted
,onlinematrixpricerange,onlinematrixpricerange_formatted,quantityavailable,displayname,itemid,o
utofstockbehavior,outofstockmessage,stockdescription,storedescription,storedisplaythumbnail,sto
redisplayname2,isinstock,isbackorderable,ispurchasable,showoutofstockmessage');
siteRecord.commitLineItem('fieldset');
siteRecord.selectNewLineItem('fieldset');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetname', 'Related Items');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetid', 'relateditems');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetrecordtype', 'ITEM');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetfields', 'itemimages_detail,itemoptions
_detail,internalid,onlinecustomerprice,onlinecustomerprice_detail,onlinecustomerprice_formatted
,onlinematrixpricerange,onlinematrixpricerange_formatted,quantityavailable,displayname,itemid,o
utofstockbehavior,outofstockmessage,stockdescription,storedescription,storedisplaythumbnail,sto
redisplayname2,isinstock,isbackorderable,ispurchasable,showoutofstockmessage');
siteRecord.commitLineItem('fieldset');
siteRecord.selectNewLineItem('fieldset');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetname', 'Order');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetid', 'order');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetrecordtype', 'ITEM');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetfields', 'itemimages_detail,itemoptions
_detail,onlinecustomerprice_detail,displayname,internalid,itemid,isfulfillable,storedisplayname
2,urlcomponent,outofstockmessage,showoutofstockmessage,isinstock,isbackorderable,ispurchasable,
pricelevel1,pricelevel1_formatted,stockdescription,matrixchilditems_detail,itemtype,minimumquan
tity,isonline,isinactive');
siteRecord.commitLineItem('fieldset');

siteRecord.selectNewLineItem('fieldset');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetname', 'Type Ahead');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetid', 'typeahead');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetrecordtype', 'ITEM');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetfields', 'itemid,displayname,storedispl
ayname2,urlcomponent,itemimages_detail');
siteRecord.commitLineItem('fieldset');

siteRecord.selectNewLineItem('fieldset');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetname', 'Related Items Details');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetid', 'relateditems_details');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetrecordtype', 'ITEM');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetfields', 'relateditems_detail');
siteRecord.commitLineItem('fieldset');

siteRecord.selectNewLineItem('fieldset');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetname', 'Correlated Items Details');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetid', 'correlateditems_details');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetrecordtype', 'ITEM');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetfields', 'correlateditems_detail');
```

```
siteRecord.commitLineItem('fieldset');
recId = nlapiSubmitRecord(siteRecord);
```

## Script with Product Reviews

If the product list records bundle is installed in your account and you need product reviews, delete the details field set data and execute the following fast script:

```
var siteID = 99999;// change to the site id you want to use
var siteRecord = nlapiLoadRecord('website', siteID);

siteRecord.selectNewLineItem('fieldset');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetname', 'search');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetid', 'search');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetrecordtype', 'ITEM');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetfields', 'urlcomponent,itemimages_detai
l,itemoptions_detail,internalid,onlinecustomerprice,onlinecustomerprice_detail,onlinecustomerpr
ice_formatted,onlinematrixpricerange,onlinematrixpricerange_formatted,quantityavailable,display
name,itemid,outofstockbehavior,outofstockmessage,stockdescription,storedescription,storedisplay
thumbnail,storedisplayname2,isinstock,isbackorderable,ispurchasable,showoutofstockmessage,custi
tem_ns_pr_count,custitem_ns_pr_rating');
siteRecord.commitLineItem('fieldset');


siteRecord.selectNewLineItem('fieldset');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetname', 'details');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetid', 'details');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetrecordtype', 'ITEM');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetfields', 'urlcomponent,itemimages_detai
l,itemoptions_detail,internalid,matrixchilditems_detail,onlinecustomerprice_detail,quantityavai
lable,displayname,itemtype,itemid,outofstockbehavior,outofstockmessage,pagetitle,rate,rate_form
atted,relateditemsdescription,stockdescription,storedetaileddescription,storedisplayimage,store
displayname2,isinstock,isbackorderable,ispurchasable,showoutofstockmessage,metataghtml,minimumq
uantity,custitem_ns_pr_count,custitem_ns_pr_rating,custitem_ns_pr_item_attributes,custitem_ns_p
r_rating_by_rate');
siteRecord.commitLineItem('fieldset');

siteRecord.selectNewLineItem('fieldset');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetname', 'Correlated Items');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetid', 'correlateditems');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetrecordtype', 'ITEM');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetfields', 'urlcomponent,itemimages_detai
l,itemoptions_detail,internalid,onlinecustomerprice,onlinecustomerprice_detail,onlinecustomerpr
ice_formatted,onlinematrixpricerange,onlinematrixpricerange_formatted,quantityavailable,display
name,itemid,outofstockbehavior,outofstockmessage,stockdescription,storedescription,storedisplay
thumbnail,storedisplayname2,isinstock,isbackorderable,ispurchasable,showoutofstockmessage,custi
tem_ns_pr_count,custitem_ns_pr_rating');
siteRecord.commitLineItem('fieldset');

siteRecord.selectNewLineItem('fieldset');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetname', 'Related Items');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetid', 'relateditems');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetrecordtype', 'ITEM');
siteRecord.setCurrentLineItemValue('fieldset', 'fieldsetfields', 'urlcomponent,itemimages_detai
l,itemoptions_detail,internalid,onlinecustomerprice,onlinecustomerprice_detail,onlinecustomerpr
```

```
ice_formatted,onlinematrixpricerange,onlinematrixpricerange_formatted,quantityavailable,display
name,itemid,outofstockbehavior,outofstockmessage,stockdescription,storedescription,storedisplay
thumbnail,storedisplayname2,isinstock,isbackorderable,ispurchasable,showoutofstockmessage,custi
tem_ns_pr_count,custitem_ns_pr_rating');
siteRecord.commitLineItem('fieldset');

recId = nlapiSubmitRecord(siteRecord);
```

## Touch Points Available for Reference Implementations

You must choose touch points for the places on your site where you want visitors to interact with the SSP Applications installed with Reference SuiteApps. After deploying touch points, you can determine the URL you need to access your site. You can use this URL to create links and buttons shoppers can use to navigate your site.

For more technical information about touch points, see the help topic Selecting Supported Touch Points.

Some touch points are for Reference ShopFlow, some are for Reference My Account, some are appropriate for Reference Checkout. Best practice is to set touch points on the Web Site Setup page. See, Web Site Setup for Domains.

- Log In
- Log Out
- Proceed to Checkout
- Register
- View Cart
- View Customer Center
- View Homepage

## Working with Combination Files

Combination files are used for the aggregation, or packaging, of content in template and javaScript files. These combination files are installed by the reference SuiteApps in the **templates**, and **js** folders respectively. For more information on Combination Files see Understanding the SuiteCommerce Advanced Template Combiner.

- templates.config: Sets the configuration for combining all template files. Located at Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Shopping > Reference ShopFlow > templates
- combiner.config: Sets the configuration for combining all JavaScript files. Located at Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Shopping > Reference ShopFlow > js

When a file in the **templates** or the **js** folder is edited, new combined *Templates.js* file and *Application.js* files are generated with the current time stamp appended to the file name. These files aggregate the content in all template and JavaScript files in the folder where it is stored, and then provides a single URL that points to all the corresponding files:

```
<script src="/c.TSTDRV972624/ShopFlow/js/Application-013dccc20b5f.js"></script>
<script src="/c.TSTDRV972624/ShopFlow/templates/Templates-013dcae55ea9.js"></script>
```

**Important:** Do not modify these combined files, as they are referenced by other files to display templates properly. Any changes you make to these files could change its functionality. For more information on the Template Combiner, see Understanding the SuiteCommerce Advanced Template Combiner.

## Combiner Configuration Settings

The template.config and combiner.config files have several configuration parameters as follows:

- **Input-files**: Identifies the files to be concatenated in the combination process. Relative paths to files will be combined separated by a space character. Wildcards and recursion can be used for version 2. For the template combiner, .txt files are included, while for the JavaScript combiner, .js files are included. For example:
  - `*.txt` - combines all text files in the current folder (no recursion, recursion does not work for the base directory where the configuration file is placed).
  - `folder/*.txt` - combines all text files in folder and all its sub-folders.
  - `a*.txt folder/b*.txt` - combines all text files from the current folder and beginning with “a” as well as all text files beginning with “b” from folder and all it's sub-folders.
  - `src/app/modules/*.js` - combines all JavaScript files in the modules folder.

  **Important:** When including JavaScript files, order is important! Use caution when editing this configuration parameter!
- **Combined-file**: `combined.js`: Points to the file that aggregates template files. Includes timestamp to ensure uniqueness of the combined file.
- **Referring-files**: `../*.ssp`: Identifies all the files that reference the latest combined file. Note that if you want to create a new reference to combined-0123abc.js, you must put combined-0.js into your referring file.
- **Keep-files**: Sets the number of time-stamped combined.js files to keep. Note that a time-stamped version of combined.js is regenerated each time you save a change to a template file.
- **Version**: Sets the version of the config file. 2 indicates that the .config file is compatible with the latest version of SuiteCommerce Advanced, which allows you to use recursion for input files and the value for variable-name.
- **Variable-name**: For template configuration only, you can set the global variable name appropriate for your template solution. For example, `SC.templates`.

# Configuring Backend Objects Using the backend.configuration.js File

The backend.configuration.js file is provided so that you can configure a variety of objects server-side. Modifying these objects can often improve performance of your website by restricting search results. Currently, the backend.configuration.js files shipped with Reference ShopFlow and Reference Checkout, and Reference My Account Reference are the same although some objects are applicable for only certain implementations as described below.

The backend.configuration.js file is located at Web Site Hosting Files/Live Hosting Files/SSP Applications/[Reference Implementation]/ssp_libraries/backend.configuration.js

**Important:** This file can not be edited directly from the Reference Implementation folder. Copy the file to the same location in the Custom Reference Implementation SSP application folder and edit the file there. For details, see Using the Custom Folder.

## order_checkout_field_keys

The `order_checkout_field_keys` object defines the fields returned by the Commerce API from within Reference Checkout and Reference My Account for SuiteCommerce Advanced websites. To maximize performance, limit the set to only those fields required for your implementation. Settings here affect both Site Builder and SuiteCommerce Advanced Sites.

## order_shopping_field_keys

The `order_shopping_field_keys` object defines the fields returned by the Commerce API from within Reference ShopFlow for SuiteCommerce Advanced websites. To maximize performance, limit the set to only those fields required for your Reference ShopFlow Implementation. Typically, you need fewer fields defined here than for the Reference Checkout. Settings here affect both Site Builder and SuiteCommerce Advanced Sites.

## items_fields_advanced_name

The `items_fields_advanced_name` object sets the **Field Set** to be used when using the Item Search API to return item details. By default it is set to “details”. When using the provided setup script or when following the instructions provided in Minimum Setup Requirements, the suggested Field Set name is “details”. If you are using a different Field Set name you should define that Field Set name here. When using multiple Reference Implementations (ShopFlow, Checkout, and My Account), you can define a different Field Set name for each implementation and then customize this Object in the backend.configuration.js for the corresponding reference implementation.

Using a different Field Set name for each implementation can improve site performance. For each implementation only those details **required** are returned for the pages specific to that implementation. For example, in the ShopFlow item details page you may want to display a different set of fields than when your customer is in the My Account pages. In this case you can define two Field Sets: “details” to be used in Reference ShopFlow and “myDetails” to be used in the Reference My Account pages. The backend.configuration.js file for Reference ShopFlow can be left as is with “details” set for the `items_fields_advanced_name` object. The backend.configuration.js file for Reference My Account should be customized to set the `items_fields_advanced_name` object to “myDetails”.

## items_fields_standard_keys

When using Reference Checkout and Reference My Account in a Site Builder website, the `items_fields_standard_keys` object defines the set of fields to be returned by the Commerce API. To optimize performance, limit the set included here to only those fields required for your website.

Also, use this object to ensure that custom item fields in your account are included in the Item Search API results. Add the custom field name to the list.

**Note:** This object is used in **Site Builder** websites only. Field Sets are used to define the fields returned in SuiteCommerce Advanced websites as described in Field Sets.

## product_reviews

The `product_reviews` object sets parameters to be used for the Product Reviews module. For detailed information, see Modifying Product Review Configuration Defaults. Settings here affect SuiteCommerce Advanced Sites only.

## checkout_skip_login

When `checkout_skip_login` is enabled, the login/register page is skipped when an anonymous user navigates to Checkout. The user is redirected directly to the first checkout step. By default, `checkout_skip_login` is disabled.

When using Skip Login, a session is not defined until you have submitted an order. Because session information is not available

**Important:** `checkout_skip_login` is not supported for Checkout in Site Builder sites. Also, for a One Page Checkout flow with `checkout_skip_login` enabled, invoices can not be set as a payment option. This is because with `checkout_skip_login` enabled a session is not established until an order is placed. Without session information, it is not known whether the user has the permissions to pay with terms (invoices). `checkout_skip_login` is also not support when Multiple Ship To is enabled and configured for your web store.

# Customizing Implementations

Reference Implementations provide a solid starting point for your eCommerce solutions but are designed so that you can customize them for your specific needs. Customizations you can do including the following:

- Change the look and feel by editing CSS files, or template files.
- Change a touch point by editing its .ssp file. Note that if you make changes to other files, such as services files that affect a particular touch point, you also need to customize the .ssp file itself.
- Modify the ShopFlow, My Account, or Checkout experience by extending the implementation with a custom SSP application that you create.

**Important:** To customize reference implementations you must have experience working with HTML, style sheets, and JavaScript. The level of experience needed depends on the level of customization being done. Core functionality can be added or modified but advanced javascript programming skills are needed for this.

Before customizing implementations be sure to read and understand the following information:

- Technology and Framework: to get an understanding of the architecture used in Reference Implementations.
- SSP Application Overview: to become familiar with SSP applications and the Commerce API that supports them.
- SuiteCommerce Advanced Overview: to understand the components of an SuiteCommerce Advanced site and how they work together with the reference implementations.
- Item Search API Overview: to learn how to construct queries for product search results.

## Customization Method

Each of the reference implementations installs two SSP application folders in your NetSuite account. A Reference application folder, and a Custom application folder. To customize an application, you build your customizations in the Custom application folder. For full details, see Using the Custom Folder.

## Reference Implementation Template Guides

When customizing the reference implementations, you can debug the application client-side using standard JavaScript debugging techniques. Some of these techniques are described in Debugging and Making a Change to a Site. We have also provided detailed Template references so that you can quickly find what templates and macros are related to which sections of the site and specific details related to each component:

- Reference ShopFlow Templates
- Reference My Account Templates
- Reference Checkout Template Reference

# Using the Custom Folder

Each of the reference implementations installs two SSP application folders in your NetSuite account. A Reference application folder, and a Custom application folder. To customize or extend the reference implementation using the Custom Folder, you must first copy files to the Custom application folder and make modifications there.

When precedence overlays are set correctly, the SSP application first checks in the Custom folder and executes files that exist there. Otherwise code is executed from files within the Reference folder. For more information on understanding precedence overlays, see Using Precedence Overlays for Web Site Customization.

**Important:** Always copy files from within the NetSuite file cabinet. Important metadata is lost if the files are copied from within your SuiteScript IDE.

### To Customize a File using the Custom Folder:

1. Find the file you need to customize.

   For tips on how to troubleshoot and find the file that requires editing for the desired customization, see Debugging and Making a Change to a Site. For most customizations you will be editing the template and macro files.
2. Copy the file you need to customize from the Reference folder for the implementation to the Custom folder for the implementation. Ensure that the location of the file is exactly the same.
3. Edit files from the Custom folder as needed.
4. If the edited file is an SSP application file, configure the application as follows:

   **Note:** The following steps only need to be performed for .ssp files associated with a touch point (for example, index.ssp, goToCart.ssp, login.ssp. or logout.ssp files).

   1. Go to Setup > SuiteCommerce Advanced> SSP Applications.

      Edit the Custom Checkout SSP application.
   2. Associate the proper library scripts.

- Select the **Script Tab**.
- Add the following library files:

  **Important:** The order of these scripts must match EXACTLY.

  - Web Site Hosting Files/Live Hosting Files/SSP Applications/[Reference Implementation]/ssp_libraries/underscore.js
  - Web Site Hosting Files/Live Hosting Files/SSP Applications/[Reference Implementation]/ssp_libraries/backbone.validation.js
  - Web Site Hosting Files/Live Hosting Files/SSP Applications/[Reference Implementation]/ssp_libraries/commons.js
  - Web Site Hosting Files/Live Hosting Files/SSP Applications/[Reference Implementation]/ssp_libraries/backend.configuration.js
  - Web Site Hosting Files/Live Hosting Files/SSP Applications/[Reference Implementation]/ssp_libraries/Models.js

  **Note:** If it is the index.ssp file that you are editing, you may also need to add the content delivery and product merchandizing scripts: Web Site Hosting Files/Live Hosting Files/SSP Applications/NetSuite Inc. — ContentDeliveryService/ssp_libraries/ContentDeliveryService.Models.js and ssp_libraries/ and Web Site Hosting Files/Live Hosting Files/SSP Applications/NetSuite Inc. — Dynamic Merchandising/MerchandisingSSP/src/DM_SS_Library.js

3. Associate the appropriate touch point.
   - Select the **Touch Points** tab.
   - Select the appropriate touch point and set the Entry page to the customized ssp application file in the Custom folder.
4. Save and Deploy the Custom Checkout SSP to your site.
   - Select the **Save & Deploy to Site** button (from the Save drop down menu).
   - Choose your site.
   - Select **Save**.

# Using Template and Macro Files

Template and Macro files provide the core set of files that are used to define page layout and functionality. These files are customizable and extendable and will be the focus of much of your customization work. Following is a brief description of these files. For an example of debugging and customizing templates and macros see Debugging and Making a Change to a Site.

## Templates

In general, Template files are used to define the presentation layer of the website. These files contain HTML markup with place holders for data and scripts. All template files are grouped by function in hierarchical folder structure at Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Shopping > Reference ShopFlow > templates

The template files are named according to function and have a .txt extension. For example, the *item_details.txt* template is used to define layout for item details (as expected).

Multiple macros can be referenced from within a template. For example, the following line from the item_details.txt template is refering to the item_details_image_macro.txt macro.

```
<%= itemDetailsImage(view.model.get('_images'), view) %>
```

**Note:** JavaScript segments within the Template files must be enclosed with `<% %>`.

## Macros

Reusable html and JavaScript snippets are placed into macros which are then called from different templates. Macros can also call other macros.

Macro files are named according to function with _macro appended to the filename. For example, the item_details_image_macro.txt file is a macro that handles image display for item details and is called by the item_details.txt template.

Like other components, macros are customizable and extendable. You can also create entirely new macros and reference them from within your template files.

**Note:** When you change a macro declaration in a template, you must also make the change in the JavaScript associated with that template.

# Debugging and Making a Change to a Site


- Making a Simple Change
- Debugging in the Browser
- Debugging Macro Files
- Debugging Template Files


## Making a Simple Change

Following is a step through example of how to perform some simple debugging tasks and then make a change to the existing implementation. This is a quick overview only. For detailed examples and explanations for specific changes, refer to the appropriate section within the Advanced SuiteCommerce documentation set.

### To make a basic change to your web pages do the following:

1. Highlight the information you'd like to modify on the web page.
2. Right-click and select Inspect element or Inspect Element with Firebug (Firefox).

   The element selected displays in the debugger window. As you highlight different line items in the debugger window, the corresponding element is also highlighted in the browser.

   If you expand your debugger window or scroll up you can see the MACRO or TEMPLATE comment that indicates which file is related to the currently selected element. You can edit the template files for presentation changes to the current layout and edit the macro files for more sophisticated JavaScript or logic changes for the current element.

   **Note:** To view HTML comments in Firebug, ensure that you have the Show Comments option enabled in the HTML view.
3. If desired, edit layout changes to the page.

Layout or presentation changes are generally done in the Templates files.

> **Note:** Some layout components can also be dynamically rendered by the related view but in general, the TEMPLATES files are a good place to start.

1. Locate the related template string in the TEMPLATE STARTS comment that wraps the element to be changed. For example:

```
<!-- TEMPLATE STARTS: product_details -->
```

2. In the SuiteScript IDE, search for the file that includes the template string in the templates directory.
3. Verify that the file exists in the Custom folder of the Reference Implementation. If not, copy the file there from within the NetSuite file cabinet and then download to your SuiteScript IDE.
4. Make the desired changes, save the file, and upload the modified file to the NetSuite file cabinet.

   Use your browser debugger to locate the exact areas to change or refer to the Reference Templates documentation for the implementation you are working on.

4. Update any macros as needed.
   1. Locate the related template string in the MACRO comment that wraps the element to be changed. For example:

```
<!-- MACRO STARTS: facetList-->
```

   2. In the SuiteScript IDE, search for the macro string in the templates directory and locate the file that includes a registerMacro string. For example:

```
<%registerMacro('facetList'.....).
```

   3. Verify that the file exists in the Custom folder of the Reference Implementation. If not, copy the file there from within the NetSuite file cabinet and then download to your SuiteScript IDE.
   4. Make the desired changes, save the file, and upload back to the NetSuite file cabinet.
5. If desired, edit the view that is bound to the template of the selected element.

   To determine exactly what function you need to edit for a desired effect takes a bit of debugging and troubleshooting. Here are some hints of how to get started:
   1. Locate the view to be edited. All views are located in the following directory: Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Shopping > Custom ShopFlow > js > src > app > modules
   2. Verify that the file exists in the Custom folder of the Reference Implementation. If not, copy the file there from within the NetSuite file cabinet and then download to your SuiteScript IDE.
   3. If needed, debug the web application to find out exactly where you need to modify the code. See Debugging in the Browser.
   4. Modify the code and upload the view file to the NetSuite file cabinet.

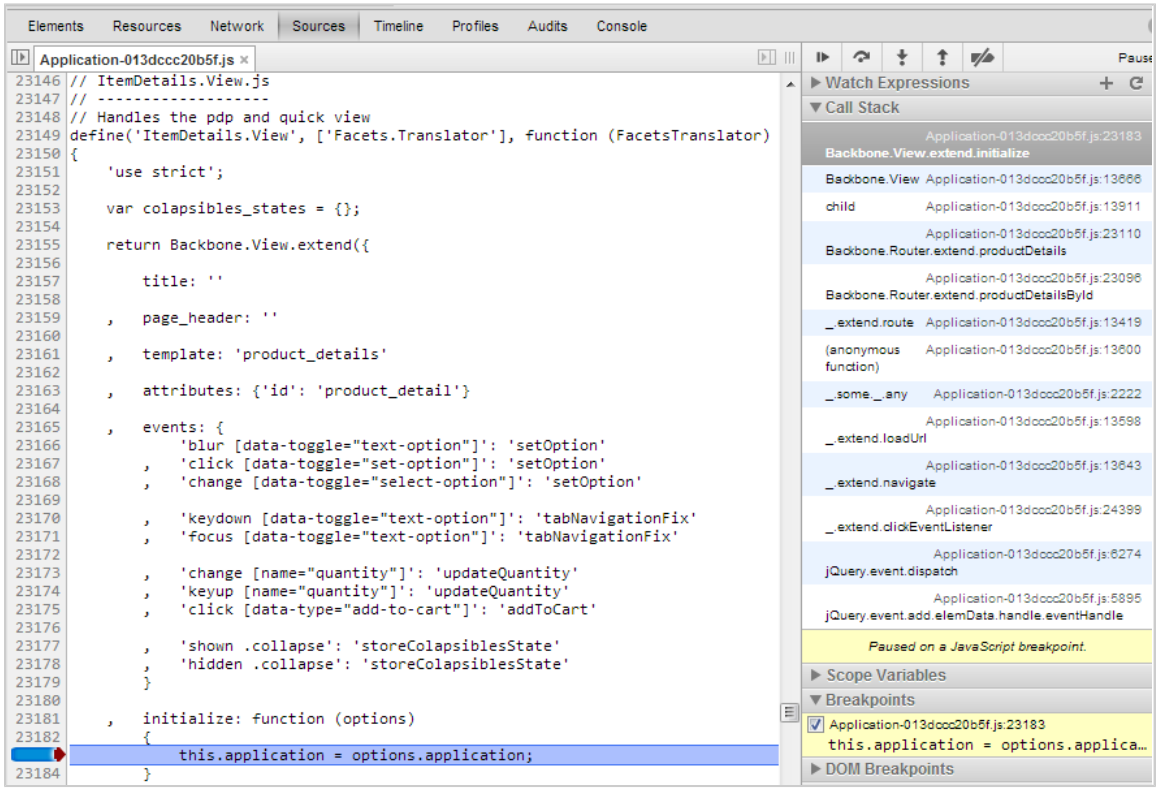
      All JavaScript files within the implementation are combined into a single Application JavaScript file using a combiner process that is controlled by the combiner.config file. These files are located at the top level of the js directory in the Custom folder. Whenever a change is made to a JavaScript file within the implementation, this combiner process

is initiated and a new combined Application JavaScript file is created. Additionally, the index.ssp file in the Reference folder is updated to reflect the new version of the Application JavaScript file.

## Debugging in the Browser

You can use standard JavaScript debugging techniques for troubleshooting your SSP application. Following are some basic guidelines.

> **Note:** For more efficient debugging, change the configuration settings within the combiner.config file as described in Setting Up Your Development Environment. As described in that section, you can debugging may be more efficient as you can examine non-minified JavaScript files individually.




### To debug your web application:

1. In the browser debugger, open the Application JavaScript file and search for the name of the View you want to edit.

   Within the Application JavaScript file, each view has comments before it with the name of the view and a short description. For example:

   ```
   // ItemDetails.View.js
   // -------------------
   // Handles the pdp and quick view
   ```

2. Insert breakpoints for desired functions by clicking on the left line item number and then step through the website as needed to located the points within the script needing modification.
3. At each breakpoint, examine the set of nested function calls currently running and waiting to return in the Callstack panel of the debugger window. Again, this is much easier to isolate

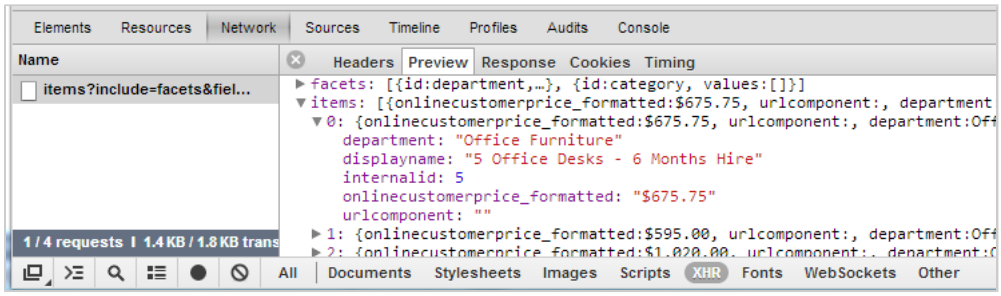
relevant code if you have created a development ssp application that loads each reference JavaScript file individually as described in Setting Up Your Development Environment.

4. Continue running the script by clicking the Resume Script Execution button (F8).
5. If desired, examine the JSON data that is being transferred.
    1. Go to the Network tab of the browser debugger window.
    2. Select XHR to display all AJAX calls being passed.
    3. Click on the desired call in the left panel to display the call details. For example, there is a item search api call made when initially searching for all items similar to the following:

```
http://suitedreams.btfeb201303b/api/items?include=facets&fieldset=search&language=
en&country=US&currency=USD&sort=relevance%3Aasc&limit=24&offset=0
```

You can also right-click on the AJAX call, select copy link address, and then paste this url in the browser to view the formatted JSON output.

> **Note:** For best results, install the JSONView extension for your browser. See Setting Up Your Development Environment.




## Debugging Macro Files

Macro file JavaScript code is not included in the main Application.js JavaScript file. Only references to macros can be found in that file. Once you have found the macro you wish to debug using the steps described in Debugging in the Browser, you can further debug the specific macro you need to change as described here.

Each time changes are made to a macro or template file and uploaded to the NetSuite file cabinet, a combiner process in initiated which combines and processes all of the files within the templates folder into a single JavaScript file. All of the macro files are pre-compiled during this combiner process and converted to pure JavaScript. For example, all symbols that are part of the templating engine are dynamically replaced. The combined Templates JavaScript file is located at:

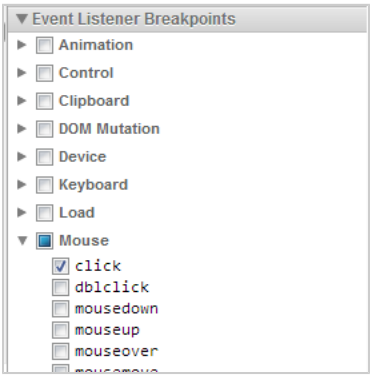
```
Custom ShopFlow/templates/Templates-[timestamp].js
```

When the Templates JavaScript file is loaded in the browser, it is loaded as a single *non-minified* file. Because of this, you can NOT directly debug this file using breakpoints and the callstack. Instead you need to set a Watch Expression for the SC.macros Object and step into and debug functions from there as described below.

### To access macro files for debugging:

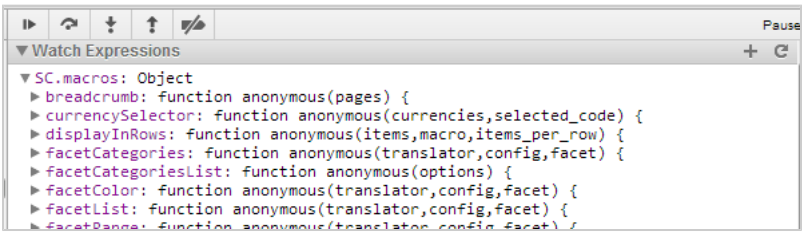
1. Set an Event Listener Breakpoint on Mouse click, reload your page, and then click your Shop button (or equavelent) as if you are navigating to the Items Search results page.

You must do this to gain access to a global variable that contains all of the pre-compiled templates and macros. This variable is not accessible until you have an active debugger session.




2. In the Watch Expressions panel, add the `SC.macros` object.

   All of the available macros are listed in this object. You can also view this global variable in the Scope Variables panel at Global > SC: Object > macros.




3. Right click on the desired macro function text and select Show function definition.

   The anonymous function is displayed in the left panel.

   > **Note:** Because macros are pre-processed and converted to pure JavaScript code during the combiner process, the functions loaded in the browser appear slightly differently than what you see if you open the macro from your SuiteScript IDE.

4. Remove the Mouse click breakpoint set in Step 1.

Now you have full access to the macro function and can debug using breakpoints or other techniques like any other JavaScript in the application.

## Debugging Template Files

Once you have found the template you wish to debug using the steps described in Debugging in the Browser, you can further debug the specific template you need to change as described here. Template files are processed real-time, not pre-processed like the macro files. Therefore the template functions are NOT accessible for debugging directly within the browser.

You can however, insert JavaScript statements directly within the template files. When adding statements you MUST add enclosing `<% %>` symbols for the statements to be recognized as JavaScript code. For example:

- Include console objects:

```
<% console.log('message'); %>
<% console.error('message'); %>
```

```
<% console.warn('message'); %>
<% console.info('message'); %>
<% console.debug('message'); %>
<% console.assert('message'); %>
```

**Note:** Once these console objects are included in your code, you can view all results together in the console panel of your brower debugger and navigate to the corresponding javascript file from there.

- Set breakpoints by including the following debugger statement. Once included, the application will break at that point and load the corresponding function in the debugger window.

```
<% debugger; %>.
```

**Important:** Ensure that you remove all debugger statements before going to production with your application!

# Technology and Framework

This section provides information about the technologies and frameworks used in the reference implementations. See the following sections:

- Overview
- Libraries
- SuiteCommerce Reference Implementation Web Assets
- Setting Up Your Development Environment

## Overview

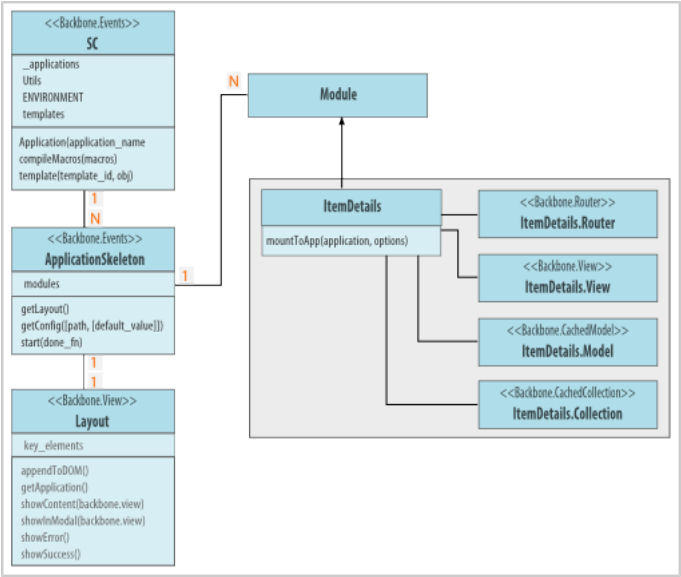
Reference implementations are built on modern technologies and provide a functional and well designed starting point for your web store experience. However, the implementations are built to be fully extensible so that you can develop on top of this framework to improve your site specific to your needs. The code consists of hundreds of modules and extensions in modular components to make it easier to maintain and add modules.

To get a good understanding of the overall architecture used in the Reference Implementations, read the following sections.

- Class Diagram
- Startup Sequence
- Extending the Implementation
- Understanding Item Data Display

### Class Diagram

The following diagram outlines the main components of the SuiteCommerce Advanced Reference Implementations.




`SC` (SuiteCommerce) is the namespace. Within the namespace there are several variables and methods. You can also have several Applications within the same `SC` namespace.

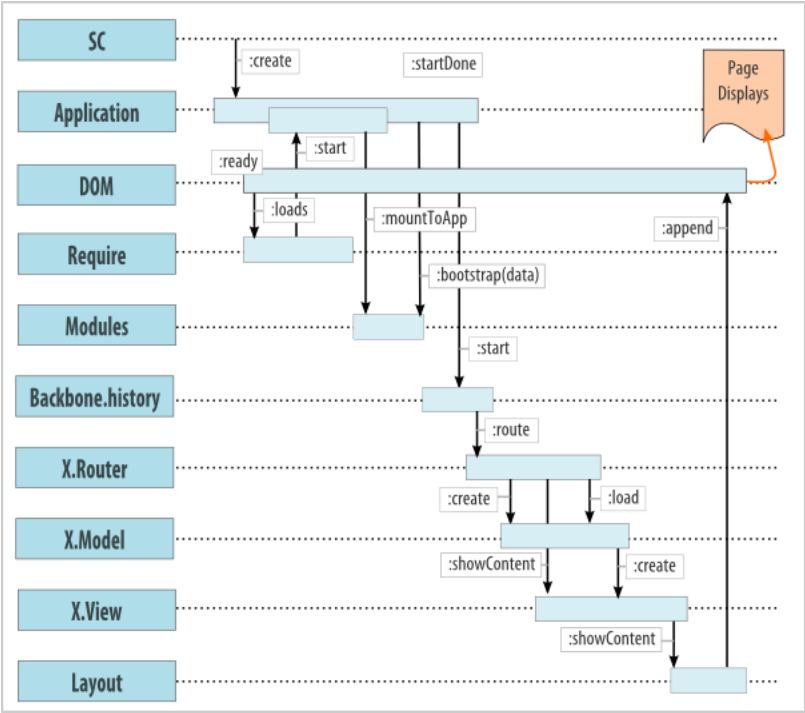
The `ApplicationSkeleton` class represents one instance of an application. For example, ShopFlow is one application, My Account another etc. The Application function is used to instantiate an application rather than creating a new class for each application. For example:

```
var application = SC.Application('Shopping');
```

Within each application function, a collection of all modules loaded for this application are defined. Also, the configuration and layout elements are defined and a start function that iterates through all modules and instantiates them. For each application, you can have multiple modules but only one Layout defined. Each module is a collection of classes. In the example above, the standard `ItemDetails` Module is displayed. This represents the most typical type of module consisting of a Router, View, Model, and a Collection.

## Startup Sequence

The following diagram illustrates the startup sequence of a SuiteCommerce Reference Implementation.




First `SC` creates the Application. Once the Application is created it remains idle until the DOM is loaded. Once the DOM is loaded, require modules that need to be bootstrapped before the application starts are loaded. Once require modules are loaded, a callback function to the Application initiates the start function. The Application then mounts the necessary modules and bootstraps additional data. Once the start function is complete, the backbone history reads the URL components and calls the necessary router, models, views and layout modules. The Layout is then appended to the DOM for presentation.

## Extending the Implementation

Reference Implementations are fully extensible. Every object and every value is publicly available. Most objects are available through the `SC` and `SC.Application` objects. For example in the following snippet you can access the facets for the current view:

```
SC.Application("Shopping").getLayout().currentView.translator.facets
```

In addition, every method can be overridden or extended. For example you can add your own custom logic:

```
Application.getLayout().showContent = _.wrap(Application.getLayout().showContent, function (fn,
 view, dont_scroll) {})
```

> **Note:** When extending the code, ensure that you use meaningful naming conventions. As the above code samples illustrate, existing code uses readable, meaningful names that are understandable.

Reference Implementation code is modularized and separated into the following two main types:

- **Core Components**: Adds functionality to the framework. JavaScript for these modules are at js/scr/core.

Within the core components, there are many extensions located in the js/scr/core/extras folder. These modules contain core functionality but often tie you to external frameworks or libraries. By containing these extensions in separate modules you can modify or replace them as needed.

- **Application Modules**: Adds functionality to the application. JavaScript for these modules are at js/scr/app.

  Application modules are designed to be very flexible. There are few limitations on how you structure them. For example, you can use standard backbone routers, views, models and collections. Many of the standard modules provided are implemented with this structure.

  However, your modules can be designed without the backbone structure to provide whatever functionality is needed. For example, the GoogleAnalytics.js module does not use the typical views. The module is defined like any other module and connected to the application via the `mountToApp` function but is used to add functions to the application that provide enhancements for communication with Google Analytics.

**Extending Code Examples:**

Following are some examples of how to extend your implementations.

- **Extend the application**: the following code found in Cart.js is used to add the `getCart` function to the application.

```
application.getCart = function ()
{
   return Model.getInstance();
};
```

- **Extend SC.Utils**: the following code found in UrlHelper.js is used to add some functionality to `SC.Utils` object. There are many utilities included with the implementations but you can add your own custom utilities by extending the `SC.Utils` object in a similar manner.

```
_.extend(SC.Utils, {
   fixUrl: fixUrl
,   setUrlParameter: setUrlParameter
,   removeUrlParameter: removeUrlParameter
});
```

- **Add code for all applications**: Ideally `SC` has only one instantiated application but it is designed to support a collection of applications if needed. In most cases, an application is extended by calling it directly as in the following code within the Configuration.js file:

```
_.extend(SC.Application('Shopping').Configuration, {
   ...
})
```

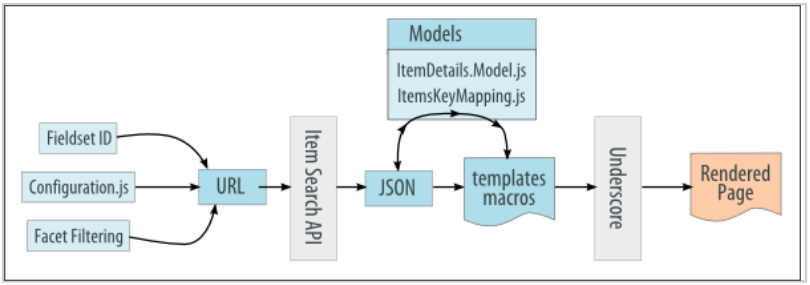
  However, you can also extend the implementation to add code to every instantiated application by using `SC_applications`. The following code found in ItemsKeyMapping.js is used to iterate through each instantiated application and add the `KeyMapping` configuration.

```
_.each(SC._applications, function (application)
{
   ...
   application.Configuration.itemKeyMapping = _.extend(application.Configuration.item
   KeyMapping || {}, {
   ...
```

```
        })
        ...
    })
```

## Understanding Item Data Display

This section illustrates how item data is retrieved and exposed from within Reference ShopFlow. Similar processes are followed for data retrieval throughout the reference implementations.




Item URL parameters can be determined by several factors including the Fieldset ID defined within the Web Site Setup record, processing done by the configuration file, and facet filtering.

Once the URL is known, an AJAX call is made to the Item Search API and data is returned in JSON format. The JSON data is handled by the template/macro processing engine that interacts with Backbone models. The Underscore library parses the resulting tagged based template/macro files to render pure JavaScript.

**Note:** Most processing is done client side with the exception of the Item Search API.

# Libraries

Listed below are the technologies and frameworks that were used to create the reference implementations. Familiarize yourself with each of these before you customize any of the reference SuiteApps you installed.

- **Backbone.js** : gives structure to web applications by providing the following components:
  - Model: Represents data and logic surrounding it. Models are used to interact with the backend data. Models communicate via JSON with the server through service files (.ss) via the commerce API and the Item Search API.
  - Collection: Ordered sets of Models
  - View: A module backed by a Model with a render method.
  - Router: Mechanism to provide linkable, sharable URLs. Router execution provides control mechanisms that are executed based upon a URL format being passed in a site.

  You can build and customize routers and views as needed. Note that if you add a new Module, you need to add the reference to the module in the config file as well.

  http://backbonejs.org/
- **Bootstrap.js** : equipped with HTML, CSS, and JS for Scaffolding, Base CSS, Components and JavaScript plug-ins. Bootstrap provides us with Responsive Design out of the box.

http://twitter.github.com/bootstrap/

- **jQuery.js:** a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.

  http://jquery.com/

- **JSON** : (javascipt object notation) the data interchange format used in the reference implementation.

  www.json.org

- **Underscore** : a utility-belt library for JavaScript that provides a lot of the functional programming support that you would expect in Prototype.js (or Ruby), but without extending any of the built-in JavaScript objects. It is used for many of the nitty gritty html details. The templating process is governed by the underscore library.

  http://underscorejs.org/

- **Require.js** : an Asynchronous Module Loader that allows us to define and require modules. The Require library helps to modularize the codebase. The Router, Model, and Views are components loaded by Require.js. All modules are loaded with a define statement and each define function is defined within the require.js file. Each define statement defines the module with all of it's dependency atttributes and parameters needed to define the required function that is executed as a callback function at the appropriate time in the script.

  http://requirejs.org/

- **LESS** : a CSS pre-processing language that extends CSS with dynamic behavior such as variables, mixins, operations and functions. LESS is leveraged in the default skin and can be used when designing your own custom CSS Skins.

  http://lesscss.org/

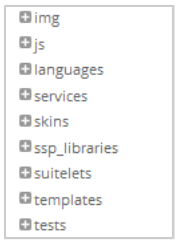
> **Note:** Shipping integration and address look-up are part of the platform exposed by NetSuite's Commerce API. For more information, see the Commerce API Overview.

## SuiteCommerce Reference Implementation Web Assets

This section details the web assests installed with all of the reference SuiteApps. When a Reference Implementation is installed, two top level folders are created, a Custom Folder and a Reference Folder. The Custom Folder is largely empty and is to be used for customizations (refer to Customizing Implementations). Here, the contents of the Reference folder is detailed.

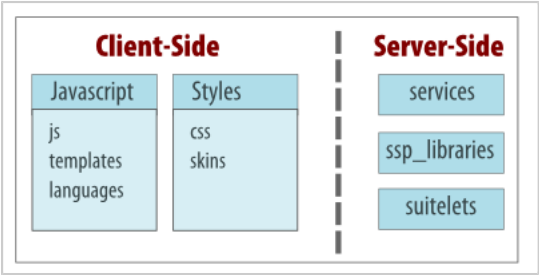
Web assests are installed in the file cabinet at File Cabinet > Web Site Hosting Files > Live Hosting Files > SSP Applications. Web assets for each SuiteApp are contained in a subfolder under SSP Applications.

> **Note:** The same folder structure is used for the Shop Flow, My Account, and Reference Checkout Reference Implementations.



## List of Web Assets

The following table outlines all of the assets included in the reference implementations. Note that files are processed client versus server side according to the following diagram.




| Folder | Files/Notes |
|---|---|
| css | This folder contains the stylesheets used by the application. Files in this folder provide default css styling including responsive web design settings. If you do not want to use these in your site, the best practice is to not call them rather than deleting them. |
| img | contains all image files |
| js | ▪ html5shiv.js<br>▪ json2.js<br>These two files are libraries used to fix problems that could arise from viewing the site with browsers that do not support HTML5 or JSON.<br>It is recommended that you do not change these files. If you do not want to use them, the best practice is to not call them in the SSP application. |
| js > extensions | Add files to extend the reference implementation here. |
| js > libs | This folder includes all of the front-end libraries used by the implementation. |
| js > src > app | This subfolder includes methods specific to each reference SuiteApp. It includes the following files:<br>▪ Application.js: extends the application with specific core methods and defines the layout.<br>▪ Configuration.js: contains the configuration of the application. (There are more configuration options in Shop Flow than in My Account). Note that if you add a new Module, you need to add the reference to the module in the config file as well.<br>▪ current.js: shows which combined file is most current and being used. |
| js > src > app > modules | Contains the front-end modules. Some modules in this folder are dependent on other modules in this folder. |
| js > src > core | ▪ main.js: defines main application architecture.<br>▪ utils.js defines methods for formatting and validation.<br>▪ ApplicationSkeleton.js defines the top-level components of the application<br>For more information on these files, review the commenting within each file. |
| js > src > core > extras | The files in this subfolder include methods used by all applications. These files provide examples of extensions of the application. These extensions override some default methods in the application. |
| languages | JSON files that include translated strings in different languages for the app. |

| Folder | Files/Notes |
|---|---|
| | Each file name must match the standard JSON language codes. The translation languages must also be configured in your NetSuite account at Setup > Company > General Preferences > Languages. |
| services | This folder contains all of the SuiteScript/services files used to communicate from front-end to the server. |
| skins | This folder contains the default skin for your implementation. Additional skins may become available at a later date. You can also create your own custom skins. |
| ssp_libraries | These files are used for the services (SSP applications). See ssp_libraries. |
| suitelets | This folder contains the following files:<br>▪ NS_SF_SS_Post_Install.js: This script runs after a bundle is installed and triggers all the combiner files. For more information, see Working with Combination Files.<br>▪ NS_PR_SS_User_Event.js: This works around permission issues in the implementation and will be deprecated in a future version.<br>**Important:** Do not remove or reference this file in custom implementations. |
| templates | This folder contains all of the template and macro files used for the reference SuiteApps. All templates and macros are organized in subfolders beneath this top directory. These files get concatenated into a single javascript file which is then served client-side and processed by the underscore templating engine. See Using Template and Macro Files. |

## ssp_libraries

These files are used for the services (SSP applications). The libraries are listed here.

The order in which these files are listed on the SSP application record is important as they build on each other. Any values or methods you define in these files can be used in any SSP applications.

These files include:

- **commons.js**

  This file contains utility methods used in the models and it extends the models to allow PUB/SUB (publish and subscribe).

  *models.js* uses *commons.js* to define the models and to have default methods.

  *commons.js* defines server-response error objects (404, 405, etc.) and the message that is displayed for each of these errors.

- **models.js**

  This file defines each model and its reuse. Each service uses the models defined here.

  This file has validations for each model. For example, you can define these models to do things like require first name for an address or to call the default shipping address for a customer order.

- **backend.configuration.js**

  This file provides a means to define several different types of server-side objects. For more information see, Configuring Backend Objects Using the backend.configuration.js File.

**Important:** These files should not be changed, but you should review them before customizing your site.

## Setting Up Your Development Environment

Any modern web development environment can be used for working with Reference Implementations. Following are some suggestions for a minimal web development configuration:

- **SuiteCloud IDE** : the SuiteCloud IDE is an Eclipse plugin that allows for easy integration with the NetSuite file cabinet. See the help topic *Setting Up Your SuiteScript Development Environment (SuiteCloud IDE)* for details on how to setup and configure the IDE.

  You'll want to create two projects: a Custom ShopFlow project and a Reference ShopFlow project.

  **Note:** When adding your projects ensure that you define the projects as an SSP Application Project and download all files from the top level of the Custom ShopFlow and Reference ShopFlow directories respectively.

- **Set Up Eclipse File Associations** : template and macro files in the Reference Implementations are .txt files even though they contain HTML markup and JavaScript code. When opened in the SuiteScript IDE, these files will display without any syntax highlighting associated with them. You can set up file associations in the IDE so that these .txt files will open by default with a text editor of your choice.

  To set up file associations:

  1. In the SuiteScript IDE, go to Window > Preferences > General > Editors > File Associations.
  2. Click Add in the top panel and enter .txt.
  3. Hightlight the new file type and then click Add in the lower panel and browse to the desired editor (for example, Komodo or Sublime). Now when you double-click on a template or macro file in the SuiteScript IDE, the file will automatically open in your default editor where you can quickly select the desired syntax highlighting.

- **Firebug/Chrome Debugger** : Chrome has a built in debugger that can be accessed via the File > Tools > Developer tools menu (or hit F12). Firebug is a Firefox plugin that must be installed separately (also accessible by hitting F12).

- **JSONView** : This is an Chrome and Firefox extension that once installed and enabled allows you to view prettified JSON in a browser window.

  URLs associated with AJAX calls can be accessed in the XHR panel of the Network tab in the debugger window. Right click on AJAX calls displayed in the Name column, select Copy Link Address and then paste this URL into your browser. The item search api JSON result set is displayed.

- **Change the defaults in combiner.config (optional)** : Whenever a change to a JavaScript file is uploaded to the NetSuite file cabinet, the combiner.conf file is used in a process that combines all JavaScript files into a single Application JavaScript file. The combiner file is located in the js folder of the Reference Implementation and has several configuration settings defined.

  The Method parameter has a default value of Minify. This causes the combiner process to produce a minified Application JavaScript file where most whitespace and comments are removed. For *development purposes only*, you can change the Method parameter to Append. When set to Append the application loads the JavaScript file as a non-minified file making it much easier to debug.

  **Important:** Ensure that you reset the Method to **Minify** when you are ready for production. Otherwise, you will see a significant performance hit on your site.

- **Itemize JavaScript Files in a Development index.ssp (optional)** : By default, all JavaScript files within the application are combined into a single Application JavaScript file. For debugging purposes, it can

be useful to load each JavaScript file individually. To do this, modify the index.ssp file to include each script. Note the following:

- Ensure you first download the latest version of the index.ssp file from the NetSuite file cabinet to your Reference folder.
- Copy the index.ssp file to your custom folder and rename it. For example, index-dev.ssp
- In the new .ssp file, replace the script line referencing the Application_[timestamp].js file with a line for each JavaScript file listed in the combiner.config file Input-files parameter and then upload the file to the NetSuite file cabinet.

  **Important:** You MUST reference these scripts in the EXACT same order that they appear in the combiner.config file.

- Add the following Library Script Files in the SSP Application page at Setup > SuiteCommerce Advanced > SSP Applications > Edit. These scripts MUST be entered in this order.
  - Web Site Hosting Files/Live Hosting Files/SSP Applications/NetSuite Inc. - Shopping/Reference ShopFlow/ssp_libraries/underscore.js
  - Web Site Hosting Files/Live Hosting Files/SSP Applications/NetSuite Inc. - Shopping/Reference ShopFlow/ssp_libraries/backbone.validation.js
  - Web Site Hosting Files/Live Hosting Files/SSP Applications/NetSuite Inc. - Shopping/Reference ShopFlow/ssp_libraries/commons.js
  - Web Site Hosting Files/Live Hosting Files/SSP Applications/NetSuite Inc. - Shopping/Reference ShopFlow/ssp_libraries/backend.configuration.js
  - Web Site Hosting Files/Live Hosting Files/SSP Applications/NetSuite Inc. - Shopping/Reference ShopFlow/ssp_libraries/Models.js
- Start your web application using the modified ssp application URL. For example, Domain/ShopFlow/index-dev.ssp.

**Note:** When itemizing JavaScript files as described here you may experience some caching issues. If you do NOT see expected changes try clearing your browser cache. Normally, caching issues are not an issue since a new combined JavaScript file with a unique timestamp is loaded but in this case we are bypassing the combiner process to load each file individually.

# Google Analytics for Reference Implementations

Google Analytics is a third party analytics solution that can help you evaluate traffic on your website using data based on visitor tracking information. You can use this information to optimize your site content and ad campaigns. There are two versions of Google Analytics, Classic Google Analytics (ga.js) and Google Universal Analytics (analytics.js).

- **Google Universal Analytics** (analytics.js): Supported for Reference ShopFlow, Reference My Account, and Reference Checkout. This is the latest Google Analytics implementation that gives you access to improved data processing, new collection methods, and more analysis tools.
- **Classic Google Analytics** (ga.js): Supported for Reference Checkout and Reference ShopFlow. This is the legacy Google Analytics implementation.

> **Note:** Support for ga.js is also included for Reference ShopFlow but we recommend that you use analytics.js if you are beginning a new implementation. Existing ga.js users should create a new web property for analytics.js and dual tag their site. For information on setting up new web properties, refer to the Google Help article Set up a property.

In order to implement Google Analytics, you'll need to edit the configuration file for each implementation. You can also customize the Google Analytics module for each implementation to exclude or include analytics for specific components.

Once Google Analytics is enabled and configured, each page hit in your ShopFlow site is tracked. For Checkout, each Item in the cart is tracked and once the order is completed details of the order such as store name, shipping, state, and country etc are also tracked.

> **Important:** Settings on the Analytics tab in the Web Site Set Up page are NOT used when enabling Google Analytics for *Reference Implementations*. Only the settings in the Configuration.js and GoogleAnalytics.js files for each implementation control the Google Analytics, as described below. For details on how to implement Google Analytics for sites NOT based on the Reference Implementations, see the help topic Working with Google Analytics.

The following files in each Reference Implementation are used to control Google Analytics:

- **Configuration.js**: located at js > src > app. Used to load the analytics modules and configure your account information.
- **GoogleAnalytics.js**: located at js > src > app > modules > GoogleAnalytics. Used to define your event methods for Classic Google Analytics.
- **GoogleUniversalAnalytics.js**: located at js > src > app > modules > GoogleAnalytics. Used to define your event methods for Google Universal Analytics.
- **ApplicationSkeleton.Tracking.js**: located at js > src > core > extras. Used to implement the support for multiple tracking solutions. Normally, this file does NOT need to be edited.

> **Note:** The GoogleAnalytics.js and file used for ShopFlow is the same as that used for the Checkout module. However, some functions are only used for the Checkout implementation versus the ShopFlow implementation.

# Enabling Google Analytics

## To enable Google Analytics for your Reference Implementation:

1. Open the Configuration.js file at js > src > app.
2. Ensure that the Google Analytics module(s) are enabled.

   By default these modules are enabled but can be commented out at any time. The module(s) are listed as shown in the figure above.
3. Enter the tracking settings.

   Further down in the Configuration.js file is the Analytics Settings section. You MUST set the following parameters:

   - `propertyID`: enter your ID as defined when you set up your Google Analytics account.
   - `domainName`: enter your domain name. This MUST match the domain name define in your Google Analytics account.

> **Note:** Ensure that you enter the proper domain or subdomain used for each implementation. For example, you may be using a secure domain for Checkout but not for ShopFlow. In this case configure the Checkout Configuration.js file with the secure domain/subdomain and configure the ShopFlow Configuration.js file with the non-secure domain. Obviously, Google Analytics can NOT be used during development on local domains.

- `trackPageview`: This parameter is available in Reference Checkout only. Set to true. Once set to true the GoogleAnalytics.js file is loaded in the application and each page view is tracked.

```
   ,   tracking: {
         // [Google Universal Analytics](https://developers.google.com/analytics/devguides/coll
ection/analyticsjs/)
         googleUniversalAnalytics: {
            propertyID: 'UA-XXXXX-Y'
         ,   domainName: 'mynetsuitesite.com'
         }
         // [Google Analytics](https://developers.google.com/analytics/devguides/collection/gaj
s/)
      ,   google: {
            propertyID: ''UA-XXXXX-Y'
         ,   domainName: 'mynetsuitesite.com'
         }
      }
```

4. Optionally, enable additional tracking.

   By default, the `trackPageview` function is used. In addition to tracking page views you can enable tracking of events. To enable event tracking add the reference to the `trackEvent` function in the Configuration.js file and set to true.

```
// Analytics Settings
,   tracking: {
         trackEvent: true
         googleUniversalAnalytics: {
            propertyID: 'UA-XXXXX-Y'
         ,   domainName: 'mynetsuitesite.com'
         }
      ,   google: {
            propertyID: 'UA-XXXXX-Y'
         ,   domainName: 'mynetsuitesite.com'
         }
      }
```

   The `trackEvent` function in the Google Analytics module defines several user events that can be tracked.

```
,   trackEvent: function (event)
      {
         win[name]('send', 'event', event.category, event.action, event.label, event.value, {
            'hitCallback': event.callback
      });   ,   trackEvent: function (event)
```

5. Optionally, edit the GoogleUniversalAnalytics.js (or GoogleAnalytics.js) file.

Once the Configuration.js file is edited as described above, your Google Analytics tracking is functional. However, you can also customize the Google Analytics module to suit your tracking needs.

By default the `addItem`, `addTrans`, and `trackTrans` Google Analytics methods are supported. For example, the `addTrans` method with all supported parameters is configured and used in Checkout Reference as shown below. You can modify this to include or exclude parameters as needed following the Google Analytics API guidelines. For more details on the Google Analytics eCommerce API refer to the page "Ecommerce Tracking - Web Tracking (analytics.js)". If you are using Google Analytics Classic, refer to "Tracking Code: Ecommerce".

- Universal Analytics code sample:

```
,   addTrans: function (transaction)
      {
         // [Adding a Transaction](https://developers.google.com/analytics/devguides/collection
/analyticsjs/ecommerce#addTrans)
         win[name]('ecommerce:addTransaction', transaction);
         return this;
      }
.
.
.
GoogleUniversalAnalytics.addTrans({
      id: transaction_id
,   revenue: order_summary.subtotal
,   shipping: order_summary.shippingcost + order_summary.handlingcost
,   tax: order_summary.taxtotal
,   currency: 'SC.ENVIRONMENT.currentCurrency.code'
});
```

- Classic Analytics code sample:

```
   ,   addTrans: function (transaction)
      {
         // [_addTrans()](https://developers.google.com/analytics/devguides/collection/gajs/met
hods/gaJSApiEcommerce#_gat.GA_Tracker_._addTrans)
         _gaq.push(['_addTrans'
         ,   transaction.id
         ,   transaction.storeName || SC.ENVIRONMENT.siteSettings.displayname
         ,   transaction.subtotal
         ,   transaction.tax
         ,   transaction.shipping
         ,   transaction.city
         ,   transaction.state
         ,   transaction.country
         ]);

         return this;
      }
```

6. Edit your Google Analytics account configuration to exclude the following URL parameters:

   `fragment, whence, quantity, order, page, last_order_id, nocache, target_id, fromcdn, seonojscache, seodebug, force, manage, show, cartitemid`

   For more information, see Configuring Your Google Analytics Account.

# Checkout 2.05.0 and Google Analytics with Firefox in a Private Window

If a visitor to your site attempts to login with Firefox in a private window and you have Google Analytics enabled, the login can fail. To prevent this login failure, you must make some minor changes to the `GoogleUniversalAnalytics.js` file and also to the `ApplicationSkeletion.Tracking.js` file. See the following procedure for instructions on making these changes.

## To enable login with Firefox in a private window:

1. In the file cabinet, copy the following file:

   Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Checkout 2.05.0 > Reference Checkout > js > src > app > modules > GoogleUniversalAnalytics > GoogleUniversalAnalytics.js

   to this new location:

   Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Checkout 2.05.0 > Custom Checkout > js > src > app > modules > GoogleUniversalAnalytics > GoogleUniversalAnalytics.js

2. In the file cabinet, copy the following file:

   Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Checkout 2.05.0 > Reference Checkout > js > src > core > extras > ApplicationSkeleton.Tracking.js

   to this new location:

   Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Checkout 2.05.0 > Custom Checkout > js > src > core > extras > ApplicationSkeleton.Tracking.js

3. In the Custom Checkout folder, edit the `GoogleUniversalAnalytics.js` file and replace the following code:

```
var GoogleUniversalAnalytics = {
                trackPageview: function (url)
```

   with this code:

```
var GoogleUniversalAnalytics = {
                isBlocked: function ()
               {
                       return win[name].q;
               }
        ,        trackPageview: function (url)
```

4. In the Custom Checkout folder, edit the `ApplicationSkeleton.Tracking.js` file and replace the following code:

```
        // GoogleUniversalAnalytics has an asynchronous callback.
        // So we anly call the non async ones if UniversalAnalytics is not there.
        !has_universal_analytics && event.callback();
```

   with this code:

```
        // GoogleUniversalAnalytics has an asynchronous callback.
        // So we only call the non async ones if UniversalAnalytics is not there.
```

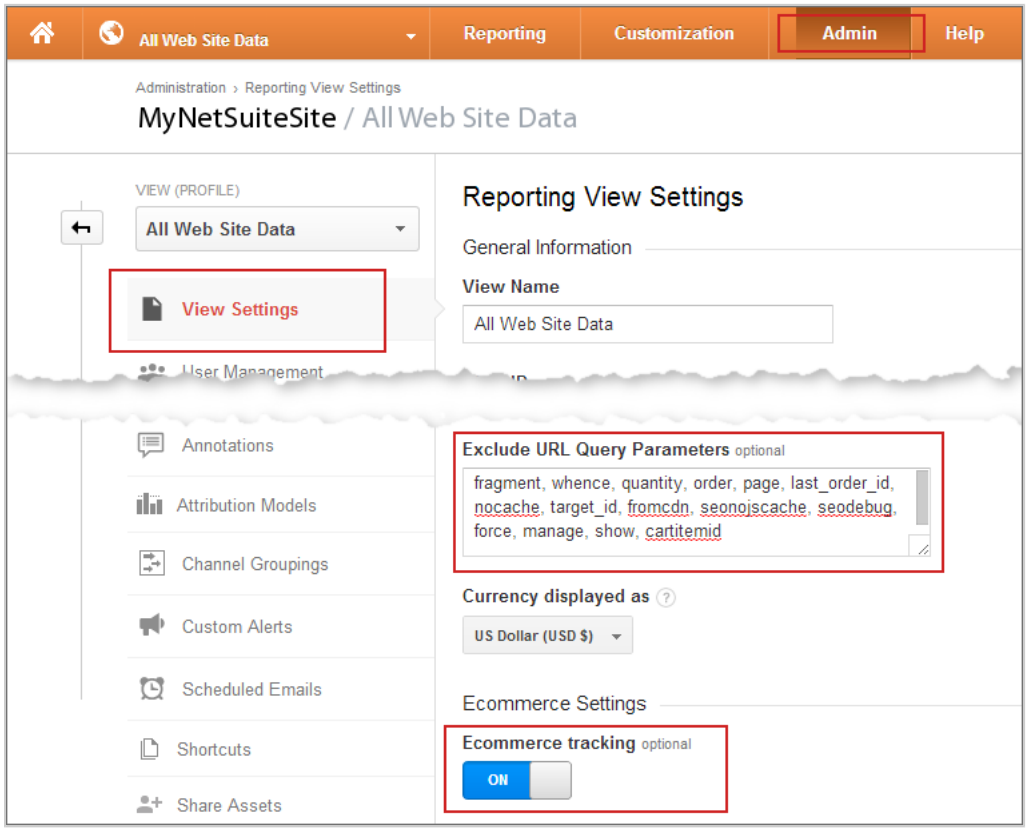
```
        // OR if we have UniversalAnalytics but the browser blocks the tracking traffic
        if (!has_universal_analytics || has_universal_analytics.isBlocked())
        {
                event.callback();
        }
```

5. Go to Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Checkout 2.05.0 > Custom Checkout > js in the file cabinet.
6. Click the **Edit** link for the `combiner.config` file. This opens the file's properties page.
7. Click the **Save** button.
8. Click the **OK** button when you are presented with a message that states the record has not changed and asks if you really want to submit it.

# Configuring Your Google Analytics Account

In order to obtain valid tracking results from your Google Analytics account you should have the following settings configured in the View (Profile) settings of your Google Analytics account. The View (Profile) settings can be found on the Admin page of your Google Analytics account.

- Exclude URL Query Parameters
- Ecommerce Tracking




## Exclude URL Query Parameters

There are several URL parameters that are used within the Reference Implementation applications. You should configure your Google Analytics account to exclude these parameters from your tracking. When

these parameters are NOT excluded, your Google Analytics data may contain duplicate and inaccurate results. For example, `fromcdn`, `seonojscache`, and `seodebug` are all used internally for debugging purposes and when used within your live site these URLs should not be tracked in your analytics data.

The following parameters should be excluded:

```
fragment, whence, quantity, order, page, last_order_id, nocache, target_id,
fromcdn, seonojscache, seodebug, force, manage, show, cartitemid
```

**Note:** When customizing your Reference Implementation, you may use additional parameters that should be excluded. You should always evaluate URL parameters to determine if they should be excluded in your Google Analytics account configuration.

## Ecommerce Tracking

Before Google Analytics can report Ecommerce activity for your website, you must enable **Ecommerce tracking** on the view (profile) settings page for your website as indicated in the above image.

# Creating Custom Analytics Trackers for 3rd Party Services

3rd party analytics services can be implemented using the same framework as the Google analytics services by creating a new **tracker** module. Each tracking module contains the logic specific to the tracking service being used but all tracking objects are called from a central file and processed in the same way. This modular design provides an easy way to extend the Reference Application to use the analytics services of your choice.

**Note:** Multiple tracker modules can be defined and used simultaneously for your site.

The script that handles trackers is **ApplicationSkeleton.Tracking.js** at /js/src/core/extras/ApplicationSkeleton.Tracking.js. This script extends the prototype of the Application with a place holder for the list of trackers, the set of predefined methods to be called for each tracker and a generic method **track** that iterates through the list of trackers and calls the method passed as a parameter.

Currently, the following predefined methods are available:

- **trackPageview**: called when the URL changes.
- **trackEvent**: called when tracking an action, like the Guest Checkout button was clicked.
- **addCrossDomainParameters**: called when changing domain, for instance going from shopping to checkout.
- **trackTransaction**: called on the Thank you page after the order is placed.

**Important:** These methods are called for each tracker. For example, if there are multiple 3rd party services with methods to track transactions, the **trackTransaction** method is called on the *Thank You* page for each service. Also, these methods are not required. If a method is not defined for a certain tracker, the application ignores it.

The logic defined for each of the above methods is specific to the tracker service. For example, the logic used for the trackPageview method in GoogleAnalytics.js is slightly different than the logic used in GoogleUniversalAnalytics.js. When implementing a 3rd party analytics solution you will need to create

your own custom logic according to the needs of that service. For example, the following code snippet shows a new CustomAnalytics module with a trackPageview method where the custom logic required for the CustomAnalytics service should be defined.

```
define('CustomAnalytics', function ()
   {
      var CustomAnalytics = {

         trackPageview: function (url)
         {
            [INSERT CUSTOM LOGIC FOR THE SERVICE BEING USED]
         }

            return this;
         }
```

You can also create your own custom methods as needed and add the method to the ApplicationSkeleton.Tracking.js file. The method is then called for each tracking service where that method is defined.

### To implement a custom analytics module:

1. Create a copy of the existing GoogleUniversalAnalytics module in the modules folder at js > src > app > modules and rename as appropriate.
2. Redefine the GoogleUniversalAnalytics variables.
3. For each tracking method (`trackPageview`, `trackEvent`, `addItem`, etc.), edit the code as needed for your own solution.
4. In the `mountToApp` method, push your new tracker to the list of existing trackers.

   ```
   application.trackers.push(MyCustomAnalyticsSolution);
   ```

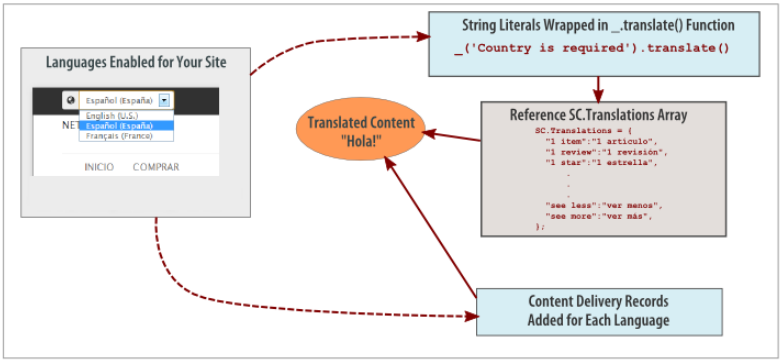
   **Important:** Do not miss this step! If the tracker is not defined in the `mountToApp` method, the ApplicationSkeleton.Tracker.js file does not know the tracker exists and no tracking is done for that module.

5. Add your new module to the list of loaded modules in the Configuration.js file at js > src > app.

# Handling Localization in Reference Implementations

This section describes how to configure and manage translated components from Reference Implementations for your Web Store. Reference Implementations support multiple languages out of the box and can be customized to add translated components.

The process of displaying translated components on a Web Store involves enabling languages for your site, wrapping new literals within a `_.translate()` function, adding these new translation name-value pairs to the `SC.Translations` array for each language, and providing Content Records with translated content generated by the Content Delivery bundle.




> **Note:** In addition to the localization elements handled by the client there are many elements handled from the NetSuite backend. These elements can NOT be customized or extended. For example, NetSuite error messages, numbers for prices and dates, etc are localized by the back end NetSuite code.

Read the following topics for detailed information on working with localized content in your site:

- Setting Up Localization for Your Web Site
- Understanding the _.translate() Function
- Localized Content from the Content Delivery Service

# Setting Up Localization for Your Web Site

## To enable your Web Site for localized content:

1. Verify that languages are defined in the **Languages** subtab of the Company General Preferences at Setup > Company > General Preferences.
2. Verify that item fields have translation records associated with them.

   When translation fields are populated in the item record under System Administration > Translation, the corresponding fields that display in the Web Store are also translated.

   > **Note:** Translation of custom fields on item records is NOT currently supported. Any data returned from custom fields for items in a Web Store will NOT have corresponding translated strings.

   For more details on entering translated elements on item records see the help topic Multi-Language Names and Descriptions.
3. Enable languages for your website.
   1. Go to Setup > SuiteCommerce Advanced > Set Up Web Site.
   2. Click **Edit** for the desired website and navigate to the **Shopping** tab and then the **Languages** subtab.
   3. Enable the desired languages in the **Online** column.

      Only languages that have been enabled in your Company General Preferences are available.

Once your Web Site is set up to support languages as described above, the language drop-down selector is available by default in the upper left region of your Web Site.

# Understanding the _.translate() Function

Within each Reference Implementation there are many string literals that are translated client-side using the `_.translate()` function. When a specific language is set for a website, string literals wrapped in any `_.translate()` functions are replaced with language specific strings. These strings are retrieved from name-value pairs defined in the `SC.Translations` arrays within the JavaScript files located in the languages directory.

The languages directory is at *Web Site Hosting Files* > *Live Hosting Files* > *SSP Applications* > *NetSuite Inc. - Shopping* > *Reference ShopFlow* > *languages* (where *NetSuite Inc. — Shopping* is the name of the Reference Implementation).

> **Note:** If a literal wrapped in a `_.translate()` function does NOT have a corresponding name-value pair in the `SC.Translations` array for the active language, that literal is NOT replaced.

For example, in the following code the string literal 'Country is required' is translated to 'Se requiere pais' when the country set for the site is Spanish. This correlates to the JSON name-value pair defined in the `SC.Translations` array of the es_ES.js file within the languages folder.

**_.translate() function used in Address.Model.js file:**

```
validation: {
        ...
      ,   country: { required: true, msg: _('Country is required').translate() }
        ...
      }
```

**Corresponding JSON in languages > es_ES.js:**

```
SC.Translations = {
      ...
"Continue Shopping":"Continuar comprando",
"Continue Shopping on our $(0)":"Continuar comprando en nuestro $(0)",
"Country is required":"Se requiere paнs",
```

```
"Credit Card Number is invalid":"Número de tarjeta de crédito no válido",
"Credit Card type is not supported":"No se admite el tipo de tarjeta de crédito",
      ...
};
```

**Note:** Values from the `SC.Translations` arrays are passed off to a translation function. Therefore, there may not always be a *direct* relationship between the values set in the `SC.Translations` array for a certain language and the actual values displayed in your Web Store.

## Passing Parameters and HTML

In addition to passing string literals to the `_.translate()` function, parameters and HTML segments can also be passed.

- **Passing Parameters:** For example, in the following code the variable values of `$(0)` and `$(1)` are passed 20 and 43 respectively.

```
_(''Subtotal: ($(0) of $(1))'').translate(20,43)
```

- **Passing HTML:** For example, in the following code an HREF segment is passed to the translate function so that the translated literal string is followed by the correct URL.

```
_('Continue Shopping on our $(0)').translate('<a href="/" data-touchpoint="home">' + _('Home Pa
ge').translate() + '</a>')
```

**Important:** When customizing your applications with new `_.translate()` functions using HTML, ensure that your translation vendor does NOT translate the HTML code segments! Only the literal strings should be translated.

## Working with Custom String Literals

You can customize existing string literals to change the default language values and corresponding translated values. Extend the application to change the string literals in the code and update the name-value pairs in the existing `SC.Translation` arrays.

You can also add new string literals that are translated at runtime. When extending an application you must ensure that new string literals you intend to localize are wrapped in the `_.translate()` function *and* that the corresponding language files are updated. If you have a small amount of customization to do, you can manually add the newly translated name-value pairs to the existing `SC.Translations` arrays for each language. However, if your Web Store includes many new string literals, we recommend that this process be scripted. Each time you compile your Web Site code all string literals wrapped in the `_.translate()` function should be processed to generate the appropriate name-value pairs in each language file.

**Important:** These language files MUST use **UTF–8** character encoding to ensure proper display of international characters!

# Localized Content from the Content Delivery Service

Reference Implementations support the locale tag so that you can deliver localized content generated from the Content Delivery bundle.

### To localize content generated from the Content Delivery bundle:

1. Create a new Content Record for each localized version of the content to be displayed by the record.

   For details on how to create a Content Delivery record see Creating a Content Record.

2. Tag each record with the corresponding locale attribute.

   For example, to specify a record as the English version use the tag **locale:EN_us**.

The locale attribute MUST exactly match the language/locale parameters set by the system when languages are enabled. To view the exact nomenclature, you can right-click on the language drop-down menu and select Inspect Element. In the browser debugger the nomenclature is listed for each language enabled.

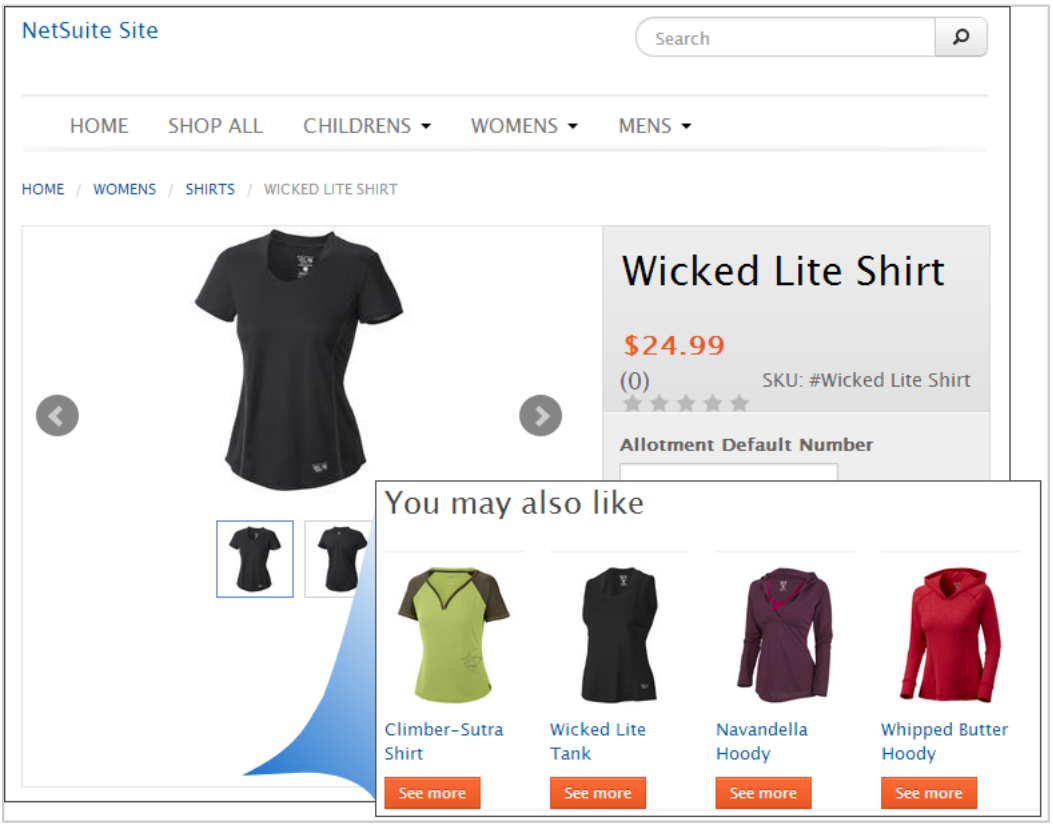
```
<!-- MACRO STARTS: languageSelector -->
▼<div class="input-prepend">
  ▶<span class="add-on">…</span>
  ▼<select data-toggle="lenguage-selector" class="input-medium">
     <option value="en_US" selected>
                English (U.S.)
            </option>
     <option value="es_ES">
                Español (España)
            </option>
     <option value="fr_FR">
                Français (France)
            </option>
   </select>
 </div>
<!-- MACRO ENDS: languageSelector -->
```

# Correlated & Related Items

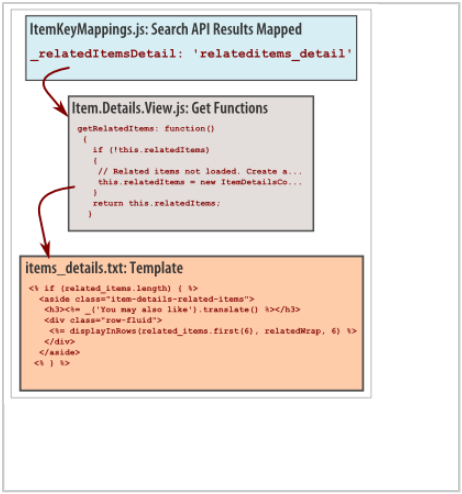
By default, when an item has correlated and related items associated with an item on the item record, Reference ShopFlow displays those correlated and related items on the item product details page within your site and within the View Cart page. You can specify how many related items and correlated items to display.

By default the correlated and related items are not displayed within the View Cart page for Site Builder sites. You can modify this default behaviorr by uncommenting the `session.getRelatedItems(items_by_id[item.internalid])` and `session.getCorrelatedItems(items_by_id[item.internalid])` functions within the Model.js file at ssp_libraries > Models.js file.

**Note:** Displaying correlated and related items in the Cart for Site Builder sites may cause performance issues.




There are three files involved in handling correlated and related items. The ItemKeyMapping.js file maps Search API information that is used by the related item functions within the Item.Details.View.js file. The item_details.txt file is then used to for rendering of the results.



# Prerequisites

## Add Related Items on Item Records

Correlated Items for an item record are determined based on purchasing history and are stored on each Item record automatically as read-only information. Related Item definitions must first be added to Item records in order for them to display in a web store. To add related items, go to the Web Store > Related Items tab on the desired Item record at Lists > Web Site > Items. For detailed information see the help topic Related Items.

## Define Required Field Sets

**Important:** If you are using Reference Checkout with a Site Builder site, the required Fields Sets are defined in the backend.configuration.js file. Normally, you do not need to do additional configuration in order for the related and correlated items to display. For more information, see Configuring Backend Objects Using the backend.configuration.js File.

In the ItemsKeyMapping.js file at Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Shopping > Reference ShopFlow > js > src > app, three related items mappings are defined to map information returned from the Item Search API. The following snippet shows the three mappings.

```
// Related items
   ,   _relatedItems: 'related_items'

      // Related items in the PDP.
   ,   _relatedItemsDetail: 'relateditems_detail'

      // Correlated (Upsell) items in the PDP.
   ,   _correlatedItemsDetail: 'correlateditems_detail'
```

To ensure that the necessary information from the Search API is available, the following **required** Field Sets must be defined in your Web Site Set Up page. For more information on Field Sets see the help topic Field Sets.

**Note:** These field sets are automatically defined when you use the Setup Script. See Using a Setup Script to Define Field Sets.

- **details**:
  - correlateditems_detail: Correlated Items (Detail)
  - relateditems_detail: Related Items (Detail)
- **correlateditems**: all the same fields as included for search
- **relateditems**: all the same fields as included for search

**Note:** For search fields required, see Minimum Setup Requirements.

# Specifying the Number of Related Records to Display

By default the first **six** related or correlated items are displayed in a single row. This is configured in the item_detail.txt template. To change the default value, edit the `related_items.length` or `correlated_items.length` variables at Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Shopping > Reference ShopFlow > templates > item_details.

**Note:** This file can NOT be edited directly from the Reference ShopFlow folder. Copy the file to the same location in the Custom ShopFlow SSP application folder and edit the file there. For see Using the Custom Folder.

The first parameter sets the number of items to display and the second parameter sets the number of rows.

For example:

- `<%= displayInRows(related_items.first(6), relatedWrap, 6) %>`: displays a single row of 6 items.
- `<%= displayInRows(related_items.first(8), relatedWrap, 4) %>`: displays two rows with 4 items in each row.

```
<%
   var related_items = view.getRelatedItems()
   ,   correlated_items = view.getCorrelatedItems();
%>
.
.
.
   <%
      if (related_items.length || correlated_items.length)
      {
         //If any of the collections has any items, define the wrapper
         var relatedWrap = function relatedWrap (item)
         {
            return SC.macros.relatedItem(item, application);
         };
      }
   %>
   <% if (related_items.length) { %>
      <aside class="item-details-related-items">
         <h3><%= _('You may also like').translate() %></h3>
         <div class="row-fluid">
            <%= displayInRows(related_items.first(6), relatedWrap, 6) %>
         </div>
      </aside>
   <% } %>
   <% if (correlated_items.length) { %>
      <aside class="item-details-correlated-items">
         <h3><%= _('Customers who bought this item also bought').translate() %></h3>
         <div class="row-fluid">
            <%= displayInRows(correlated_items.first(6), relatedWrap, 6) %>
         </div>
      </aside>
   <% } %>
```

# List Headers

Reference Implementations include a List Header component that defines sorting and pagination options used in pages that render lists of items or records. The List Header component can be used within different views and interacts with the collection displayed within that view. It is reusable and configurable to suit a variety of uses.

Available default sorting options include:

- Sort by Date
- Sort by Amount
- Sort by Number
- Filter by Date
- Reverse Order

> **Note:** When an option is selected by the shopper the URL is also updated to reflect the sorting and filtering criteria.

For details, see the following topics:

- Setup
- Upgrading from Previous Version
- Customizing List Headers

# Setup

- Prerequisites
- Feature Architecture
- Configuration Options

## Prerequisites

The List Header feature is available by default with My Account 1.03 and higher. List Header is not currently available by default with ShopFlow 1.04 or Checkout 2.0.

## Feature Architecture

The List Header component is a Backbone View that appends itself to any view that instantiates it. The initialization sets the collection, the parent view, and the options with available parameters for the List Header. For each page, a collection JavaScript file defines an `update` method called by the List Header. On the backend, a SuiteScript Server page defines the `search` method and a backend model accesses the SuiteScript API directly to make the necessary `nlobjSearchFilter` calls. The List Header renders itself on `afterViewRender`.

The primary files used to define the behavior of the List Header include:

- **list_header.txt**: Defines the template used for rendering the header. Each available option is defined in a conditional statement and only rendered when that option is enabled in the requesting page.
- **ListHeader.js**: Provides the logic for listening and responding to user events on header objects. If List Headers are not defined by default for a collection you are implementing, you must also edit the related pages to include necessary methods for List Header processing. See List Headers
- **list-Header.less**: Contains all of the CSS related to List Headers.

### Options

When instantiating the List Header, the following parameters MUST be provided:

- **view**: The parent view that contains it.
- **collection**: The collection that will interact with, used to fetch the new data.
- **application**: The application that contains it, used for the `killerId`.

Depending on the functionality desired you may also include:

- **filters**: Array of filter options, each option object contains name, and value. If one has the attribute selected set to true, that option will be the selected by default.
- **sorts**: Same as filter but for sorting options.
- **selectable**: If set to true, then select and unselect all check-box will display. This currently calls the method on the parent view.
- **classes**: Extra class names for the List Header instance.
- **rangeFilter**: Input type for the Range Filters.

For example, in the TransactionHistory.Views.js file the following initialization function defines the parameters used in the List Header for that page.

```
,   initialize: function (options)
   {
      var application = options.application;

      this.application = application;

      this.listenCollection();

      // manges sorting and filtering of the collection
      this.listHeader = new ListHeader({
         view: this
      ,   application: application
      ,   collection: this.collection
      ,   filters: this.filterOptions
      ,   sorts: this.sortOptions
      ,   rangeFilter: 'date'
      });
   }
```

## Collection Update Method

Each collection that includes a List Header has an `update` method that receives the currently applied parameters in the List Header initialization. This method is also called whenever the List Header options are changed by the shopper. The `update` method is not a default Backbone method. It is uniquely defined for each collection shipped with Reference Implementations that includes a List Header and must be defined for each new custom collection you create.

### Example: Transaction History

For example, in the Transaction.Collection.js file the native Backbone method `fetch` is called to synchronize the frontend collection to the backend collection while filtering for each selected attribute.

```
,   update: function (options)
   {
      var range = options.range || {};

      this.fetch({
```

```
        data: {
           filter: options.filter.value
        ,   sort: options.sort.value
        ,   order: options.order
        ,   from: range.from ? new Date(range.from.replace(/-/g,'/')).getTime() : null
        ,   to: range.to ? new Date(range.to.replace(/-/g,'/')).getTime() : null
        ,   page: options.page
        }
     ,   reset: true
     ,   killerId: options.killerId
     });
  }
```

**Note:** A `killerId` must be passed with each List Header AJAX request to ensure that when a user navigates away from the page or triggers a second AJAX request, the first AJAX request is cancelled.

The `fetch` method calls a NetSuite service, transaction-history.ss, where a `search` method is defined with all parameters passed to it.

```
switch (method)
{
   case 'GET':
      Application.sendContent(TransactionHistory.search({
         filter: request.getParameter('filter')
      ,   order: request.getParameter('order')
      ,   sort: request.getParameter('sort')
      ,   from: request.getParameter('from')
      ,   to: request.getParameter('to')
      ,   page: request.getParameter('page') || 1
      }));
   break;
```

Finally, the Models.js file uses the SuiteScript API to query for data based on the filter criteria selected.

## Configuration Options

In order to configure the filtering, sorting and other options to be displayed in a List Header, you must customize your application. There are no specific configuration options available for filtering, sorting, etc within a List Header. In order to modify the options available in any View, you need to customize your application. See Customizing List Headers.

### Configuring Pagination

In the Configuration.js file you can set configuration options for default pagination. These setting are used for all pages across a site.

```
,   defaultPaginationSettings: {
      showPageList: true
   ,   pagesToShow: 9
   ,   showPageIndicator: false
   }
```

The following options are available:

- **showPageList**: sets whether the pages list portion of the List Header component is displayed or not. Defaults to true.
- **pagesToShow**: sets the number of pages you want to be displayed in the pages list. Defaults to 4.
- **showPageIndicator**: sets whether an indicator of the current page is displayed and how many page in total there are. Defaults to false.

## Upgrading from a Previous Version

List Headers is a new feature as of Reference ShopFlow 1.05. No special upgrade requirements specific to this module are needed.

## Customizing List Headers

**Note:** When customizing reference implementations, the files mentioned in this section can NOT be edited directly from the reference application. Copy the file to the same location in the Custom SSP application folder and edit the file there. For details, see Using the Custom Folder.

### Modifying Existing List Headers

In most cases, modifying the List Headers involves adding or deleting available options. To modify the set of available options available for a certain page, customize the **View** for that page. The initialize method within the view lists all available parameters for that view. You can add or delete parameters as desired.

**Note:** If you are changing the available options for a certain View, you do not need to edit the backend service files or models. Editing the frontend View is sufficient. If you need to change the SuitesScript logic, you will need to customize the corresponding service files and model for that view.

### Adding List Headers to Pages

In order to customize an existing page (or a new custom page you have created) to include a List Header there are four components that must be considered:

- **View**: the view for the page is used to instantiate the List Header and define the options to be available.
- **Collection**: the collection must include an `update` method.
- **Service**: a NetSuite service file defines the `search` method for the List Header.
- **Model**: the model contains all of the SuiteScript API calls used to retrieve the selected data.

**Important:** For more detailed information on the architecture you should use when implementing your own List Headers, see Feature Architecture for List Headers. A good example to leverage is the Transaction History feature.

# Site Management Tools Templates

Several of the Reference ShopFlow template files contain pre-defined Site Management Tools areas that enable you to quickly add content to your site without the need for creating custom template files. The template files with pre-defined areas include the following.

- Page Header and Footer Templates
- Home Page Template
- Facet Search Page Template
- Item Details Template

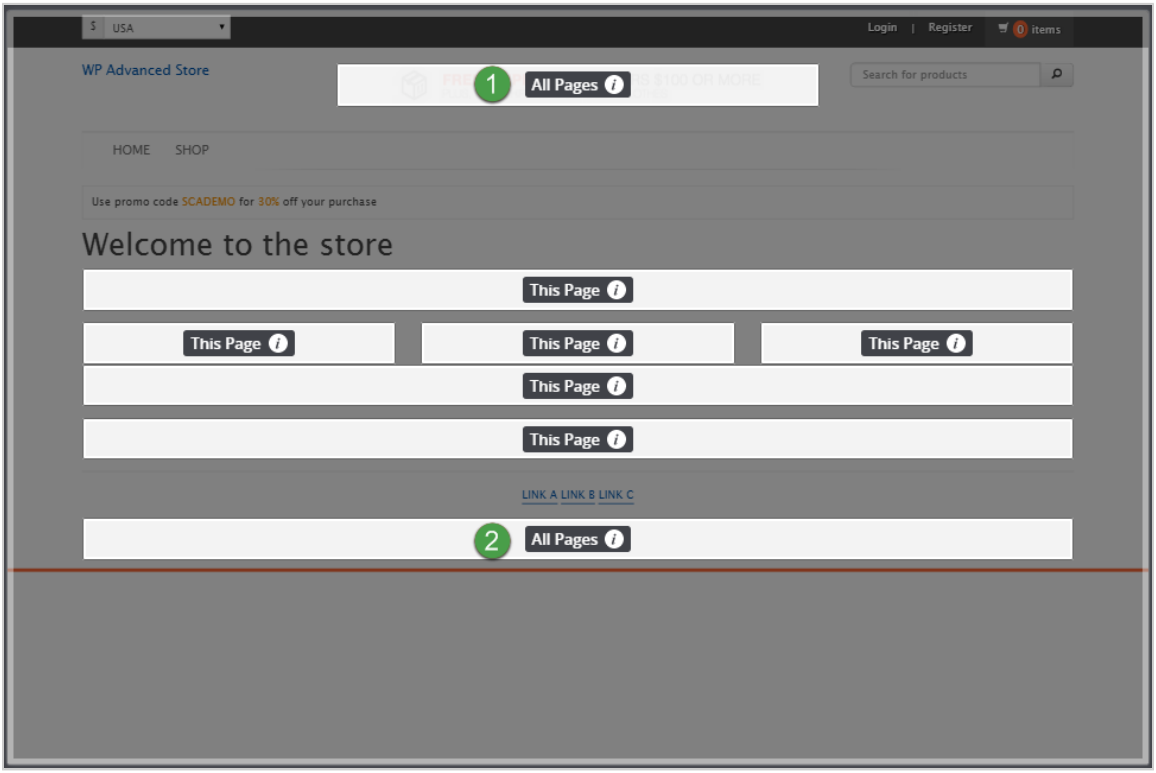
If you determine that the default template files do not meet your needs you can customize them to add, edit, or delete any of the pre-defined areas. If you already have customized template files, you can add areas to those files instead. For more information on Site Management Tools Areas, see the help topic Site Management Tools Areas For more information on customizing template files see Customizing Template Files.

> **Note:** These areas are available by default in template files for ShopFlow 1.07 and later. To take advantage of Site Management Tools in an existing Reference ShopFlow 1.06 implementation, you can migrate changes into that implementation. For full details refer to the release notes, Release Notes Prior to SuiteCommerce Advanced Denali.

# Page Header and Footer Templates

The layout of the header and footer areas of your site pages are determined by the `header_macro.txt` and `footer_macro.txt` files. By placing Site Management Tools areas in these template files you define areas in your page header and footer. The scope of these pre-defined areas is **All Pages** which means any content you add to these areas displays across all of your shopping pages. These pre-defined areas simplify the process of adding content that you want to appear at the top or at the bottom of every page.

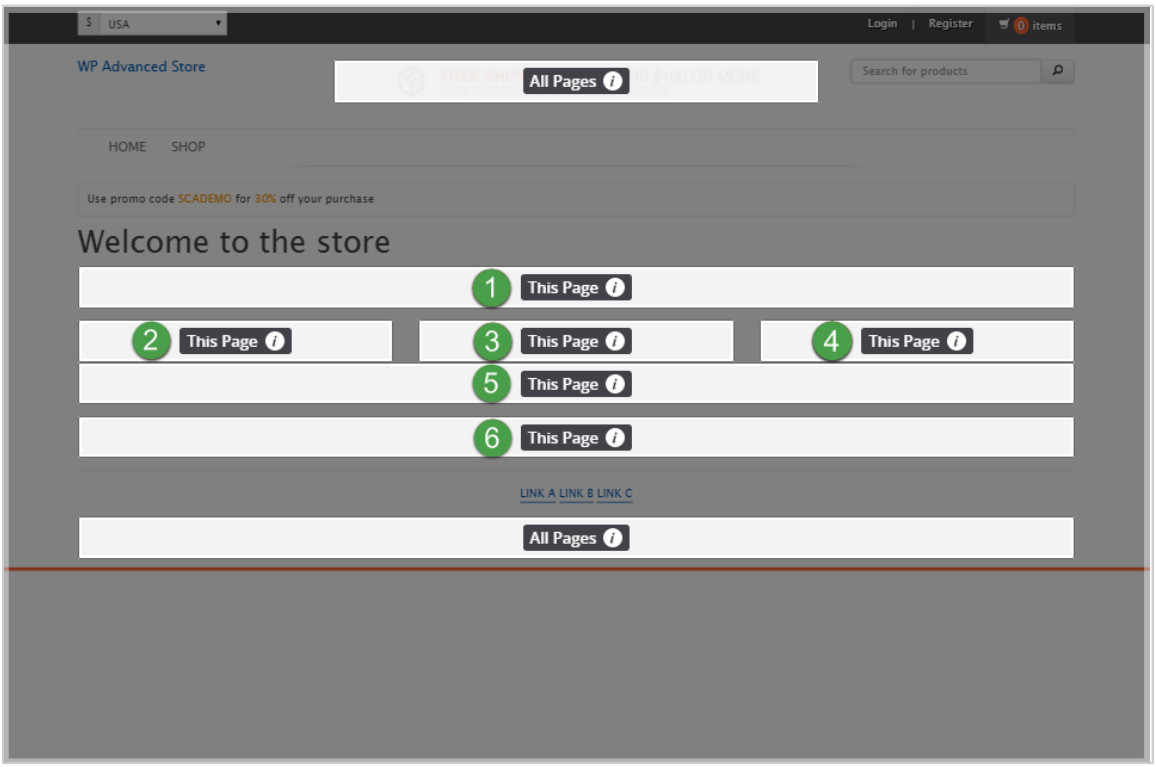
When a web page on your site is displayed, the page contains not only the areas defined in the page header and footer but also those areas defined by the page's template file. In the following screen shot, you see the header and footer areas as they are displayed on the home page when adding content. These same areas also display on the search page and item detail pages.



1. The header template file is `header_macro.txt` and it contains one area named `header_banner` with **All Pages** scope.
2. The footer template file is `footer_macro.txt` and it contains one area named `home_bottom` with **All Pages** scope.

## Home Page Template

The layout of your site's home page is determined by the `home.txt` template file. In the following screen shot you see the areas defined in the home page template file. You also see the header and footer areas at the top and bottom of the page.

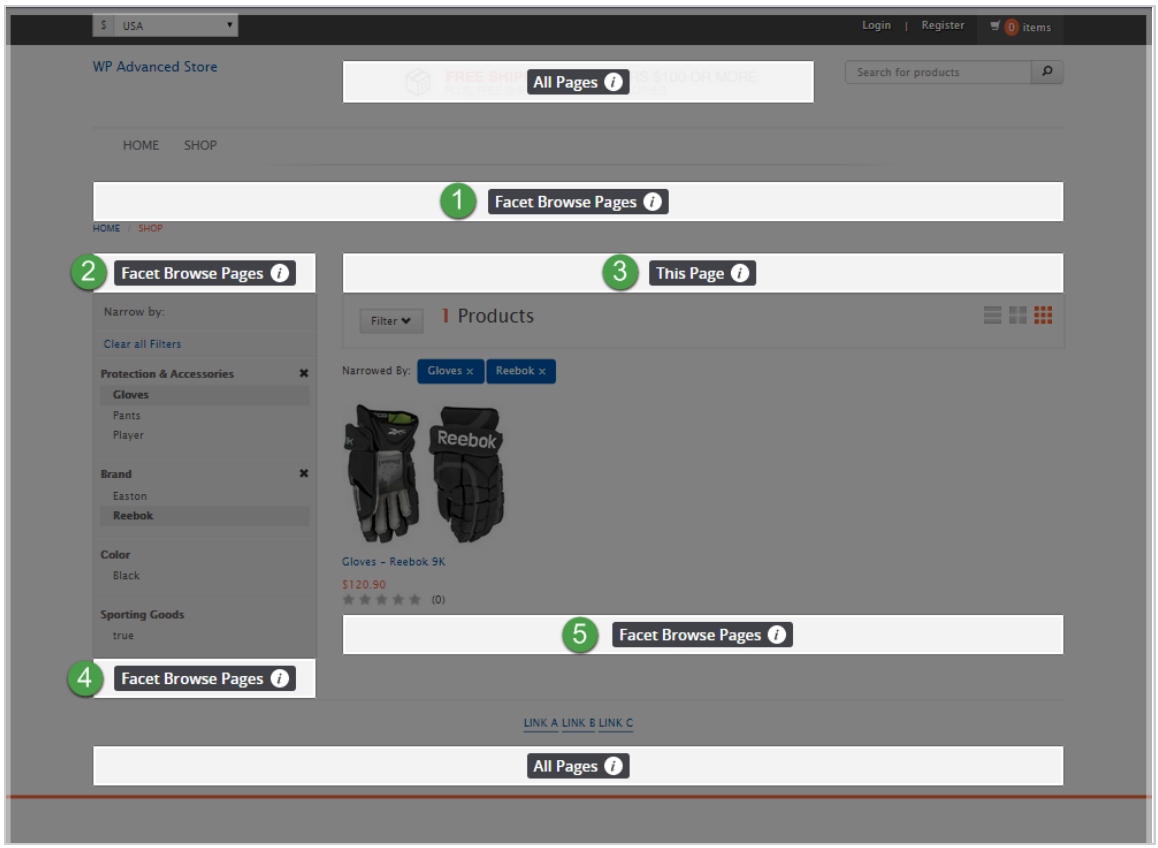



The `home.txt` template file contains the following **This Page** areas.

1. `home_banner_top`
2. `home_main`
3. `home_banner_1`
4. `home_banner_2`
5. `home_banner_3`
6. `item_list_banner_bottom`

## Facet Search Page Template

The layout of your site's search page is determined by the `facet_browse.txt` template. As you can see in the following example, the search page includes five pre-defined areas. The search page also includes the page header and footer areas.

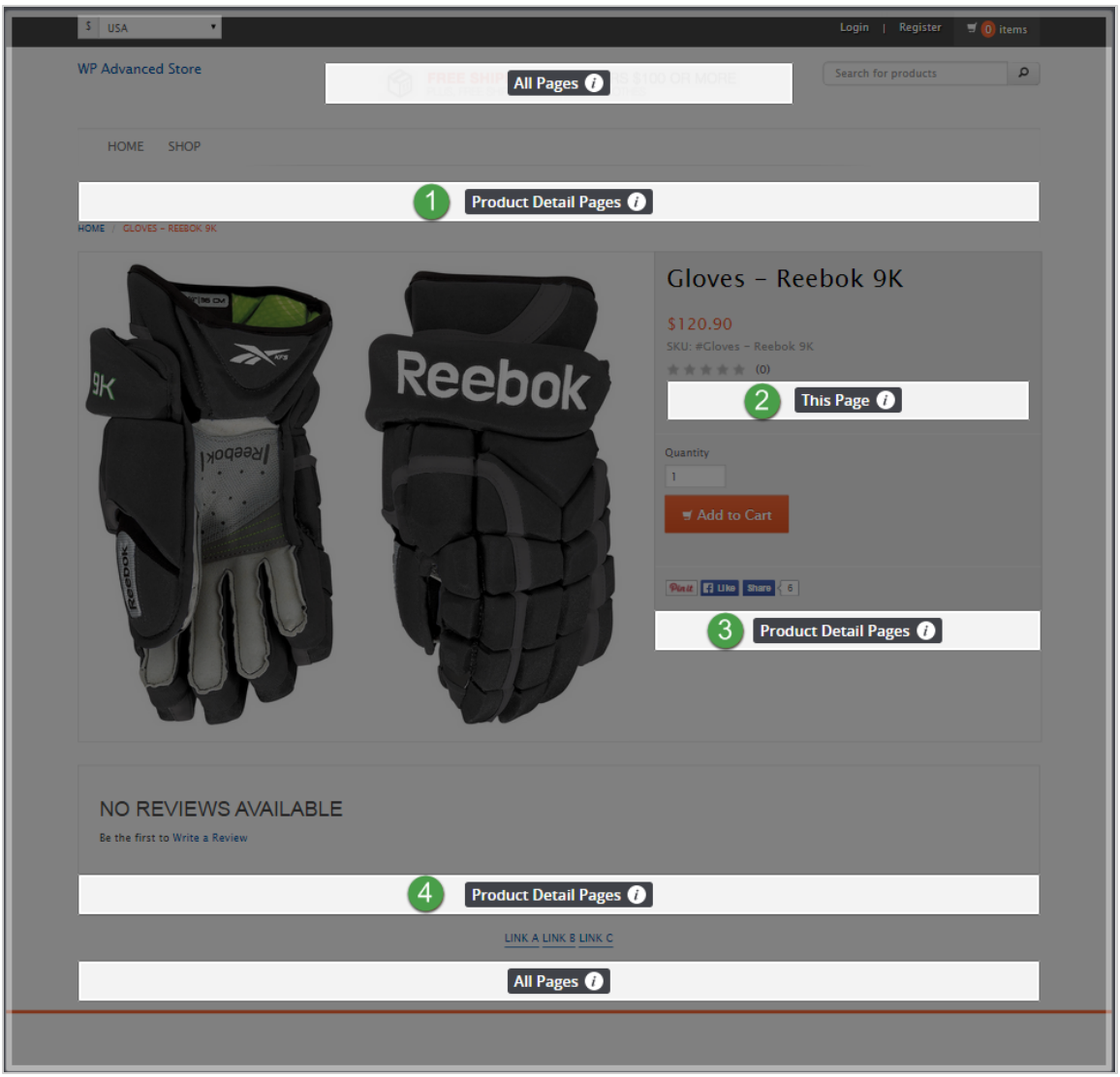



The `facet_browse.txt` template file contains 4 **Page Type** areas and one **This Page** area.

1. Page Type — `item_list_banner`
2. Page Type — `facet_navigation_top`
3. This Page — i `tem_list_banner_top`
4. Page Type — `facet_navigation_bottom`
5. Page Type — `item_list_banner_bottom`

## Item Details Template

The layout of the item detail pages on your site is determined by the `item_details.txt` template file. As you can see in the following example, the item detail page includes four pre-defined areas. The page also includes the page header and footer areas.




The `item_details.txt` template file contains 3 **Page Type** areas and one **This Page** area.

1. Page Type — `item_details_banner`
2. This Page — `item_info`
3. Page Type — `item_info_bottom`
4. Page Type — `item_details_banner_bottom`

## Customizing Template Files

When modifying template files, you can use the Custom Folder precedence overlay method. This customization method involves placing copies of the files to be customized, in this case the template files, into their corresponding location in the Custom Shopflow folder in the file cabinet. The file path location in the custom folder mirrors the path of the file's original location in the Reference Shopflow folder.

For Site Management Tools, this means that you copy the desired template files from the Reference Shopflow folder to the corresponding Custom Shopflow folder. You then modify the template files in

Custom Shopflow folder to add your customizations. This results in a precedence overlay in which the SSP application first checks the Custom folder. Consequently, the Reference Shopflow implementation uses the customized files when building the web pages.

It is necessary to make the changes to the files in the Custom Shopflow folder, because the files in the Reference Shopflow folder are locked and cannot be edited.

| Template File | Reference Shopflow File Cabinet Location | Custom Shopflow File Cabinet Location |
|---|---|---|
| home.txt | Web Site Hosting Files > Live Hosting Files > SSP Applications > [Reference Implementation] > Reference Shopflow > templates > home | Web Site Hosting Files > Live Hosting Files > SSP Applications > [Reference Implementation] > Custom Shopflow > templates > home |
| item_details.txt | Web Site Hosting Files > Live Hosting Files > SSP Applications > [Reference Implementation] > Reference Shopflow > templates > item_details | Web Site Hosting Files > Live Hosting Files > SSP Applications > [Reference Implementation]> Custom Shopflow > templates > item_details |
| facet_browse.txt | Web Site Hosting Files > Live Hosting Files > SSP Applications > [Reference Implementation]> Reference Shopflow > templates > items_list | Web Site Hosting Files > Live Hosting Files > SSP Applications > [Reference Implementation] > Custom Shopflow > templates > items_list |
| header_macro.txt | Web Site Hosting Files > Live Hosting Files > SSP Applications > [Reference Implementation]> Reference Shopflow > templates > header > macros | Web Site Hosting Files > Live Hosting Files > SSP Applications > [Reference Implementation] > Custom Shopflow > templates > header > macros |
| footer_macro.txt | Web Site Hosting Files > Live Hosting Files > SSP Applications > [Reference Implementation]> Reference Shopflow > templates > footer > macros | Web Site Hosting Files > Live Hosting Files > SSP Applications > [Reference Implementation] > Custom Shopflow > templates > footer > macros |

**Note:** If your site already has one or more customizations, your template files may have already been added to the Custom Shopflow folder, and you can add the areas to those custom template files. If the template files have not been copied to the Custom Shopflow folder, you need to copy the files to the Custom Shopflow folder and then add your customizations.

# Bronto

Bronto is a NetSuite company that provides an advanced marketing automation engine and solutions for shopping cart abandonment, post-purchase campaigns and so forth. Bronto can be easily integrated with your Reference ShopFlow, My Account, and Checkout implementations.

The following Bronto Applications are supported:

- Cart Recovery
- Conversion Tracking
- Coupon Manager
- Pop-up Manager

To implement Bronto integration, you need to the modify the Configuration.js file for each bundle to include your `accountId`.

```
,   bronto: {
      accountId: ''
   }
});
```

With the `accountId` configured, the BrontoIntegration module causes a configuration.js file to load after the SuiteCommerce Application has initialized. This configuration.js file is cached in the Bronto CDN and uses the `accountId` to pull down the appropriate settings to enable and configure the various integrations setup in the Bronto Connector.

**Note:** The BrontoIntegration module essentially just provides an interface to the Bronto CDN. Therefore, any updates or enhancements to Bronto applications are completely separate from SuiteCommerce Advanced releases and will be made available through the Bronto CDN. There is no additional customization needed in the SuiteCommerce source code to take advantage of Bronto enhancements.

For detailed information on configuring and managing your Bronto applications, refer to the Bronto documentation.

# Reference SuiteApp FAQ's

## Is it possible to have different Reference Checkout experiences per site?

Yes. You need separate hosting roots for the different versions of the application though. For more information, see Derivative Hosting with SuiteCommerce Advanced.

## Can I customize the background color differently for each site? (without elaborate custom code... just different CSS files)

Yes, same as above.

## When you deploy a bundle to multiple sites, do all sites share the same code?

You do not actually deploy a bundle to a site, you configure touch points on an SSP application. The SSP application is installed by a bundle, and it contains an .ssp file with custom code. If you deploy the SSP application to multiple sites, all sites point to the same .ssp file and the sites share the same code.

## At a site level (on the Web Site Setup page), is there a place where I can see all bundles that have been deployed to that particular Site?

On the Touch Points tab on the Web Site Setup page, you can see which .ssp file is associated with each touch point. You can also see the SSP Application associated with each .ssp file.

## For a specific bundle (i.e, Reference Checkout), can I see all sites that the bundle has been deployed to?

Note that the bundle is a container that installs all the assets needed to run the SSP application on your site. You can find the site where a specific SSP Application is deployed by going to the Domains setup page (Setup > SuiteCommerce Advanced > Domains).

### How do I 'un-deploy' a bundle from one site only? Removing the touch points from within Set Up Web Site will hide the reference implementation, but can you 'un-deploy' a bundle from a single site other than un-installing it?

Removing the touch points is the equivalent of un-deploying, not uninstalling. The bundle still exists in the account in full, it won't be connected to that particular site anymore.

# Secure Shopping Domain (pre-Denali)

A secure shopping domain creates an encrypted connection between your web server and your customer's web browser. You can use an SSL certificate to secure the shopping portion of your web store under an HTTPS domain. HTTPS is currently the industry standard for ecommerce services and consumers prefer seeing the secure icon in their browser address bar.

Including secure technology in your shopping area assures your customers that their activities on your site are safe. In addition, search engines tend to rank secure sites higher than non-secure sites.

There are two new SuiteCommerce API methods that help make HTTPS support possible. These methods distinguish between the checkout, shopping, and single domains.

- `isCheckoutSupported()`: returns true if we are in the checkout domain or in a single domain
- `isShoppingSupported()`: returns true if we are in the shopping domain or in a single domain

`!~request.getURL().indexOf('https')` is the legacy domain check and it will no longer work to distinguish between the domains.

The `session.isLoggedIn()` method is the legacy session check for a user's logged in status. This method has been replaced with the new `session.isLoggedIn2()` method in later versions of SuiteCommerce Advanced. The new session check method is required in this migration to help maintain the identity of the user.

To leverage a secure shopping domain in a pre-Denali version of SuiteCommerce Advanced, you need to:

- Replace all instances of the legacy domain check with the new domain check methods.
- Replace all instances of the legacy session check with the new session check method.

## Migration Tasks

This migration is highly technical and involves customizing multiple files. Update the following files in each SSP Application to use the new domain check methods as well as the new session check method where applicable:

- commons.js
- Utils.js
- Models.js
- sc.environment.ssp
- index.ssp and index-local.ssp
- jQuery.ajaxSetup.js
- live-order.ss and live-order-line.ss
- CheckoutSkipLogin.js
- NavigationHelper.js

In addition, complete these tasks.

- Update Legacy Checks and Methods
- Deploy the Migration

**Important:** You **must** make the following modifications to SuiteCommerce Advanced in a **NetSuite 2017.1** account.

This document is intended to be a guide only. It is not a comprehensive procedure that guarantees a correct implementation of HTTPS support for your web store.

### To modify files for HTTPS:

**Important:** Before you begin making changes, back up any custom files that already exist. Be careful not to overwrite existing custom files.

For each file, go to the SSP Application under Documents > Files > File Cabinet > Web Site Hosting Files > Live Hosting Files > SSP Applications and copy the Reference files to the same location in the Custom folder.

1. Open each file at the specified location in the Custom folder.
2. Search each file for the appropriate location using the line number or surrounding text from the code snippet.

   Use the following conventions to determine when you need to add or remove code.

   ```
   --- = Remove this line of code.
   +++ = Add this line of code.

   Note: Surrounding text and line numbers vary depending on each SSP Application as well as exist
   ing customizations. They are included as reference points only.
   ```

3. Add or remove statements as indicated in the code snippet and save.

## commons.js

File locations:

- ShopFlow > Custom ShopFlow > ssp_libraries > commons.js
- MyAccount > Custom MyAccount > ssp_libraries > commons.js
- Checkout > Custom Checkout > ssp_libraries > commons.js

Add the entire `Utils` variable to the Utilities section of each `commons.js` file. The `Utils` variable contains the new methods that distinguish between the domains.

```
...

    // Utilites
+++         var Utils = {
+++
+++             // @method isCheckoutDomain determines if we are in a secure checkout
+++             // domain or in a secure single domain environment
+++             // @return {Boolean} true if in checkout or in single domain
+++             isCheckoutDomain: function isCheckoutDomain()
+++             {
+++                 return session.isCheckoutSupported();
```

```
+++             }
+++             // @method isShoppingDomain determines if we are in shopping domain (secure or
non secure)
+++             // or in a secure single domain environment
+++             // @return {Boolean} true if in shopping or single domain
+++         ,   isShoppingDomain: function isShoppingDomain()
+++             {
+++                 return session.isShoppingSupported();
+++             }
+++             // @method isSingleDomain determines if we are in a single domain environment
+++             // @return {Boolean} true if single domain
+++         ,   isSingleDomain: function isSingleDomain()
+++             {
+++                 return this.isShoppingDomain() && this.isCheckoutDomain();
+++             }
+++             // @method isInShopping determines if we are in shopping ssp
+++             // @return {Boolean} true if in shopping domain, false if in checkout or myacco
unt
+++         ,   isInShopping: function isInShopping (request)
+++             {
+++                 return this.isShoppingDomain() && (request.getHeader('X-SC-Touchpoint') ===
 'shopping' || request.getParameter('X-SC-Touchpoint') === 'shopping');
+++             }
+++             // @method isInCheckout determines if we are in checkout ssp or my account ssp
+++             // @return {Boolean} true if in checkout domain
+++         ,   isInCheckout: function isInCheckout (request)
+++             {
+++                 var self = this;
+++
+++                 if (!self.isSingleDomain())
+++                 {
+++                     return self.isCheckoutDomain();
+++                 }
+++                 else
+++                 {
+++                     var paypal_complete = ModelsInit.context.getSessionObject('paypal_compl
ete') === 'T'
+++                     ,   is_in_checkout = request.getHeader('X-SC-Touchpoint') === 'checkout
' ||
+++                                     request.getHeader('X-SC-Touchpoint') === 'myaccount' ||

+++                                     request.getParameter('X-SC-Touchpoint') === 'checkout'
||
+++                                     request.getParameter('X-SC-Touchpoint') === 'myaccount'
;
+++                     return self.isCheckoutDomain() && (is_in_checkout || paypal_complete);
+++                 }
+++             }
+++         }

...
```

Call the new `session.isLoggedIn2()` method to maintain user identity.

```
...
```

```
999:    function isLoggedIn ()
        {
            'use strict';
---         // MyAccount (We need to make the following difference because isLoggedIn is always
 false in Shopping)
---         if (request.getURL().indexOf('https') === 0)
---         {
---             return session.isLoggedIn();
---         }
---         else // Shopping
---         {
---             return parseInt(nlapiGetUser() + '', 10) > 0 && !session.getCustomer().isGuest(
);
---         }

+++         return session.isLoggedIn2();
        }

...
```

Modify the `getEnvironment()` method to add the new domain check methods.

```
...
        ,   init: function () {}
220:    ,   getEnvironment: function (session, request)
            {
                'use strict';
                // Sets Default environment variables
                var context = nlapiGetContext()
---             ,   isSecure = request.getURL().indexOf('https:') === 0;
+++             ,   isSecure = Utils.isCheckoutDomain();

                ,   siteSettings = session.getSiteSettings(['currencies', 'languages'])
                ,   result = {
---                         baseUrl: session.getAbsoluteUrl(isSecure ? 'checkout' : 'shopping',
 '/{{file}}')
+++                         baseUrl: session.getAbsoluteUrl(Utils.isInCheckout(request)  ? 'che
ckout' : 'shopping', '/{{file}}')
                        ,   currentHostString: request.getURL().match('http(s?)://(.*)/')[2]
                        ,   availableHosts: SC.Configuration.hosts || []
                        ,   availableLanguages: siteSettings.languages || []
                        ,   availableCurrencies: siteSettings.currencies || []
                        ,   companyId: context.getCompany()
                        ,   casesManagementEnabled: context.getSetting('FEATURE', 'SUPPORT') ==
= 'T'
                        ,   giftCertificatesEnabled: context.getSetting('FEATURE', 'GIFTCERTIFI
CATES') === 'T'
                    };
                // If there are hosts asociated in the site we iterate them to check which we a
re in
                // and which language and currency we are in
---             if (result.availableHosts.length && !isSecure)
+++             if (result.availableHosts.length && Utils.isShoppingDomain())
                {
                    for (var i = 0; i < result.availableHosts.length; i++)
```

```
...
```

## Utils.js

File locations:

- ShopFlow > Custom ShopFlow > js > src > core > Utils.js
- MyAccount > Custom MyAccount > js > src > core > Utils.js
- Checkout > Custom Checkout > js > src > core > Utils.js

Update the `getDownloadPdfUrl()` method to add the new domain check methods.

```
...
635:    function getDownloadPdfUrl (params)
        {
---         params = params || {};
---         params.n = SC.ENVIRONMENT.siteSettings.siteid;
---         var origin = window.location.origin ? window.location.origin :
---                 (window.location.protocol + '//' + window.location.hostname + (window.locat
ion.port ? (':' + window.location.port) : ''));
---         return  _.addParamsToUrl(origin + _.getAbsoluteUrl('download.ssp'), params);

+++         params = params || {};
+++         params.n = SC && SC.ENVIRONMENT && SC.ENVIRONMENT.siteSettings && SC.ENVIRONMENT.si
teSettings.siteid || '';
+++
+++         if(_.isSingleDomain())
+++         {
+++             return  _.addParamsToUrl(_.getAbsoluteUrl('download.ssp'), params);
+++         }
+++         else
+++         {
+++             var origin = window.location.origin ? window.location.origin :
+++             (window.location.protocol + '//' + window.location.hostname + (window.location.
port ? (':' + window.location.port) : ''));
+++
+++             return  _.addParamsToUrl(origin + _.getAbsoluteUrl('download.ssp'), params);
+++         }

        }

...

797:    function reorderUrlParams (url)
        {
            var params = []
            ,   url_array = url.split('?');
            if (url_array.length > 1)
            {
                params = url_array[1].split('&');
                return url_array[0] + '?' + params.sort().join('&');
            }
            return url_array[0];
```

```
        }


+++     // @method isShoppingDomain determines if we are in shopping domain (secure or non secu
re)
+++     // or single domain
+++     // @return {Boolean} true if in checkout or in single domain
+++     function isShoppingDomain ()
+++     {
+++         return SC.ENVIRONMENT.siteSettings.shoppingSupported;
+++     }
+++
+++     // @method isCheckoutDomain determines if we are in a secure checkout
+++     // domain or in a secure single domain environment
+++     // @return {Boolean} true if in checkout or in single domain
+++     function isCheckoutDomain ()
+++     {
+++         return SC.ENVIRONMENT.siteSettings.checkoutSupported;
+++     }
+++
+++     // @method isSingleDomain determines if we are in a single domain environment
+++     // @return {Boolean} true if single domain
+++     function isSingleDomain ()
+++     {
+++         return SC.ENVIRONMENT.siteSettings.isSingleDomain;
+++     }
+++
+++     // @method isInShopping determines if we are in shopping ssp
+++     // used when there are frontend features only shown in the shopping domain
+++     // @return {Boolean} true if in shopping domain, false if in checkout or myaccount
+++     function isInShopping ()
+++     {
+++         return _.isShoppingDomain() && (SC.ENVIRONMENT.SCTouchpoint === 'shopping' || SC.EN
VIRONMENT.siteSettings.sitetype === 'STANDARD');
+++     }
+++
+++     // @method isInCheckout determines if we are in checkout or my account ssp
+++     // @return {Boolean} true if in checkout domain
+++     function isInCheckout ()
+++     {
+++         return !_.isSingleDomain() ? _.isCheckoutDomain() : _.isCheckoutDomain() && (SC.ENV
IRONMENT.SCTouchpoint === 'checkout' ||  SC.ENVIRONMENT.SCTouchpoint === 'myaccount');
+++     }
...
```

## Models.js

File locations:

- ShopFlow > Custom ShopFlow > ssp_libraries >Models.js
- MyAccount > Custom MyAccount > ssp_libraries > Models .js
- Checkout > Custom Checkout > ssp_libraries > Models .js

Verify the domain using the new domain check methods.

```
...

19:     Application.defineModel('SiteSettings', {

...

121:    settings.is_logged_in = session.isLoggedIn();
        settings.touchpoints = basic_settings.touchpoints;
        settings.shopperCurrency = session.getShopperCurrency();

...

+++     settings.checkoutSupported = Utils.isCheckoutDomain();
+++     settings.shoppingSupported = Utils.isShoppingDomain();
+++     settings.isSingleDomain = settings.checkoutSupported && settings.shoppingSupported;

...
```

Call the new `session.isLoggedIn2()` method and update the `getFieldValues()` method to use the new domain check methods.

```
...

167:            Application.defineModel('LiveOrder', {

...

309:    ---     ,   isSecure: request.getURL().indexOf('https') === 0
        ---     ,   isLoggedIn: session.isLoggedIn()

        +++     ,   isLoggedIn: session.isLoggedIn2()
        +++     ,   isSecure: Utils.isCheckoutDomain()

...

681:             ,   getFieldValues: function ()
                     {
                       'use strict';
        ---             var order_field_keys = this.isSecure ? SC.Configuration.order_checkout_
field_keys : SC.Configuration.order_shopping_field_keys;
        +++             var order_field_keys = Utils.isInCheckout(request) && session.isLoggedI
n2() ? SC.Configuration.order_checkout_field_keys : SC.Configuration.order_shopping_field_keys;


...
```

Update the `verifySession()` method to remove the legacy domain check.

```
...

1726:       Application.defineModel('ProductList', {

...
```

```
            // Returns a product list based on a given userId and id
1759:       ,   get: function (user, id)
                {
---                 // Verify session if and only if we are in My Account...
---                 if (request.getURL().indexOf('https') === 0)
---                 {
---                     this.verifySession();
---                 }
+++           this.verifySession();

...

            // Retrieves all Product Lists for a given user
1896:       ,   search: function (user, order)
                {
---                  // Verify session if and only if we are in My Account...
---                 if (request.getURL().indexOf('https') === 0)
---                 {
--- ,                     this.verifySession();
---                 }

                    var filters = [new nlobjSearchFilter('isinactive', null, 'is', 'F')
                    ,   new nlobjSearchFilter('custrecord_ns_pl_pl_owner', null, 'is', user)]
                    ,   template_ids = []

...
```

## sc.environment.ssp

File locations:

- Checkout > Custom Checkout > sc.environment.ssp
- MyAccount > Custom MyAccount > sc.environment.ssp
- ShopFlow > Custom ShopFlow > sc.environment.ssp

Add a touchpoint check for Checkout and update the `session.GetAbsoluteUrl()` parameters to replace the legacy domain check.

```
...

94:     <% if (SiteSettings) { %>
            // Site Settings Info
            <%
            // under some wired cases the terms and conditions bring a script tag if there is a
 body tag present
            // This code eliminates it in the case
            var site_settings_json = JSON.stringify(SiteSettings).replace(/
>/ig, '').replace(/<\/body*?>/ig, '')
            %>
            SC.ENVIRONMENT.siteSettings = <%= site_settings_json %>;
            // Site site (ADVANCED or STANDARD)
            SC.ENVIRONMENT.siteType = '<%= SiteSettings.sitetype %>';

+++         // SCTouchpoint indicates the touchpoint the user is effectively in. We can only kn
```

```
ow with certain this in the proper ssp
+++         // because there is still code that depends on the touchpoint
+++         // when in single ssp check if this it's necessary
+++         SC.ENVIRONMENT.SCTouchpoint = 'checkout';
        <% } %>

...

179:        if(!SC.ENVIRONMENT.baseUrl)
            {
---             SC.ENVIRONMENT.baseUrl = '<%=  session.getAbsoluteUrl(request.getURL().indexOf(
'https:') === 0 ? 'checkout' : 'shopping', '/{{file}}') %>';
+++             SC.ENVIRONMENT.baseUrl = '<%=  session.getAbsoluteUrl('checkout', '/{{file}}')
%>';
            }

...
```

Add a touchpoint check for MyAccount and update the `session.GetAbsoluteUrl()` parameters to replace the legacy domain check.

```
...

106:    <% if (SiteSettings) { %>
            // Site Settings Info
            SC.ENVIRONMENT.siteSettings = <%= JSON.stringify(SiteSettings) %>;
            // Site site (ADVANCED or STANDARD)
            SC.ENVIRONMENT.siteType = '<%= SiteSettings.sitetype %>';

+++         // SCTouchpoint indicates the touchpoint the user is effectively in. We can only kn
ow with certain this in the proper ssp
+++         // because there is still code that depends on the touchpoint
+++         // myaccount value is added just in case someone needs it
+++         // when in single ssp check if this it's necessary
+++         SC.ENVIRONMENT.SCTouchpoint = 'myaccount';

        <% } %>

...

205:    if(!SC.ENVIRONMENT.baseUrl)
        {
---         SC.ENVIRONMENT.baseUrl = '<%=  session.getAbsoluteUrl(request.getURL().indexOf('htt
ps:') === 0 ? 'checkout' : 'shopping', '/{{file}}') %>';
+++         SC.ENVIRONMENT.baseUrl = '<%=  session.getAbsoluteUrl('checkout', '/{{file}}') %>';

        }

...
```

Add a touchpoint check for ShopFlow and update the `session.GetAbsoluteUrl()` parameters to replace the legacy domain check.

```
...
```

```
73:     <% if (SiteSettings) { %>
            // Site Settings Info
            SC.ENVIRONMENT.siteSettings = <%= JSON.stringify(SiteSettings) %>;

+++         // SCTouchpoint indicates the touchpoint the user is effectively in. We can only kn
ow with certain this in the proper ssp
+++         // because there is still code that depends on the touchpoint
+++         // when in single ssp check if this it's necessary
+++         SC.ENVIRONMENT.SCTouchpoint = 'shopping';

        <% } %>

...

115:  if (!SC.ENVIRONMENT.baseUrl)
      {
---         SC.ENVIRONMENT.baseUrl = '<%=  session.getAbsoluteUrl(request.getURL().indexOf('http
s:') === 0 ? 'checkout' : 'shopping', '/{{file}}') %>';
+++         SC.ENVIRONMENT.baseUrl = '<%=  session.getAbsoluteUrl('shopping', '/{{file}}') %>';
      }

...
```

## index.ssp and index-local.ssp

File locations:

- ShopFlow > Custom ShopFlow > index.ssp
- MyAccount > Custom MyAccount > index.ssp
- Checkout > Custom Checkout > index.ssp
- ShopFlow > Custom ShopFlow > index-local.ssp
- MyAccount Premium> Custom MyAccount Premium > index-local.ssp

Modify each of these files to add the `&X-SC-Touchpoint=` query parameter when requesting `sc.environment.ssp` and `sc.user.environment.ssp`.

**ShopFlow > Custom ShopFlow > index.ssp**

```
...

146:    <script>
            if (!SC.isCrossOrigin())
            {
                // Do we have SEO Support
                if (SC.isPageGenerator())
                {
                    document.body.className = document.body.className + ' seo-support';
                }
                SC.ENVIRONMENT.seoSupport = !!~document.body.className.indexOf("seo-support");
                /* load language and sc.environment.ssp */
                loadScript([
---                 '<%= session.getAbsoluteUrl("shopping", "sc.environment.ssp?lang=" + Langua
ge + "&cur=" + Currency) %>'
+++                 '<%= session.getAbsoluteUrl("shopping", "sc.environment.ssp?lang=" + Langua
```

```
ge + "&cur=" + Currency  + "&X-SC-Touchpoint=shopping") %>'
                ,   '<%= session.getAbsoluteUrl("shopping", "languages/" + Language + ".js") %>
'
                ]);
                if (SC.isPageGenerator())
                {
                    SC.ENVIRONMENT.PROFILE = {};
                }
                // Loads the application files, if you need to have a less agresive cacheing yo
u can move them
                // to the sc.environment.ssp (Moderate cacheing) or to the sc.user.environment.
ssp (No cache but less performant)
                loadScript([
                    '<%= session.getAbsoluteUrl("shopping", 'js/libs/Libraries-014c760ca5c2.js'
) %>'
                ,   '<%= session.getAbsoluteUrl("shopping", 'templates/Templates-0152658bf631.j
s') %>'
                ,   '<%= session.getAbsoluteUrl("shopping", 'js/Application-01526a8e6d58.js') %
>'
                // ,    '/cms/1/cms.js'
                ]);
                if (SC.ENVIRONMENT.jsEnvironment == 'browser')
                {
                    loadScript({
---                     url: '<%= session.getAbsoluteUrl("shopping", "sc.user.environment.ssp?l
ang=" + Language + "&cur=" + Currency) %>&t=' + new Date().getTime()
+++                     url: '<%= session.getAbsoluteUrl("shopping", "sc.user.environment.ssp?l
ang=" + Language + "&cur=" + Currency  + "&X-SC-Touchpoint=shopping") %>&t=' + new Date().getTi
me()
                    ,   async: true
                    });
                }
            }
        </script>

...
```

**MyAccount > Custom MyAccount > index.ssp**

```
...

            <script>
            var SC = window.SC = {
                ENVIRONMENT: {
                    jsEnvironment: (typeof nsglobal === 'undefined') ? 'browser' : 'server'
                }
                , isCrossOrigin: function() { return '<%= Environment.currentHostString %>' !==
 document.location.hostname; }
                , isPageGenerator: function() { return typeof nsglobal !== 'undefined'; }
                , getSessionInfo: function(key)
                {
                    var session = SC.SESSION || SC.DEFAULT_SESSION || {};
                    return (key) ? session[key] : session;
                }
            };
```

```
        </script>
65:     --- <script> src="<%= session.getAbsoluteUrl('checkout', 'sc.environment.ssp?lang=' + L
anguage + '&cur=' + Currency) %>"</script>
65:     +++ <script> src="<%= session.getAbsoluteUrl('checkout', 'sc.environment.ssp?lang=' + L
anguage + '&cur=' + Currency + "&X-SC-Touchpoint=myaccount") %>"</script>

...
```

**Checkout > Custom Checkout > index.ssp**

```
...

139:    <% if (login) { %>
---         <script src="<%= session.getAbsoluteUrl('checkout', 'sc.environment.ssp?lang=' + La
nguage + '&cur=' + Currency + (cart_bootstrap ? "&cart-bootstrap=T" : "") ) %>"></script>
+++         <script src="<%= session.getAbsoluteUrl('checkout', 'sc.environment.ssp?lang=' + La
nguage + '&cur=' + Currency + (cart_bootstrap ? "&cart-bootstrap=T" : "") + "&X-SC-Touchpoint=c
heckout" ) %>"></script>
        <% } else { %>
        <script>
            loadScript({
---             url: '<%= session.getAbsoluteUrl("checkout", "sc.environment.ssp?lang=" + Langu
age + "&cur=" + Currency + (cart_bootstrap ? "&cart-bootstrap=T" : "") ) %>&t=' + (new Date().g
etTime())
+++             url: '<%= session.getAbsoluteUrl("checkout", "sc.environment.ssp?lang=" + Langu
age + "&cur=" + Currency + (cart_bootstrap ? "&cart-bootstrap=T" : "") + "&X-SC-Touchpoint=chec
kout" ) %>&t=' + (new Date().getTime())
            });
        </script>
        <% } %>

...
```

**ShopFlow > Custom ShopFlow > index-local.ssp**

```
...

80:     /* load language and sc.environment.ssp */
        loadScript([
---         '<%= session.getAbsoluteUrl("shopping", "sc.environment.ssp?v=711&lang=" + Language
 + "&cur=" + Currency) %>'
+++         '<%= session.getAbsoluteUrl("shopping", "sc.environment.ssp?v=711&lang=" + Language
 + "&cur=" + Currency + "&X-SC-Touchpoint=shopping") %>'
        ,   '<%= session.getAbsoluteUrl("shopping", "languages/" + Language + ".js") %>'
        ]);
        if (SC.ENVIRONMENT.jsEnvironment === 'server')
        {
            SC.ENVIRONMENT.PROFILE = {};
        }
        loadScript([
            '<%= root %>templates/Templates.php'
        ,   '<%= root %>js/Application.php'
        ]);

        if (SC.ENVIRONMENT.jsEnvironment == 'browser')
```

```
        {
            loadScript({
---             url: '<%= session.getAbsoluteUrl("shopping", "sc.user.environment.ssp?lang=" +
Language + "&cur=" + Currency) %>&t=' + new Date().getTime()
+++             url: '<%= session.getAbsoluteUrl("shopping", "sc.user.environment.ssp?lang=" +
Language + "&cur=" + Currency + "&X-SC-Touchpoint=shopping") %>&t=' + new Date().getTime()
            ,   async: true
            });
        }

...
```

**MyAccount Premium> Custom MyAccount Premium > index-local.ssp**

```
...

            <script src="<%= root %>js/utils/BootUtilities.js"></script>
70: ---     <script src="<%= session.getAbsoluteUrl('checkout', 'sc.environment.ssp?lang=' + La
nguage + '&cur=' + Currency) %>"></script>
    +++     <script src="<%= session.getAbsoluteUrl('checkout', 'sc.environment.ssp?lang=' + La
nguage + '&cur=' + Currency + '&X-SC-Touchpoint=myaccount') %>"></script>

...
```

## jQuery.ajaxSetup.js

File locations:

- ShopFlow > Custom ShopFlow > js > src > core > extras > jQuery.ajaxSetup.js
- MyAccount > Custom MyAccount > js > src > core > extras > jQuery.ajaxSetup.js
- Checkout > Custom Checkout > js > src > core > extras > jQuery.ajaxSetup.js

Update the `beforeSend()` method to send the header to the backend for each request.

```
...

        // This registers an event listener to any ajax call
85:     jQuery(document)
            // http://api.jquery.com/ajaxStart/
            .ajaxStart(SC.loadingIndicatorShow)
            // http://api.jquery.com/ajaxStop/
            .ajaxStop(SC.loadingIndicatorHide);
        // http://api.jquery.com/jQuery.ajaxSetup/

        jQuery.ajaxSetup({
93:     beforeSend: function (jqXhr, options)
        {
            // BTW: "!~" means "== -1"
            if (!~options.contentType.indexOf('charset'))
            {
                // If there's no charset, we set it to UTF-8
                jqXhr.setRequestHeader('Content-Type', options.contentType + '; charset=UTF-8')
;
            }
```

```
+++         // Add header so that suitescript code can know the current touchpoint
+++         jqXhr.setRequestHeader('X-SC-Touchpoint', SC.ENVIRONMENT.SCTouchpoint);
        }

...
```

## live-order.ss and live-order-line.ss

File locations:

- ShopFlow > Custom ShopFlow > services > live-order.ss
- MyAccount > Custom MyAccount > > live-order.ss
- Checkout > Custom Checkout > services > live-order.ss
- ShopFlow > Custom ShopFlow > services > live-order-line.ss
- MyAccount > Custom MyAccount > services > live-order-line.ss
- Checkout > Custom Checkout > services > live-order-line.ss

In each file, add an `if` statement to verify if the user is in shopping or logged in.

```
...

        // If we are not in the checkout OR we are logged in
        // When on store, login in is not required
        // When on checkout, login is required
---     if (!~request.getURL().indexOf('https') || session.isLoggedIn())
+++     if (Utils.isInShopping(request) || session.isLoggedIn2())

...
```

## CheckoutSkipLogin.js

Customizing this file requires that you copy the **CheckoutSkipLogin** module folder from Reference Checkout to Custom Checkout because it is not created automatically.

Update Checkout > Custom Checkout > js > src > app > modules > CheckoutSkipLogin > CheckoutSkipLogin.js to use the new domain check method.

```
                 return promise;
                };

        ---     // don't wrap on non-secure domains (Site Builder cart is in Checkout :/ )
89:     ---     if (window.location.protocol !== 'http:')
        +++     // Site Builder cart is in Checkout :/ don't wrap if in shopping
        +++     if (Utils.isInCheckout())
                {
                    LiveOrderModel.prototype.save = _.wrap(LiveOrderModel.prototype.save, wrapp
er);
                }

...
```

## NavigationHelper.js

Customizing this file requires that you copy the **NavigationHelper** module folder from Reference ShopFlow to Custom Shopflow because it is not created automatically.

File locations:

- ShopFlow > Custom ShopFlow >js > src > app > modules > NavigationHelper > NavigationHelper.js
- MyAccount > Custom MyAccount >js > src > app > modules > NavigationHelper > NavigationHelper.js
- Checkout > Custom Checkout >js > src > app > modules > NavigationHelper > NavigationHelper.js

Add the new domain check methods in each of the files.

```
...
                // if we are heading to a secure domain (myAccount or checkout), keep setting t
he language by url
299:    ---     if (target_touchpoint.indexOf('https:') >= 0)
        +++     if (Utils.isInCheckout())
                {
                    var current_language = SC.ENVIRONMENT.currentLanguage;
                    if (current_language)
                    {
                        target_data.parameters = target_data.parameters ?
                                target_data.parameters + '&lang=' + current_language.locale :
                                'lang=' + current_language.locale;
                    }
                }

...
```

## Update Legacy Checks and Methods

Previous sections have covered customizing multiple files. This procedure provides steps for finding and replacing any other instances of the legacy domain checks, the legacy `getAbsoluteUrl()` methods, and the legacy `session.isLoggedIn()` methods.

### To update the legacy checks and methods:

1. Search the files for remaining instances of these legacy domain checks.
   - `!~request.getURL().indexOf('https')`
   - `isSecure`
   - `isSecureDomain`
     a. If instances are located in files that have not already been copied to the Custom folder, copy as needed.
     b. For each instance, determine the reason for the domain check.
        - Was it to protect a resource from being accessed through the shopping domain? In this case, call the `Utils.isCheckoutDomain()` method.
        - Does the method return a different result depending on whether we are in the checkout domain or the shopping domain? In this case, call the `Utils.isInCheckout()` method.

- Is there a visual change in the web store depending on whether we are in the checkout domain or the shopping domain? Call the `Utils.isInCheckout()` method or the `Utils.isInShopping()` method as appropriate for each domain.

c. Replace each instance with the appropriate new domain check method.

See the code snippet in commons.js for the method definitions.

- `Utils.isCheckoutDomain()`
- `Utils.isInCheckout()`
- `Utils.isInShopping()`
- `Utils.isSingleDomain()`

2. Search the files for remaining instances of:

```
session.getAbsoluteUrl(request.getURL().indexOf('https:') === 0 ?
'checkout' : 'shopping', '/{{file}}') :
```

a. If instances are located in files that have not already been copied to the Custom folder, copy as needed.

b. In Checkout or MyAccount, replace the instance with `session.getAbsoluteUrl( 'checkout', '/{{file}}')`.

c. In Shopping, replace the instance with `session.getAbsoluteUrl( 'shopping', '/{{file}}')`.

d. You can also replace any instance with:

```
session.getAbsoluteUrl(Utils.isInCheckout(request) ? 'checkout' :
'shopping', '/{{file}}').
```

This method call will check for either domain.

3. Search the files for remaining instances of the `session.isLoggedIn()` method.

a. If instances are located in files that have not already been copied to the Custom folder, copy as needed.

b. Replace each instance with the `session.isLoggedIn2()` method.

## Deploy the Migration

When all customizations are complete, you need to configure NetSuite to use the custom files instead of the reference files. Complete the following tasks:

- Configure the secure domain.
- Add or update library scripts for each customized SSP Application.
- Add or update touch points for each customized SSP Application.
- Deploy your site to the secure domain.

**Important:** These customizations are extensive. You should test them in a sandbox account or other test site before deploying to a production account.

### To deploy the migration:

1. If you have not already done so, complete the steps to configure a Secure Domain. For details, refer to Setting up Domains in NetSuite.
2. Go to Setup > SuiteCommerce Advanced > SSP Applications.
3. Click **Edit** to modify the custom SSP Application.

4. On the **Library** subtab under **Scripts**, add the library scripts as indicated:

   **Important:** The order of these scripts must match exactly.

   Web Site Hosting Files/Live Hosting Files/SSP Applications/<application>/...

   - ... Reference <SSP Application>/ssp_libraries/underscore.js
   - ... Reference <SSP Application>/ssp_libraries/backbone.validation.js
   - ... Custom <SSP Application>/ssp_libraries/commons.js
   - ... Reference <SSP Application>/ssp_libraries/backend.configuration.js
   - ... Custom <SSP Application>/ssp_libraries/Models.js

   For example, to update the commons.js file for ShopFlow 1.07.0, go to:

   Web Site Hosting Files/Live Hosting Files/SSP Applications/NetSuite Inc. — ShopFlow 1.07.0/Custom ShopFlow /ssp_libraries/commons.js

5. Click on the **Supported Touch Points** subtab.
6. Add the appropriate touch points using the custom .ssp files that were modified as part of this migration. See the help topic Selecting Supported Touch Points.
7. Click **Save**.
8. Deploy the Custom SSP Applications to your Secure Domain.
   a. Select the **Save & Link to Domain** button (from the **Save** drop down menu).
   b. Choose your secure domain.
   c. Click **Save**.
9. Repeat the Scripts, Touch Points, and Deploy steps for each SSP Application you customized (Custom Checkout, Custom MyAccount, and Custom ShopFlow).

# SuiteCommerce Services

SuiteCommerce Services enable you to apply the business logic in your NetSuite account to your e-commerce site. Service files work in the browser making backend calls to access business logic stored in your NetSuite account. This sections describes two services delivered in SuiteApps that you can install.

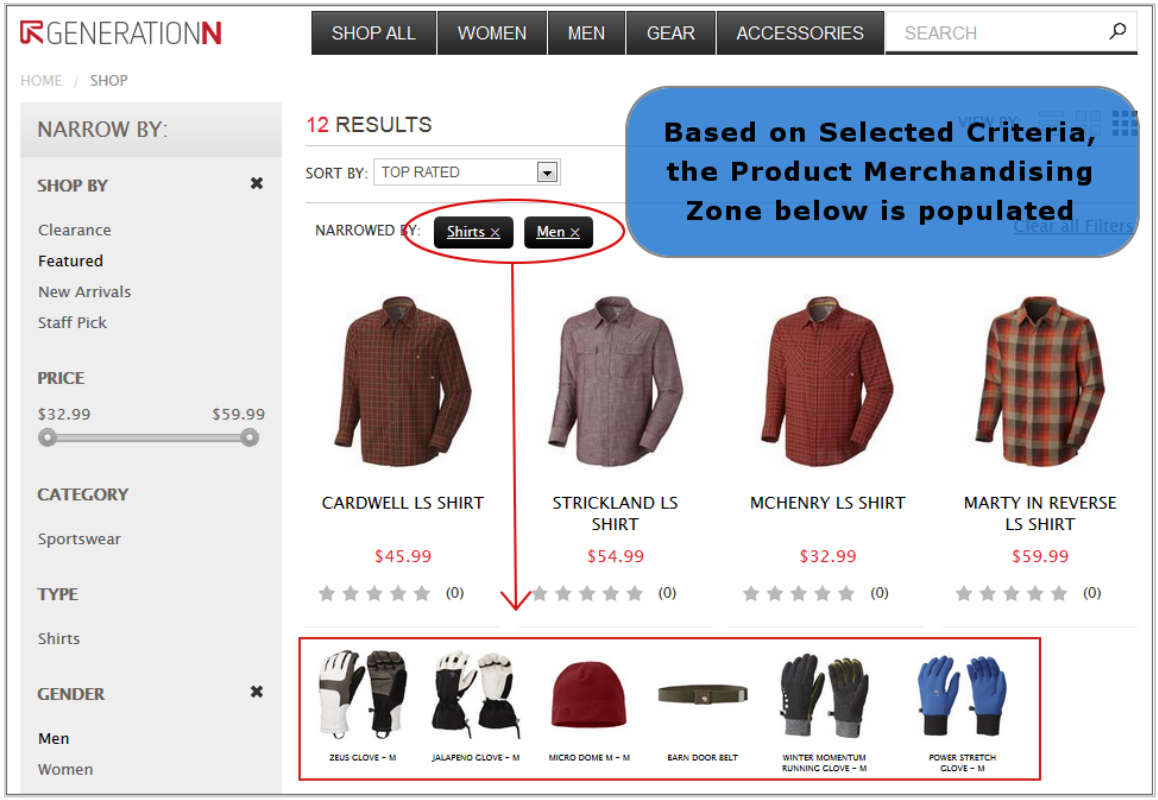
- Product Merchandising
- Content Delivery

> **Note:** These services are available for the ShopFlow, Checkout, and My Account Reference Implementations.

## Product Merchandising

The Product Merchandising SuiteApp allows you to define and display merchandising zones on your e-Commerce site. Merchandising zones display complementary, cross-sell or up-sell products. The content for merchandising zones is generated from interactive elements that display dynamic groups of items based on search rules you define.

For example, suppose that on a sporting goods store website a buyer has narrowed the search criteria on a page to display Mens featured items. You would like to market various accessories to this buyer as well. A merchandising rule is defined such that whenever the search criteria is narrowed to display mens shirts, then the rule would search for accessories and display the results in a merchandising zone on the page.



**Important:** The Product Merchandising SuiteApp is currently only available with the SuiteCommerce Advanced feature.

# Understanding Product Merchandising

When using the Product Merchandising SuiteApp, a product merchandiser or marketing manager must work with a web developer to set up site templates with merchandising zones. Once the intial setup is complete, these zones can be targeted with different content defined within merchandising rules without the need to rely on a developer for further customization.

**Note:** Good initial planning in the types of merchandising zones required will simplify the maintainence of sites and reduce the need for development costs.

The workflow for how a merchandising zone is implemented is as follows:

- First, the product merchandiser works with a developer to define the selection criteria that would result in certain products to display. Selection criteria is based on available facets and sort options defined during the SuiteCommerce Advanced Web Site Setup.
- The developer adds merchandising zones to website template files or creates new templates as needed to accomodate the desired display. These templates call the merchandising rule that initiates a query to the item search API, which then returns item data as requested by the rule.

  **Note:** If Content Delivery pages are used for the display of the Merchandising Rules, these pages must be defined after the Product Merchandising rules are defined. See Content Delivery for details on using the Content Delivery SuiteApp.

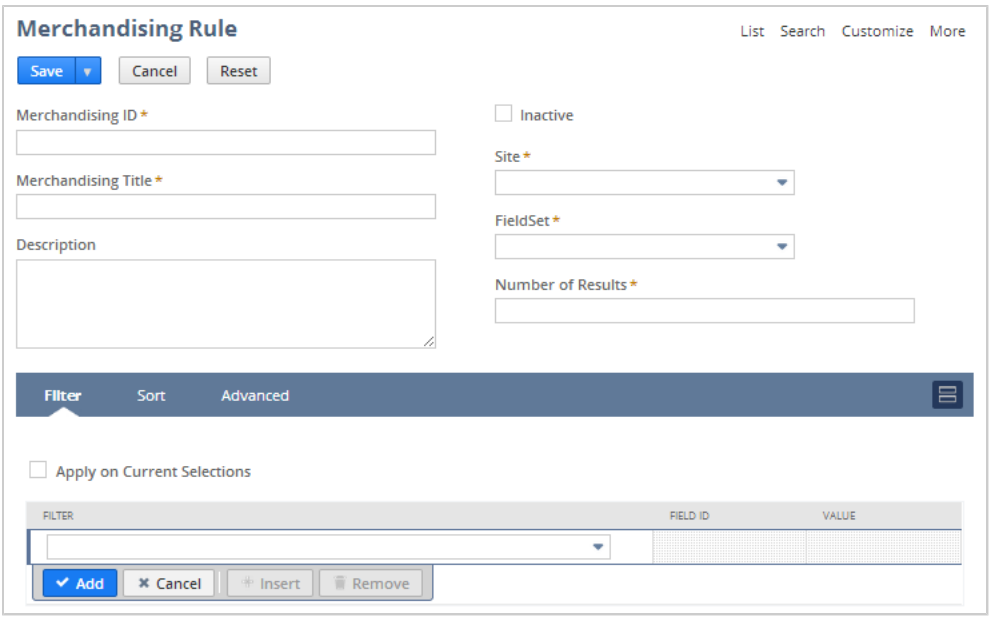
- The developer provides the product merchandiser with the Merchandising IDs to be used when setting up merchandising rules.
- The product merchandiser defines the Product Merchandising rules based on the information used from the developer.

# Creating a Product Merchandising Rule

Product Merchandising rules can be defined after you have coordinated with your developer to define the merchandising zones that will be available on your website. Refer to Understanding Product Merchandising.

## To create a merchandising rule:

1. Go to Setup > SuiteCommerce Advanced > Merchandising Rules > New.




2. Enter a **Merchandising ID** for this merchandising rule.

   This ID is used as a key by your website template to display the corresponding merchandising rule parameters. A list of available IDs should be provided by your website developer.

   > **Note:** If desired, you could first define the Product Merchandising rules and then provide your developer with the appropriate Merchandising IDs. The Merchandising IDs must match EXACTLY to what the developer is using when defining the content templates. When Content Delivery pages are used for defining merchandising zones, active Product Merchandising IDs are automatically available for selection during the Enhance or Landing page setup.

3. Enter a **Title** for your merchandising rule.
4. In the **Description** field, enter a short description of this rule.
5. Enable **Inactive** if this rule should NOT be used in the website at this time.

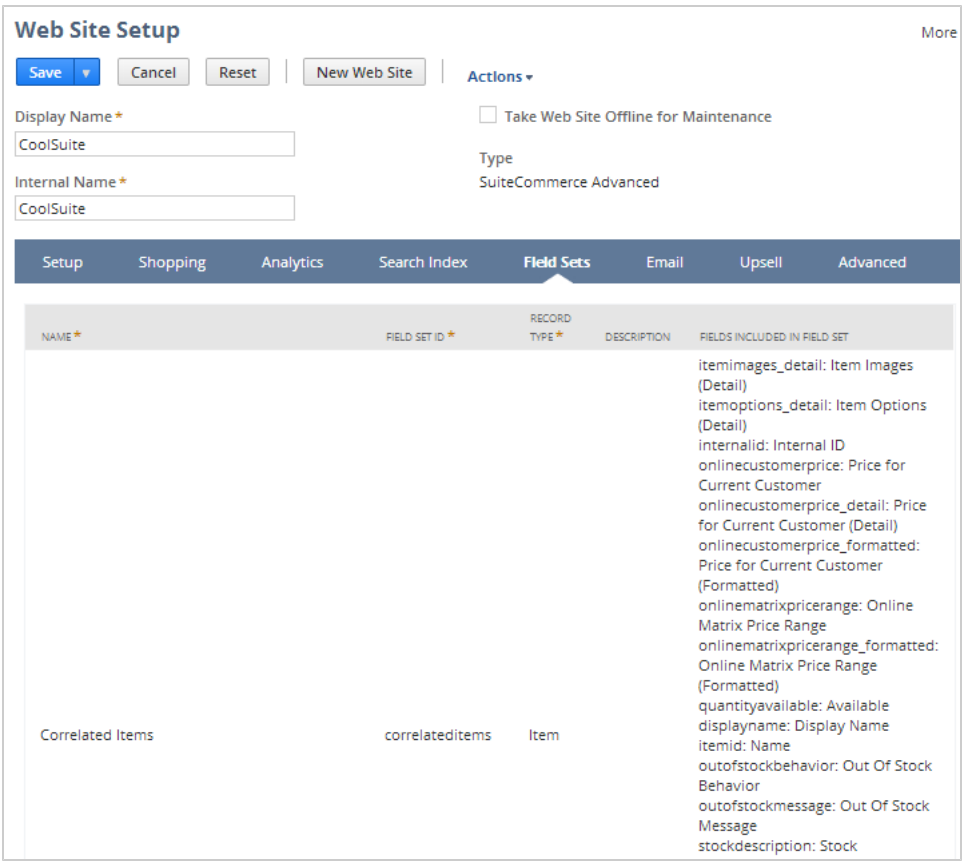
   > **Important:** Merchandising rules that are defined as Inactive are NOT available in Content Delivery records.

6. Select the website this rule is displayed in.

   Merchandising rules are only available in SuiteCommerce Advanced websites.

7. Select a field set to define the information your website uses when displaying item results.

   Field sets are defined during the initial site setup at Setup > SuiteCommerce Advanced > Set Up Web Site > Field Sets. Only those field set associated with the selected website are available.

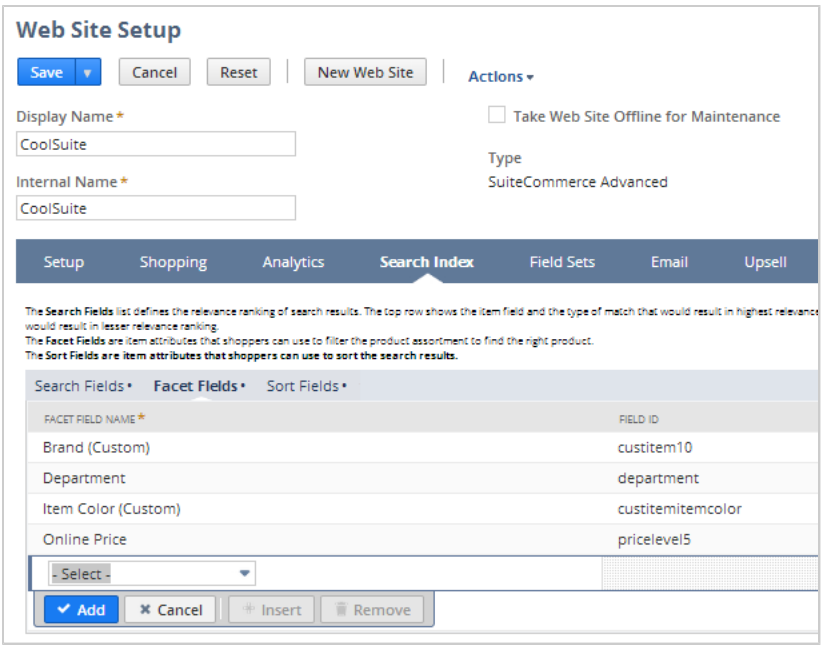



8. Enter the maximum **Number of Results** you want the rule to return.

   The number entered here should match the ideal number that can be accomodated by the page design for these items in your website.

9. Enter your **Filter** criteria:

   You can filter items based on Search Index Facet Fields defined for your website at Setup > SuiteCommerce Advanced > Set Up Web Site > Search Index. Facet fields are defined during the intial Web Site Set Up and are used in the website to determine item filter criteria available on the website. For detailed information on defining Facet Fields, refer to Facet Fields.



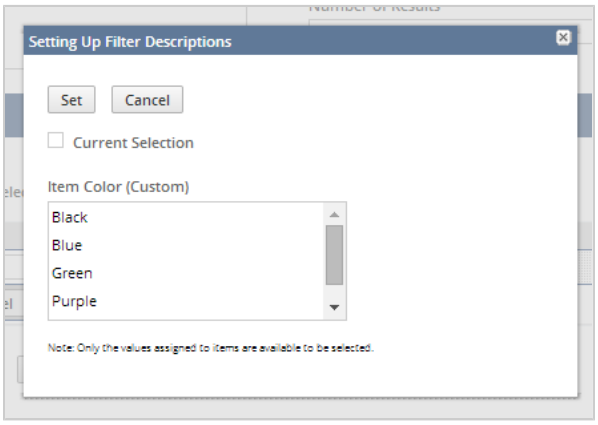

1. If desired, enable the **Apply on Current Selections** check box.

   When enabled, the current search criteria is included in the query to the item search API.

2. In the **Filter** column, select a field from the dropdown menu.

   The fields displayed are determined by the website chosen in Step 6. Once a filter is selected, a popup for setting up values opens. Depending on the type of field selected the options available for your filter vary. Fill in the filter criteria you need for your rule.

   > **Note:** Depending on the type of field you can again select whether the item will only be displayed if it matches the search criteria of the current session as described in Step 9- If desired, enable the Apply on Current Selections check box. by enabling **Current Selection**.




3. Click **Set** to close the popup, and then **Add** to save the line.

10. Enter your **Sort** criteria:

    You can sort items based on Sort defined for your website at Setup > SuiteCommerce Advanced > Set Up Web Site > Sort Fields. Sort fields are defined during the initial Web Site Set Up and are

used in the website to determine the sort order of search results. For detailed information on defining Sort Fields, refer to Sort Fields.

1. In the **Sort** column, select which field you want results sorted by.

   Fields available are determined by the website chosen in Step 6. Once a filter is selected, a popup for setting up values opens. Depending on the type of field selected the options available for your filter vary. Fill in the filter criteria you need for your rule.

2. Select **Ascending** or **Descending** in the **Sort Option** column.
3. Click **Add**.

   Sorting is applied in the order each field is entered.

11. Define **Advanced** options:

    In the **Advanced** tab you can specify Items to exclude and whether a custom template should be used for this rule. Check with your developer if a custom template needs to be specified. If left blank the default template is used.

12. Click **Save**.

Product Merchandising rules are now available for your website. If not already done, your developer will need to set up Product Merchandising zones that call these rules from your website template files. Once rules are defined and called from a website you can modify the rule as needed without need for further development work.

If you use the Content Delivery SuiteApp, you can also use merchandising rules to define the search results to display in a content area. For more information, see Content Delivery.

# Preparing Pages for Product Merchandising Zones

This section describes how to prepare Reference Implementation pages for Product Merchandising zones. Preparing pages for Product Merchandising zones should be done by your web application developers as it involves editing code within the related page.

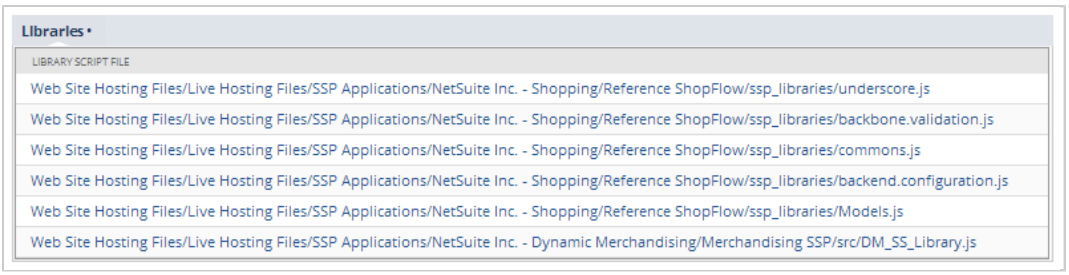
- Configuring the Reference Implementation
- Adding Merchandising Zones to Template Files
- Understanding Merchandising Zone Templates

## Configuring the Reference Implementation

In order to call Merchandising Rules from your Reference Implementation pages, you must first configure the Reference SSP to include the merchandising script.

### To include the Product Merchandising script:

1. Go to Setup > SuiteCommerce Advanced > SSP Applications.
2. Click on the Reference ShopFlow (or Checkout/My Account) application link.
3. Click **Edit**.
4. In the **Libraries** subtab under **Scripts**, click **Add**.
5. Navigate to the path of the DM_SS_Library.js file in the Product Merchandising reference installation.




> **Important:** If you are using a custom ssp application, you will need to define this library for your custom ssp application.

6. Click **Save**.

## Adding Merchandising Zones to Template Files

Once defined, Product Merchandising rules can be called from any template file to be displayed in a Merchandising Zone on the corresponding page view.

### To add the Product Merchandising zone to your site:

1. Find the template to contain the Product Merchandising Zone.
2. Copy the template into your Custom folder.
3. Add a reference to the new Product Merchandising zone in the following format:

```
<div data-type="merchandising-zone" data-id="merchandisingID"></div>
```

   where **merchandisingID** matches EXACTLY the merchandising ID set when defining the Product Merchandising rule. To view a list of existing Merchandising Rules, go to Setup > SuiteCommerce Advanced > Merchandising Rules.

> **Important:** The merchandising IDs set in the template file and in the Product Merchandising rule must match EXACTLY. Also, if the Product Merchandising rule is set to Inactive, any merchandising-zone data types defined in your templates related to the Product Merchandising rule will NOT be rendered.

4. Save and upload the modified template file to the NetSuite file cabinet.

On page load, the item search api will return product merchandising results based on parameters set within the merchandising rule being called.

For example, if a merchandising rule is defined that specifies that mens accessories should be displayed with a limit of 5 items in ascending order whenever the search criteria is narrowed to Mens Shirts, the item search api URL would include the following string:

```
&limit=5&fieldset=search&gender=men&type=shirts&sort=relevance%3Aasc.
```

## Understanding Merchandising Zone Templates

Any merchandising zone called from a Reference ShopFlow template has the default **merchandising** template applied to it automatically but you can create a new custom merchandising templates to display merchandising zones formatted exactly how you would like.

The current default merchandising template is located in the Reference ShopFlow directory at:

Reference ShopFlow > templates > merchandising > macros > merchandising_zone_macro.txt

The default template displays the Merchandising Title, the Merchandising description (if available) and then each of the items.

**Default Merchandising Macro (merchandising_zone_macro.txt):**

```
<% registerMacro('merchandisingZone', function (zone) { %>
<aside class="well well-inverse no-margin-bottom shopping-cart-recent-items">
   <h3><%= _(zone.title).translate() %></h3>
   <% if (zone.description) { %>
   <p><%= zone.description %></p>
   <div class="row-fluid">
      <%
         var relatedWrap = function relatedWrap (item) {
            return SC.macros.relatedItem(item, zone.application);
         };
      %>
      <%= SC.macros.displayInRows(zone.items, relatedWrap, 4) %>
   </div>
</aside>
<%})%>
```

To add your own custom merchandising template, make a duplicate of the existing default macro and edit the template as needed. Ensure that you supply a unique `registerMacro` name.

> **Note:** When using **custom** merchandising zone templates, the registerMacro name must be set in the template definition for the corresponding Product Merchandising rule. The template definition can be found on the Advanced tab of the Merchandising Rule record at Setup > SuiteCommerce Advanced > Merchandising Rules. Developers should provide the `registerMacro` name to the product merchandiser or whoever is responsible for creating and maintaining Product Merchandising Rules.

### Customising Item Display Macros within the Merchandising Zone

Merchandising zone templates are used to define the **merchandising zone** only. Each of the these templates calls a `relatedItem` template, `SC.macros.relatedItem`, to define the display of the individual items. You can further customize the display of your merchandising zones by providing a custom item display template. The default `relatedItem` template is defined in the macro at Reference ShopFlow > templates > globals > macros > related_item_macro.txt

```
<% registerMacro('relatedItem', function (item, application) { %>
<div class="item-cell item-cell-grid">
   <a class="thumbnail" href="<%= item.get('_url') %>">
      <% var thumbnail = item.get('_thumbnail'); %>
      <img src="<%= application.resizeImage(thumbnail.url, 'thumbnail') %>" alt="<%= thumbnail.
altimagetext %>">
   </a>
   <h5>
      <a href="<%= item.get('_url') %>">
         <%= item.get('_name') %>
      </a>
   </h5>
   <a class="btn btn-small btn-primary" href="<%= item.get('_url') %>">
      <%= _('See more').translate() %>
```

```
    </a>
</div>
<% }) %>
```

To add your own custom item display template, make a duplicate of the related_item_macro.txt macro and then edit the template as needed. Ensure that you supply a unique `registerMacro` name.

# Content Delivery

The Content Delivery SuiteApp combines customer forms, service files, and customer records to allow you to conveniently create and manage content pages and enhanced content areas for your website. Users who may not have a background in website scripting, such as marketing managers or site administrators, can integrate graphic elements, multimedia, and marketing content without writing code.

**Note:** The Content Delivery bundle is supported for the ShopFlow, Checkout, and My Account Reference Implementations. Content displayed in the Shopping Reference Implementation is delivered using this content delivery system. You can extend the reference implementation by creating new content records for use with your own e-commerce solution. However, you can NOT extend the Content Delivery SuiteApp itself.

## Understanding Content Delivery

There are two types of Content Delivery:

- **Landing Pages** : Allow you to create entire custom pages with custom URLs. You can specify as many different content areas within the new page and substitute content based on a set of pre-defined custom rules.
- **Enhanced Pages** : Allow you to supply custom content to existing pages. For example, for a page that returns item records based off of search criteria defined within the faceted navigation links you could add a descriptive content area to the top. This descriptive area could be further customized to display different content based off of attributes assigned to the enhanced page content area.

Content administrators create pages with containers to display content and define custom content tags to characterize page attributes such as language, delivery method, geography, or domain. Once pages have been designed, new content records can be created with rules allowing users to decide how and when their content is displayed on the website. Users can create rules with various attributes to support marketing efforts or scheduled website changes.

Content is generated as JSON data and can be presented in your NetSuite-powered website, mobile applications, or 3rd party services, such as an externally hosted website or content marketing service.

**Important:** Before defining Content Delivery pages, you must work with your developer to understand initial page design including content layout, content tag definitions, and css selectors used to call the content pages. Good initial planning will simplify the development and maintainence costs of your website and enable non-developers to define custom content when needed.

## Using Content Delivery

Follow these steps to create new content areas for your website. To complete your new content areas, you also need to create **content records** and **Content Tags**. These records are defined on-the-fly from the Content Delivery page forms.

Content Delivery pages can be defined after you have coordinated with your developer to define the templates, content tags, and css selectors that will be used on your website. Refer to Understanding Content Delivery.

## To add content delivery records:

1. Go to Setup > SuiteCommerce Advanced > Content Delivery.
2. In the lower right corner, click on one of the following:
   - **New Landing Page**
   - **New Enhanced Page**
3. In the **Name** field, enter a name for this content page record.

   This name does NOT display on your website. It is for record identification only. Each content record must have a unique name.
4. For **Landing Page Content** only, set the **URL** for this new page by entering a slash (/) followed by the desired URL. For example, `/about_us`.

   > **Note:** The URL set here is amended to the selected site's URL to form the URL for this specific page. This URL must be unique to the site. If you are duplicating an existing URL you must deactivate the URLs you no longer want to display. You can NOT set a URL here when defining Enhanced Page Content.
5. If desired, enable the **Inactive** checkbox. An Inactive page can NOT be called from the website or previewed.
6. Select the website you want this page to display in. Select multiple sites by holding down the Ctrl key.

   > **Note:** Content Delivery pages are only available in SuiteCommerce Advanced websites.
7. In the **Content Tag** field, select the tags that must be available to the page in order for this content to display.

   Select existing tags or **New** to create a new tag. You can select multiple tags by holding down the Ctrl button.

   Tags are NOT always required but may be used to further filter when the content area is displayed in your site. Content Tags reflect page attributes defined within the template files. When defining tags, enter the tag prefix (or group) followed by a colon and then the attribute.

   For example you could have a group of language attributes.

   locale:en_US
   locale:es_ES
   locale:fr_CA

   > **Note:** Locale attributes are supported out of the box in Reference Implementations. When content is specified for a certain locale and a Web Site is configured for multiple languages, content is displayed for the language set by the current shopper. See Handling Localization in Reference Implementations for more information.

   or a group of device attributes:

   device:mobile
   device:desktop
   device:iphone

> **Important:** Content Tags are used during customization and development of your website. Coordinate with your developer to obtain the list of available tags or to define a new set of tags. Content Tag names must match EXACTLY to the Content Tags as defined in the website assets by your developer.

8. For **Enhanced Pages** only, define the **Target URL**.

   Pages at the target URL use the enhanced content records defined here to replace tags with the specified content. These are existing URLs in your website. You can target content to multiple URLs. You can also use wildcards to specify a group of pages.

   For example, your site may have faceted navigation set up that allows users to filter on different types of shoes. When a specific URL based off of the selected filters is loaded, such as `/shoes/women/*`, then the enhanced content is displayed.

   > **Note:** This tab is not available when defining Landing Page Content. For Landing Page Content, a unique new URL must be provided as described in Step 4.

9. Define Content Rules:

   Content rules are used to define the content that will correspond to the current rule and where the content will display within the page. For Landing Pages, you may be defining content for each area within the page. For Enhanced pages you may have only a couple of areas being replaced with the content defined within these rules.

   1. For **Landing Pages** only, select a content record to display in the **Main Body** of your page.

      You can select an existing content record or create a new record by selecting New. See Creating a Content Record for more information.

      > **Note:** Selecting content in this field ensures that your page does not appear blank if no other rules are defined.

   2. In the **Target** column, enter a container name.

      Container names correspond to div areas defined in your Reference ShopFlow templates. The value entered here MUST match exactly to these div names. This information is provided by your website developer and corresponds to the area you want to display the selected content.

      > **Note:** Developers, you can refer to Preparing Pages for Content Delivery for more information.

   3. Select the content to be displayed.

      You can choose an existing content record or create a new record by selecting New. See Creating a Content Record.

> **Note:** In **Landing Pages**, the content record selected for the **Main Body** is NOT available for selection for other target areas.

10. Enter **Advanced** criteria.
    1. Define page elements.

       When this content is being created for a new Landing page, the page always loads with the information defined here. When created for Enhanced Page Content, the page is loaded only when the criteria for the enhanced page is met.

       - **Page Title**: this title appears in the browser title bar.
       - **Meta Keywords and Meta Description**: used to help search engines find your page.
       - **Page Header**: This content replaces the existing header for the page.
       - **Addition to <head> field**: enter any information you want to add to the <head> HTML element for this page.

         For example, you could enter JavaScript that collects page hit information for marketing purposes.
    2. Enter the **Name** of the template for this page.

       Normally this can be left blank. However, if you have a reference shopping site with a custom template, indicate the template used for that site. This information is obtained from your website developer.
11. Click **Save**.

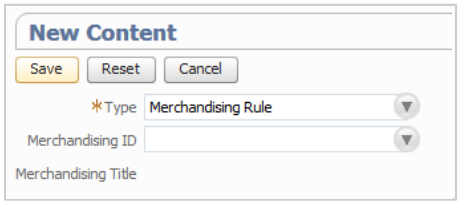
## Creating a Content Record

Content records collect information about media and files you want to display. You then define delivery rules with the Content Delivery SuiteApp. The New Content Record is accessed when defining main body content for Landing Pages, or when defining Content Rules for either Landing Pages or Content Records as described in Using Content Delivery.

When creating a new Content Record, you select one of two types, Merchandising Rule or HTML.

### Using a Merchandising Rule

Only previously defined Merchandising Rules are available for selection. Simply select the desired rule and then click Save. For details on defining Product Merchandising rules, refer to Product Merchandising.




### Using HTML

You can use your own custom HTML elements to define the content displayed. HTML content can be defined for your record in one of two ways:

- **File**: Select this to use an existing HTML file. This file must first be uploaded into the file cabinet. Although you can store these files anywhere in your file cabinet, it is recommended that you keep them with other website files in the folder at Web Site Hosting Files > Live Hosting Files.

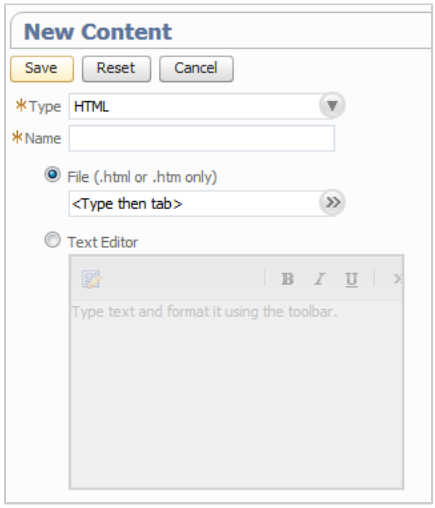
- **Text Editor**: Use the WYSIWYG editor to define the Content. Type your content in the Content field and using the built-in editor apply special formatting as desired.

> **Note:** When the Type field is set to image, enter the url for the image. The url entered is an absolute path from your Live Hosting Files/site folder. This is normally the top images folder. For example /images/navbar.jpg.

You can also embed client-side JavaScript to add custom logic to your Content Delivery pages.

> **Note:** If using jQuery methods in your logic you must use the `jQuery` prefix and not the `$` prefix. For example, use `jQuery(document).ready(function(){alert("Hello World");});` instead of `$(document).ready(function(){alert("Hello World");});`.

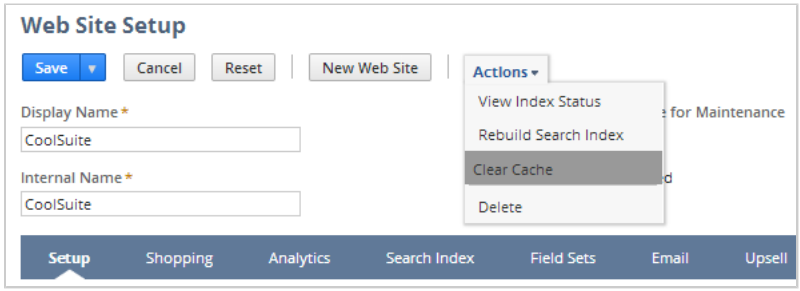
> **Important:** Although you can add client-side JavaScript to Content Delivery records, you cannot reference **external** JavaScript libraries in Content Delivery records. External libraries must be referenced from the SSP application files or by using more comprehensive customization techniques.




## Understanding Content Delivery Caching

When a page displays data from a content delivery record, that data is stored in the CDN cache for 2 hours by default. Changes to a content delivery record are not immediately reflected until this time has lapsed or the cache is cleared. For more information on CDN caching, see the help topic Configure CDN Caching in SuiteCommerce Advanced.

You can manually clear the cache on the Web Site Setup page. Go to Setup > SuiteCommerce Advanced > Set Up Web Site > Edit. Then select Clear Cache from the More Actions drop-down menu.



### Modifying the Default Caching Duration for Content Delivery Data

You can modify the caching duration for content delivery data by adding a **time to live (ttl)** parameter in the Content.DataModels.js module file at js > src > app > modules > Content. This parameter is validated by Content Delivery libraries to determine the requested caching duration.

> **Note:** ttl parameters are only valid for values set between 300 (5 minutes) to 7200 (2 hours). Any value less than 5 minutes results in 5 minutes caching and any value greater than 2 hours results in 2 hours of caching.

For example, in the following code snippet the ttl parameter is set to 300 seconds (5 minutes).

```
new Pages.Model({internalid: foundUrl.get('pageid'), id: foundUrl.get('pageid')}).fetch({
   data: {cache:'short',ttl:'300'}
,   killerId: this.Application.killerId
,   success: function (page)
   {
      Pages.Collection.getInstance().add(page);
      donefn(page);
   }
,   reset: true
});
```

# Preparing Pages for Content Delivery

This section describes how to prepare Reference Implementation pages for content delivery. Preparing pages for Content Delivery should be done by your web application developers as it involves editing code within the related page.

## Configuring the Reference Implementation

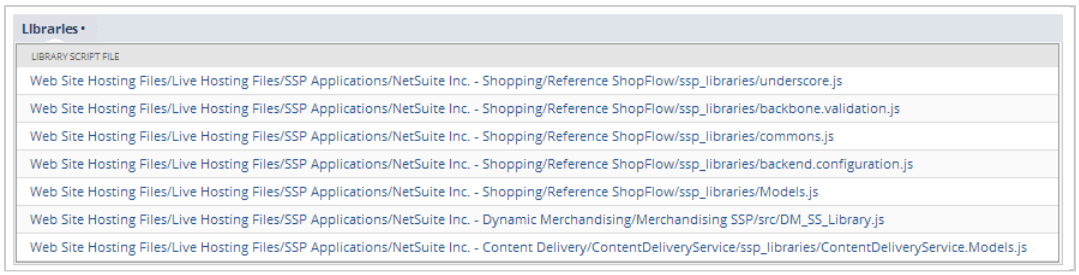
In order to display Content Delivery pages from your Reference Implementation pages, you must first configure the Reference SSP to include the content delivery scripts.

When multiple Reference Implementations are being used for your web store you can specify which implementations use a Content Delivery record by using the “app” tag with the name of the implementation as follows: Checkout, MyAccount, or Shopping. For example, using the tag app:Shopping causes the content to display in the Shopping Reference implementation only. If no app tag is used, the content is available for all Reference Implementations.

You can also include content generated from the Content Delivery service inside Backbone Modals. To use Content Delivery content inside a modal, the target ID defined in the Content Delivery record must be referenced by adding the prefixViewId “in-modal-”. You can also specify whether that content should display only when inside a modal, when inside or outside of a modal, or only when outside of a modal.

### To include the Content Delivery scripts:

1. Go to Setup > SuiteCommerce Advanced > SSP Applications.
2. Click on the Reference ShopFlow (or Reference Checkout/Reference My Account) application link.
3. Click **Edit**.
4. In the **Libraries** subtab under **Scripts**, click **Add**.
5. Navigate to the path of the ContentDeliveryService.Models.js file in the Content Delivery reference installation.



Libraries *

| LIBRARY SCRIPT FILE |
| --- |
| Web Site Hosting Files/Live Hosting Files/SSP Applications/NetSuite Inc. - Shopping/Reference ShopFlow/ssp_libraries/underscore.js |
| Web Site Hosting Files/Live Hosting Files/SSP Applications/NetSuite Inc. - Shopping/Reference ShopFlow/ssp_libraries/backbone.validation.js |
| Web Site Hosting Files/Live Hosting Files/SSP Applications/NetSuite Inc. - Shopping/Reference ShopFlow/ssp_libraries/commons.js |
| Web Site Hosting Files/Live Hosting Files/SSP Applications/NetSuite Inc. - Shopping/Reference ShopFlow/ssp_libraries/backend.configuration.js |
| Web Site Hosting Files/Live Hosting Files/SSP Applications/NetSuite Inc. - Shopping/Reference ShopFlow/ssp_libraries/Models.js |
| Web Site Hosting Files/Live Hosting Files/SSP Applications/NetSuite Inc. - Dynamic Merchandising/Merchandising SSP/src/DM_SS_Library.js |
| Web Site Hosting Files/Live Hosting Files/SSP Applications/NetSuite Inc. - Content Delivery/ContentDeliveryService/ssp_libraries/ContentDeliveryService.Models.js |

**Important:** If you are using a custom ssp application, you will need to define this library for your custom ssp application.

6. Click **Save**.

## Working with Existing Content Delivery Areas

In the Reference Implementation templates, there are many default content delivery areas. Each content delivery area is defined by the css class `content-banner`.

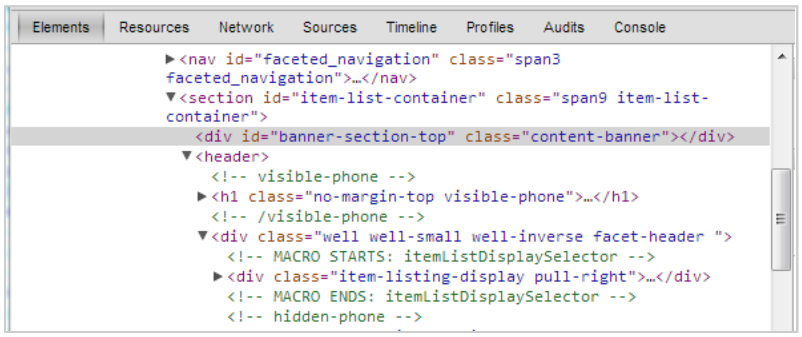
### To locate the desired content delivery area:

1. Open your web application in your browser, highlight the general area of interest, right-click and select Inspect Element.
2. In the debugger window, hover over nearby divs defined with the content-banner class. For example:

```
<div id="banner-section-top" class="content-banner"></div>
```

As you hover over each div, the corresponding area is indicated in the browser window. Each `content-banner` class div also has an id which indicates the associated area.

Once you've identified the correct div id, provide the CSS selector for this id in the Content Rule definition for the desired Content Delivery Enhanced Page definition. See Step 9 in Using Content Delivery. For example, to replace the top banner section, enter **#banner-section-top** in the Target column for the Content Rule.




## Adding Custom Content Delivery Areas

In addition to using the default content delivery areas, you can create your own custom content delivery areas.

### To create a custom content delivery area:

1. Find the template file you'd like the content delivery area to be applied to.
    1. Highlight the general area on the desired page, right-click and select Inspect Element.
    2. Scroll up until you see a reference to a template file similar to the following:

    ```
    <!--TEMPLATE STARTS: facet_browse_tmpl -->
    ```

    3. In the SuiteScript IDE, search in the templates directory for the file that includes the template string found in step b. Scroll up until you see a reference to a template file similar to the following:.

    > **Note:** The corresponding template file is normally named with the first part of the string such as facet_browse.txt rather than facet_browse_tmpl.txt.

    4. Verify that the file exists in the Custom folder of the Reference Implementation. If not, copy the file there from within the NetSuite file cabinet and then download to your SuiteScript IDE.
2. Modify the template file as desired with the custom content area div. For example:

    ```
    <div class="someCustomAreaName"></div>
    ```

    Leave the div area empty as this is dynamically populated only when content rules are met as specified in Enhanced Content Delivery Page definitions.
3. Upload the modified file to the NetSuite file cabinet.
4. Add the new custom Content Delivery target to Content Rules as needed.

    Depending on the definition for the div you may be using various CSS selectors when defining the target for the Content Rules. For example, dot notation is used for classes so if the div defined is `<div class="someCustomAreaName"></div>` then the Target defined in the Content Rule would be .someCustomAreaName instead of #someCustomAreaName (# is the CSS selector for div ids).

# Reference ShopFlow

**Important:** When upgrading to a new bundle version, refer to the Denali Release of SuiteCommerce Advanced for information on enhancements and fixes in that bundle.

The SuiteCommerce Reference ShopFlow Implementation delivers all the components you need to create an easy shopping experience for shoppers on your ecommerce website. Reference ShopFlow includes an SSP application, site template files, and other assets for you to get started using SuiteCommerce Advanced capabilities.

After installing Reference ShopFlow and deploying its touch points, you can deliver a best in class e-commerce experience to your customers online. Because Reference ShopFlow is installed in your NetSuite account, you have access to all the site templates, CSS, SuiteScript, and JavaScript libraries, for full control of the site behavior. You can extend Reference ShopFlow to customize the shopping experience for your site.

The site templates in the Reference ShopFlow include the following features:

- Responsive Design Layout (desktop, tablet, mobile) using SSP architecture and modern JavaScript Libraries
- Interactive Product Display — Includes image zooming, alternate views, and different colors. Size and color selection is available without dropdown menus.
- Product Listing Layouts — Provide customers with multiple display options of your list page. For example, shoppers can choose to view your items displayed either in a grid that is one, two, or three items wide.
- Quick View — Simplify the browsing experience by allowing customers to view detailed information about any item within a Quick View. The quick view lightbox is displayed above the main shopping experience and provides shoppers the most important information for making a purchase, including images, price, product description, and the ability to add the item to their cart.

Some additional capabilities installed with Reference ShopFlow include:

- Product Reviews — Custom forms are installed, so customers can submit product reviews on your site. Corresponding custom records are used to capture the product review information.
- Gift Cards — You can offer gift cards for sale, by creating an inventory item for sale, and then assigning the price.
- Newsletter Subscription — Site templates include fields where visitors on your site can enter information for subscribing to your company newsletter.
- A Set of Templates for SEO — The Reference SuiteApp installs site templates that are optimized for search engines.
- Shopping Cart — Reference ShopFlow includes its own cart and checkout templates. These templates differ from the Reference Checkout templates.
- Mini Cart — Support visibility into your customers' shopping carts by providing global access to a summary of their items. This feature allows shoppers to view a summary of their shopping cart from any page on your site.

Read the topics listed below to learn how install Reference ShopFlow, and how to identify and work with the components installed:

- Managing Reference Implementation Installations
- SuiteCommerce Reference Implementation Web Assets

- Technology and Framework
- Touch Points Available for Reference Implementations
- Reference ShopFlow Templates

# Quick Start

We recommend that you walk through the procedures outlined in the following section to familiarize yourself with Reference ShopFlow and get a site up and running quickly. This is meant as a quick start. For in depth setup, configuration, and customization details you'll want to peruse the complete set of Advanced SuiteCommerce documentation.

## Getting a Reference ShopFlow Site Running

### To set up your first Reference ShopFlow website:

1. Enable the required features.

   For details, refer to the section 'Creating a SuiteCommerce Advanced Web Site' in the SuiteCommerce Advanced documentation.

2. Install the bundle at Setup > SuiteCommerce Advanced > SSP Applications > New.

   You can install the Reference ShopFlow bundle but for future use you may want to install all of the following bundles, Reference ShopFlow, Reference My Account, Reference Checkout, Content Delivery, and Product Merchandising.

   For Reference ShopFlow you'll be installing the Reference ShopFlow bundle which contains the base ssp application and the Custom ShopFlow bundle where you will put any customizations to the Reference ShopFlow bundle.

   > **Note:** Your account representative provides the Bundle IDs available for your installation.

3. Ensure that the Custom ShopFlow bundle has a higher precedence than the Reference ShopFlow bundle.

   1. Go to Setup > SuiteCommerce Advanced > SSP Applications > Change Precedence.
   2. Edit the **Precedence** column values as needed and then click submit.

   With precedence set correctly, when the web application is executed any customizations done in the Custom ShopFlow bundle override the defaults in the Reference ShopFlow.

4. Go to Setup > SuiteCommerce Advanced > Set Up Web Site > New.
5. Click **Setup Site**.
6. Define the following fields in the New Web Site page:
   - **Display Name**: Enter a unique name for this site.
   - **Internal Name**: Automatically populates with the Display Name but you can enter another name.
7. Define a Domain for the site.

   Enter the **Domain Name**, set the HTML Hosting Root as the Live Hosting Files, and enable the domain as the Primary Web Site URL.

> **Note:** If you do not yet have a registered domain name you can define a routing within your windows hosts file as described in Creating a Domain for Viewing a Site Under Development.

8. Click **Save**.
9. Set a Touch Point for the application.
   1. Click **Edit** next to the site you created at Setup > SuiteCommerce Advanced > Set Up Web Site.
   2. In the **Touch Points** subtab, enter:
      - **Name**: View Homepage
      - **SSP Application**: Reference ShopFlow
   3. Click **Deploy to Site**.
10. Set up some **Field Sets**.

    These field sets define the data sets the Item Search API returns from a query.

    > **Important:** The following fieldset/field definitions are suggested as a starting point only to get a site up and running. As a minimum, you MUST include the field sets described below. To quickly add all required field sets you can also use the Setup Script as described in Using a Setup Script to Define Field Sets

    1. Return the Web Site Setup page at Setup > SuiteCommerce Advanced > Set Up Web Site and click **Edit** on the new site line item.
    2. Add the following field set:
       - **Name**: search
       - **Field Set ID**: search
       - **Record Type**: Item
       - **Fields Included in Field Set**: itemimages_detail: Item Images (Detail), itemoptions_detail: Item Options (Detail), onlinecustomerprice_formatted: Price for Current Customer (Formatted), description: Description, displayname: Display Name, internalid: Internal ID, urlcomponent: URL Component, itemid: Name, isinstock: In Stock, ispurchasable: Available for Purchase, isbackorderable: Available for Backorder, showoutofstockmessage: Display Out-of-Stock Message.
    3. Add the following field set:
       - **Name**: details
       - **Field Set ID**: details
       - **Record Type**: Item
       - **Fields Included in Field Set**: itemimages_detail: Item Images (Detail), itemoptions_detail: Item Options (Detail), onlinecustomerprice_formatted: Price for Current Customer (Formatted), description: Description, displayname: Display Name, internalid: Internal ID, urlcomponent: URL Component, isinactive: Inactive, isonline: Available Online, itemtype: Type

       > **Note:** Access the **New Field Set** dialog box by clicking the little popup icon in the lower right corner of the Fields Included in Field Set entry.
11. Add a **Facet Field**.

    Facet fields define the navigation facets displayed on your website.

1. Click the **Search Index** tab and then the **Facet Fields** subtab.
2. Add **Online Price**.

> **Note:** Adding Facet Field definitions in the Web Site Setup page initiates a rebuild of the search index. This can take a couple of minutes. You are automatically redirected to the status page but to return to the page if you navigate away select View under the Search Index Status column for your site at Setup > SuiteCommerce Advanced > Set Up Web Site.

12. Click **Save**.
13. Test the site.

    Your new site should now be available at the domain entered in Step 7.

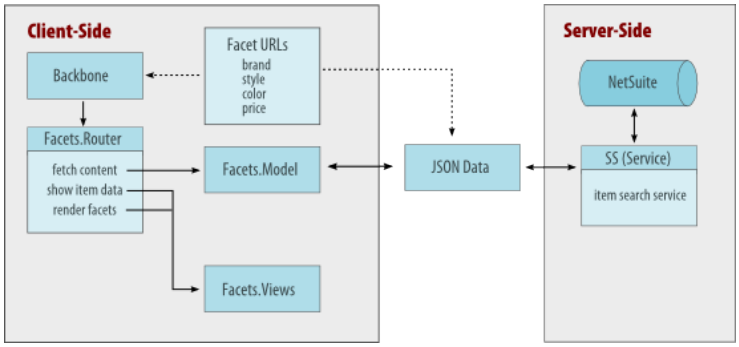
# Working with Faceted Navigation

Reference ShopFlow uses modern faceted navigation for displaying product results. With faceted navigation users can refine search queries incrementally based on specific item attributes defined within the facet configuration.

> **Note:** These item attributes can reflect values from standard fields or custom fields.

When a facet field is clicked, a call is made to the backend and a new set of search results filtered according to the facet parameters is generated for display within the Product Display area. Routers execute based off of selected URLs passed by the faceted navigation. Note that all selected facets are concatenated into a single URL so that you can pass multiple attributes for your item search criteria. Faceted navigation URLs are made to be short and readable to maximize SEO.

The router then calls the Facets.Model which returns data via the Search API in a JSON object. The Facets.View then takes the JSON data, generates the result set for presentation, and then regenerates the facet navigation area to update any dynamic changes there.




## Configuring Facet Fields

Facet fields are defined in the Web Site Setup Record at Setup > SuiteCommerce Advanced > Set Up Web Site > Edit > Search Index > Facet Fields (see the help topic Facet Fields). By default, only the Online Price field (with a field id of pricelevel5) is defined.

- Customizing New Facet Fields
- Facet Configuration Options

- Customizing Facet Configuration Options
- Available Facet Macros
- Specifying Which Facets to Display
- Rearranging Facets

## Customizing New Facet Fields

When a facet field is added, and item records exist that have field values set for the given facet field, the navigational area on the website automatically displays the facet allowing users to narrow their search results by that field. You can customize the display of the facet by editing configuration options in the Configuration.js file.

### To add facet definitions for newly defined facet fields:

1. In the NetSuite file cabinet, copy the Configuration.js file in the Reference folder to the Custom folder in the same directory.

   Copy:

   Documents > Files > File Cabinet > Web Site Hosting Files > Live Hosting Files > SSP Applications > [Implementation Name] > Reference ShopFlow > js > src > app.

   To:

   Documents > Files > File Cabinet > Web Site Hosting Files > Live Hosting Files > SSP Applications > [Implementation Name] > Custom ShopFlow > js > src > app.
2. In the SuiteScript IDE, download the Configuration.js file.
3. Edit the Configuration.js file to include the facets to be modified.
   1. Find the field id of the facet field you want to define.

      This id is listed within the facet field definition at Setup > SuiteCommerce Advanced > Set Up Web Site. Click Edit next to your site, then navigate to the Search Index tab and Facet Fields subtab.
   2. Define the facet configuration, setting the id as the id of the facet field you wish to modify.

For example, the Configuration.js has a single default facet, `pricelevel5`, defined.

```
,   facets: [
      {
          id: 'pricelevel5'
      ,   name: 'Price'
      ,   url: 'price'
      ,   priority: 0
      ,   max: 5
      ,   behavior: 'range'
      ,   macro: 'facetRange'
      ,   step: 50
      ,   uncollapsible: true
      ,   parser: function (value)
         {
            return _.formatCurrency(value);
         }
      }
   ]
```

In the following code, a second facet, Item Color, has been added:

```
,   facets: [
      {
          id: 'pricelevel5'
      ,   name: 'Price'
      ,   url: 'price'
      ,   priority: 0
      ,   max: 5
      ,   behavior: 'range'
      ,   macro: 'facetRange'
      ,   step: 50
      ,   uncollapsible: true
      ,   parser: function (value)
         {
            return _.formatCurrency(value);
         }
      },
      {
          id: 'custitemitemcolor'
      ,   name: 'Item Color'
      ,   url: 'color'
      ,   priority: 1
      ,   max: 5
      ,   behavior: 'multi'
      ,   macro: 'facetColor'
      ,   colors: color_map
      }
   ]
```

The `color_map` option is an array construct.

## Facet Configuration Options

In the above code snippet, note the following default configuration options:

- **id**: This is the internalId of the facet field and must be an exact match.
- **name**: sets the display name for the facet
- **url**: sets the URL string. To ensure the best SEO results, keep this short and descriptive.
- **max**: sets the number of facet choices that are displayed before a 'see more' expansion link is displayed
- **behavior**: Single allows for a single selection. Multi allows for multiple selections of facet choices. Range allows for a range of selections.
- **priority**: sets the display order of priority for the list of facets included in the configuration file. Priority 1 would display at the top, with priority 2 next etc.
- **macro**: defines a specific macro attribute for the facet.

  By default, the `facetList` macro is used to define the facet options but you can create your own custom macros for facet navigation and then apply it to a specific facet by setting this macro attribute.

  You can also define a different default facet macro attribute in the defaults section of the configuration.js file.

```
// default macros
,   macros: {
      facet: 'facetList'
      // each apply to specific item option types
   ,   itemOptions: {
         select: 'itemDetailsOptionTile'
      ,   standard: 'itemDetailsOptionText'
      }
      // for rendering options in the shopping cart
   ,   cartOption: 'shoppingCartOptionDefault'
   }
```

- **uncollapsible**: if set to true, the facet can be collapse in the user interface by clicking a down arrow icon. If set to false, the facet can NOT be collapsed.
- **colors**: an array construct defined within the Configuration.js file which defines colors defined for a field and html color values. Add your own custom constructs here to define a unique set of color attributes to be used. This array should match the set of colors available for the facet field in the UI.

  For example in the Configuration.js file notice the `color_map` array. You could add a second array like the following and then refer to it in the colors option.

```
_.extend(color_map, {
    'Red': 'red'
,   'Blue': 'blue'
,   'Gray': 'gray'
,   'White': 'white'
,   'Green': 'green'
,   'Black': 'black'
,   'Violet': 'violet'
,   'Silver': 'silver'
});
_.extend(mycustomcolors_map, {
    'Red': '#FF0000'
,   'Blue': '#0033CC'
,   'Gray': '#808080'
,   'White': '#FFFFFF'
,   'Green': '#008000'
,   'Black': '#000000'
,   'Orange': '#FF6600'
,   'Violet': '#660099'
,   'Yellow': '#FFFF00'
,   'Silver': '#C0C0C0'
});
```

## Customizing Facet Configuration Options

You can customize the available options by updating the facetList macro with custom code. Facet options are available as attributes of the `config` object in this macro. To modify the facet option, search for the `config.[attribute]` string in the facetList macro and edit as desired.

For example, the following code snippet demonstrates how you could modify config.name with some basic html markup to change the display color of the facet name option.

Original code:

```
<%= _(config.name).translate() %>
```

Modified code:

```
<span style="color:red">
   <%= _(config.name).translate() %>
</span>
```

You can customize which facet fields to include in your navigation area. See Customizing the Facet Navigation Panel.

## Available Facet Macros

Several Facet macros are available as outlined in the following table.

| Macro Name | File Name | Description |
|---|---|---|
| facetCategories | facet_categories_macro.txt | |
| facetColor | facet_color_macro.txt | |
| facetList | facet_list_macro.txt | This is the default macro. |
| facetRange | facet_range_macro.txt | |

Macro files are generally named with the format of *facets_x_macro.txt*, where x is the name of the facet macro, and can be found at:

Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Shopping > Reference ShopFlow > templates > items_list > macros

To verify if a selected macro file supports facet configuration, examine the top lines of the file. There should be something similar to:

```
<% registerMacro('facetColor', function(translator, config, facet) { %>
```

Values for the config parameter are set in the facet definition within the Configuration.js file. For example, the pricelevel5 facet has several configuration values such as name, url, priority, etc:

```
,   facets: [
      {
          id: 'pricelevel5'
      ,   name: 'Price'
      ,   url: 'price'
      ,   priority: 0
      ,   max: 5
      ,   behavior: 'range'
      ,   macro: 'facetRange'
      ,   step: 50
      ,   uncollapsible: true
      ,   parser: function (value)
         {
            return _.formatCurrency(value);
         }
      }
```

```
    ]
```

The default macro used for facets where the macro parameter is not explicitly set in the Configuration.js file is `facetList`. You can change this default setting.

### To change the default macro used for facets:

1. If not already done, in the NetSuite file cabinet copy the Configuration.js file from the Reference folder to the Custom folder.
2. Download the Configuration.js file to your SuiteScript IDE.
3. Change the facet default defined in the macros variable. For example, change `facetList` to `facetRange`.

```
    // default macros
  ,  macros: {
       facet: 'facetList'

    ,   itemOptions: {
          // each apply to specific item option types
          selectorByType:
          {
             select: 'itemDetailsOptionTile'
          ,   'default': 'itemDetailsOptionText'
          }
          // for rendering selected options in the shopping cart
       ,   selectedByType: {
             'default': 'shoppingCartOptionDefault'
          }
       }

    }
```

# Customizing the Facet Navigation Panel

You can fully customize the Facet Navigation Panel to modify which facets are displayed and how. The view file for the facet panel is used to coordinate multple views for the backend facet data. This file is located at js > src > app > modules > Facets > Facets.Views.js.

- Specifying Which Facets to Display
- Rearranging Facets

## Specifying Which Facets to Display

In the Facets.Views.js file, there are two jQuery selectors defined which can be used to control the facets displayed. These selectors are defined in the `renderFacets` function as follows:

- **all-facets** : this is the default selector used and displays all facets defined in the Configuration.js file.

```
this.$('div[data-type="all-facets"]').each(function (i, nav)
         {
            var $nav = jQuery(nav).empty()
```

```
            ,   exclude = _.map( ( $nav.data('exclude-facets') || '').split(','), function (res
ult) {
                  return jQuery.trim( result );
               })
            ,   ordered_facets = facets && facets.sort(function (a, b) {
                  // Default Prioriry is 0
                  return (translator.getFacetConfig(b.id).priority || 0) -
                   (translator.getFacetConfig(a.id).priority || 0);
               })
            ,   content = '';

            _.each(ordered_facets, function (facet)
            {
               var facet_config = translator.getFacetConfig(facet.id);
               if ( !_.contains(exclude, facet.id) )
               {
                  content += SC.macros[facet_config.macro ||
                   self.application.getConfig('macros.facet')](translator, facet_config, facet)
;
               }
            });

            $nav.append( content );
         });
```

- **facets**: generates a single facet based upon a given `facet-id` attribute or `facet-macro` attribute as defined in the Configuration.js file.

```
      this.$('div[data-type="facet"]').each(function (i, nav)
      {
         var $nav = jQuery(nav).empty()
         ,   facet_id = $nav.data('facet-id')
         ,   facet_config = translator.getFacetConfig( facet_id )
         ,   facet_macro = $nav.data('facet-macro') || facet_config.macro ||
             self.application.getConfig('macros.facet')
         ,   facet = _.find(facets, function (facet) {
               return facet.id === facet_id;
            });

         $nav.append( SC.macros[ facet_macro ](translator, facet_config, facet) );
      });
```

## To customize the Facet Navigation to display only specific Facets:

1. Copy the following files to your Custom reference implementation folder in the NetSuite file cabinet and download them to your SuiteScript IDE:
   - js > src > app > modules > Facets.Views.js
   - templates > items_list > facet_browse.txt
   - js > src > app > Configuration.js
2. In the facet_browse.txt template, modify the div that defines the facet navigation to use the facet jQuery selector.

   In your browser debugger if you inspect the facet navigation element, you'll notice the div with the `id="default-facets-container"` is associated with the facet navigation panel.

This is the div we need to edit in the facet_browse.txt template, redefining the data-type selector and specifying which facet to include.

Replace:

```
<div id="default-facets-container" class="facets-container" data-type="all-facets"
data-exclude-facets="custitem_exclude_from_search"></div>
```

With:

```
<div id="default-facets-container" class="facets-container">
   <div data-type="facet" data-facet-id="facetId"></div>
</div>
```

where `facetId` is the id attribute for the desired facet as configured in the Configuration.js file

> **Note:** In the Facets.Views.js file, the jQuery selector takes the *facet-id* attribute whereas in the facet_browse.txt file, you need to code this attribute as *data-facet-id*.

3. Add additional divs for each facet you want displayed.

```
<div id="default-facets-container" class="facets-container">
   <div data-type="facet" data-facet-id="facetId"></div>
   <div data-type="facet" data-facet-id="facetId2"></div>
   <div data-type="facet" data-facet-id="facetId3"></div>
</div>
```

4. Upload the modifed files back to your NetSuite file cabinet.

Now, your facet navigation panel will contain only those facets explicitly set in the facet_browse.txt template. Of course you can also modify the the display as desired by adding css classes to the divs.

## Rearranging Facets

You can rearrange the display of facets on the page. Use your browser debugger to find the div that corresponds to the location you'd like the facet navigation to display and then search for that div within the facet_browse.txt template. Move the facet div to this new location or create a new facet div.

# Customizing and Extending Shop Flow Components


- Customizing Item URLs for SEO
- Customizing the View By Area


## Customizing Item URLs for SEO

By default, item URLs are displayed using the urlcomponent defined on item records, or if that is not defined for an item record, the product and internalid for the item is used. For example you may see something similar to domain.com/product/361 as the URL for an Item. In order to maximize SEO it is better to use url components.

Item URLs are controlled by the following files:

- Template macros: depending on your configuration these are the item_cell_grid_macro.txt, item_cell_list_macro.txt or the item_cell_table_macro.txt files at ShopFlow/templates/items_list/macros/.

```
<h5><a href="<%= item.get('_url') %>"><%= item.get('_name') %></a></h5>
```

- ItemsKeyMappings.js: located at /js/src/app/ItemsKeyMapping.js.

```
    ,   _url: function (item)
      {
        // if its a standar version we need to send it to the canonical url
        if (SC.ENVIROMENT.siteType && SC.ENVIROMENT.siteType === 'STANDARD')
        {
          return item.get('canonicalurl');
        }
        // If this item is a child of a matrix return the url of the parent
        else if (item.get('_matrixParent') && item.get('_matrixParent').get('internalid'))
        {
          return item.get('_matrixParent').get('_url');
        }
        // Other ways it will use the url component or a default /product/ID
        return item.get('urlcomponent') ? '/'+ item.get('urlcomponent') : '/product/'+
        item.get('internalid');
      }
```

From this code we can see that if a URL component is set for an Item record this will be used in the URL instead of the /product/id/ attributes.

### To use urlcomponents item urls:

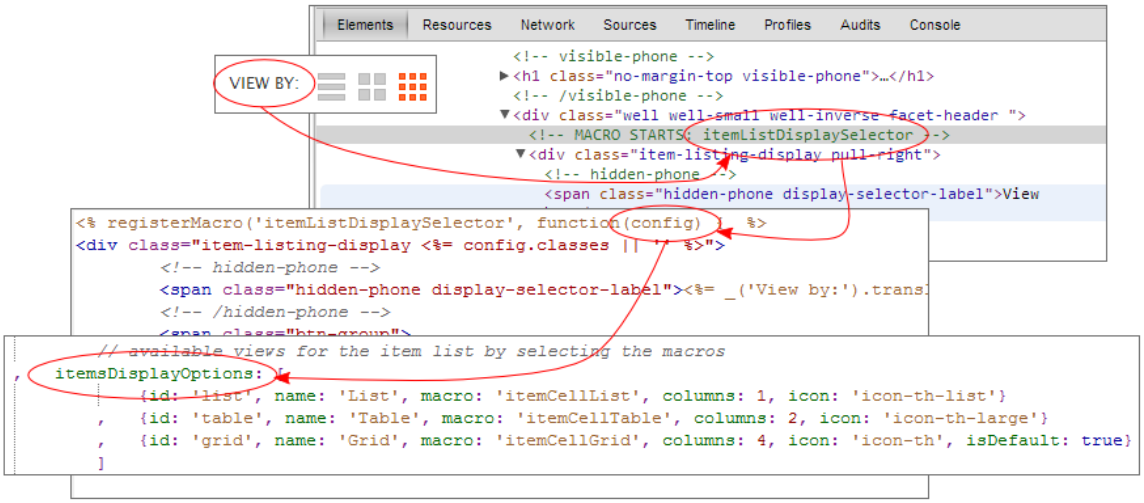
1. Add the URL Component Field to the Search Field Set.
    1. Go to Setup > SuiteCommerce Advanced > Set Up Web Site and click Edit next to the desired site.
    2. In the **Field Sets** tab, select Fields Included in Field Set details for the search Field Set and click the **Set** icon in the lower right corner.
    3. In the **Edit Field Set** dialog box, Select the **URL Component** field and click **Add**.
    4. Click **Submit** and then **Done** on the search line item.

    The urlcomponent field is now added to the field set.
2. Set the urlcomponent field on the items records.
    1. Go to Lists > Web Site > Items.
    2. Click **Edit** next to the desired Item.
    3. Enter the desired **URL Component** in the **Web Store** tab under the **Search Engine** and **Feeds** section.

## Customizing the View By Area

By default, Reference ShopFlow has three View By categories which allow the user to display item search results in either a List, Table, or Grid view. These view options can be customized in several ways.

The following graphic shows the relationships between the files involved in the display of the View By categories. If you inspect the View By element, you can see that the macro for `itemListDisplaySelector` is being called. This macro is the item_list_display_selector_macro.txt file. As shown, the `itemListDisplaySelector` macro function takes configuration parameters from the Configuration.js file.




## To customize the View By categories:

1. Verify that the Configuration.js file exists in the Custom folder of the ShopFlow Implementation. If not, copy the file there from within the NetSuite file cabinet and then download to your SuiteScript IDE. This file is located at:

   Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Shopping > Reference ShopFlow > js > src > app

2. Edit the file to rearrange, add, remove or change the parameters as desired.

   This configuration file allows you to make basic changes to the existing setup. For example, you could add a fourth option with two columns of detailed information by adding an additional line:

```
   // available views for the item list by selecting the macros
   ,   itemsDisplayOptions: [
          {id: 'list', name: 'List', macro: 'itemCellList', columns: 1, icon: 'icon-th-list'}
      ,   {id: 'columns', name: 'Columns', macro: 'itemCellList', columns: 2, icon: 'icon-th-co
lumns'}
      ,   {id: 'table', name: 'Table', macro: 'itemCellTable', columns: 2, icon: 'icon-th-large
'}
      ,   {id: 'grid', name: 'Grid', macro: 'itemCellGrid', columns: 4, icon: 'icon-th', isDefa
ult: true}
      ]
```

   **Note:** Item display options use the current width of the screen device, not the width of the browser, when the user loads the page. To reset the display, the user must resize the window and refresh the browser, regardless of device.

3. Edit the related macro or create new macros as needed (again, first verify the macro file exists in the Custom folder).

   There are three default macros available (as referenced from the Configuration.js file). These macros are located at:

Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Shopping > Reference ShopFlow > templates > items_list > macros

- item_cell_grid_macro.txt
- item_cell_list_macro.txt
- item_cell_table_macro.txt

4. Upload the edited file to the NetSuite file cabinet.

# Product Reviews

Reference ShopFlow has Product Review capabilities available as standard functionality. Shoppers can submit reviews for items and review ratings and details are displayed for each item on a website. Also, content from product reviews use microdata markup so that search engines display review information for items in search results.

> **Note:** There is a Product Review bundle available for **Site Builder** sites as well but that is a separate bundle and can NOT be used for SuiteCommerce Advanced sites.

## Getting Started with Product Reviews

Before you begin with Product Reviews you must install the Reference Product Review Records bundle. This bundle contains all of the required custom records, lists, and fields needed to store and manage Product Reviews. These custom records can be modified as needed. For detailed information on installing the reference bundle, see Managing Reference Implementation Installations.

The following two Product Review custom records are installed at Customization > Lists, Records, & Fields > Record Types.

- **Product Reviews**: This record is used to define the Product Reviews subtab under the SuiteCommerce Extensions tab on the Item record. When a shopper submits a review for an item on your website, the review is appended to this record.

  > **Important:** If modifying the Product Reviews record, do NOT edit or remove the **Item ID** and **Item** fields on the record. These fields are used in critical backend logic and should NOT be edited.

- **Product Reviews Attr. Rating**: This record is used to define the Product Review Attr. Ratings subtab under the SuiteCommerce Extensions tab on the Item record. Here, product review attributes available on the web store can be defined. By default, three attributes can be set for any given item: Price, Quality, or Durability. Additional attributes can be created.

### Prerequisites

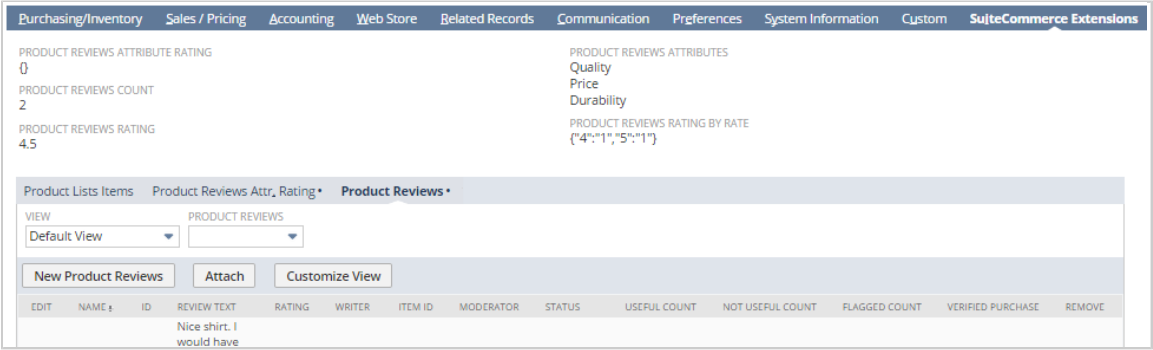
Before you begin working with Product Reviews ensure that the following fields are included in your Field Set definitions. For more information on Field Sets see the help topic Field Sets.

- NS PR Attributes (Custom)
- NS PR Attributes Rating (Custom)
- NS PR Count (Custom)

- NS PR Rating (Custom)
- NS PR Rating By Rate (Custom)

## Using the Product Reviews Record

The Product Reviews record displays all reviews for a given item in the Product Reviews subtab under the SuiteCommerce Extensions tab on an item record and allows a moderator to evaluate reviews and control the display of reviews on your website.




> **Note:** Indexing for websites is performed when item elements are changed such as product review Attributes or when the Status is changed. Therefore any changes to the item records may take a few minutes to be reflected on the website. You can check the indexing status by clicking View next to the desired website at Setup > SuiteCommerced Advanced > Set Up Web Site.

In the top panel of the Product Reviews record, Product Review analysis and Attribute configuration can be done. The following fields are available:

- **Product Reviews Attribute Rating**: Lists the average ratings for each attribute in a JSON object.

  The JSON object consists of a name value pair with the Attribute ID and the calculated average rating. You can check Attribute IDs on the NS PR Attributes custom record at Customization > Lists, Records, & Fields > Lists.

  For example, {"3":"3.0","2":"2.7","1":"3.0"} indicates average ratings of each of the three default Attributes where Quality is ID 1, Price is ID 2, and Durability is ID 3.
- **Product Reviews Count**: Displays the total number of reviews for this item.
- **Product Reviews Rating**: Displays the Overall rating for this item.
- **Product Reviews Attributes**: Displays the Attributes set for this item. For more information, see Defining Product Review Attributes.
- **Product Reviews Rating by Rate**: Lists the total count for each overall rating score in a JSON object.

  The JSON object consists of a name value pair with the Overall rating value and the number of reviews set to that rating.

  For example, {"1":"1","4":"2"} indicates a single instance of an Overall rating of 1 and 2 instances of an Overall rating of 4.

> **Note:** Product Review statistics are only calculated for **Approved** reviews.

Note the following details for line items on the Product Reviews record:

- When a product review is submitted by a shopper who also bought the product, the **Verified Purchase** field is set to Yes.
- By default only items set to **Approved** are displayed in the website and included in the product review rating analysis.
- The Writer field corresponds to the Customer ID for the shopper who submitted the review. It is only populated for logged in users.

# Defining Product Review Attributes

Product Review Attributes settings are used to define individual review components for an item. If no Attributes are defined for an item, only an overall rating can be given or displayed for an item.

> **Note:** Product Review Attributes can NOT be set for Assembly/Bill of Materials items.

## To define Product Review Attributes for an item:

1. Find the item you want to define the attributes for at Lists > Web Site > Items.
2. Click the **SuiteCommerce Extensions** tab.
3. If necessary create a custom **Product Reviews Attribute**.

   By default Price, Quality, and Durability are available but you can add any custom Attributes as needed.

   1. Click **New** next to the **Product Reviews Attributes** field.
   2. Enter the name of the Attribute and click **Save**.

   > **Note:** You can also define custom Product Review Attributes from the custom record at Customization > Lists, Records, & Fields > Product Reviews Attributes.

4. Select the Attributes desired for the item and then save the item record.

> **Important:** All Attributes that you want available must be highlighted when the record is saved. Once saved, the item record is re-indexed. It may take a few minutes before the rating Attributes are displayed on the website.

# Modifying Product Review Configuration Defaults

There are several modifications that can be done by editing the following two configuration files:

- Configuration.js: at Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Shopping > Reference ShopFlow > js > src > app
- backend.configuration.js: at Web Site Hosting Files/Live Hosting Files/SSP Applications/[Reference Implementation]/ssp_libraries/backend.configuration.js. For more details on the backend.configuration.js file see Configuring Backend Objects Using the backend.configuration.js File.

**Important:** These files can NOT be edited directly from the Reference ShopFlow folder. Copy the file to the same location in the Custom ShopFlow SSP application folder and edit the file there. For details, see Using the Custom Folder.

The available configuration options are self-explanatory in the following two code samples. For example, you can set whether the reviewer must be logged in before rating a product. By default a shopper does not need to be logged in to submit a review. Or, you can also set the default sort option. By default this is set to *Most Recent*.

**Configuration.js File:**

```
// Product Reviews Configuration
      // ----------------------------
   ,   productReviews: {
         maxRate: 5
      ,   computeOverall: true
      ,   reviewMacro: 'showReview'
      ,   loginRequired: false
      ,   filterOptions: [
            {id: 'all', name: _('All Reviews').translate(), params: {}, isDefault: true}
         ,   {id: '5star', name: _('$(0) Star Reviews').translate('5'), params: {rating: 5}}
         ,   {id: '4star', name: _('$(0) Star Reviews').translate('4'), params: {rating: 4}}
         ,   {id: '3star', name: _('$(0) Star Reviews').translate('3'), params: {rating: 3}}
         ,   {id: '2star', name: _('$(0) Star Reviews').translate('2'), params: {rating: 2}}
         ,   {id: '1star', name: _('$(0) Star Reviews').translate('1'), params: {rating: 1}}
         ]
      ,   sortOptions: [
            {id: 'recent', name: _('Most Recent').translate(), params: {order: 'created_on:DESC
'}, isDefault: true}
         ,   {id: 'oldest', name: _('Oldest').translate(), params: {order: 'created_on:ASC'}}
         ,   {id: 'best', name: _('Better Rated').translate(), params: {order: 'rating:DESC'}}
         ,   {id: 'worst', name: _('Worst Rated').translate(), params: {order: 'rating:ASC'}}
         ]
      }
```

**backend.configuration.js File:**

```
   ,   product_reviews: {
            // maxFlagsCount is the number at which a review is marked as flagged by users
            maxFlagsCount: 2
         ,   loginRequired: false
            // the id of the flaggedStatus. If maxFlagsCount is reached, this will be its new s
tatus.
         ,   flaggedStatus: 4
            // id of the approvedStatus
         ,   approvedStatus: '2'
            // id of pendingApprovalStatus
```

```
        ,   pendingApprovalStatus:   1
        ,   resultsPerPage: 25
    }
```

# Save For Later

Using Save For Later, users can move items from the Cart to a list for later reference and then add items from the Save for Later list directly to the Cart.

**Note:** The Save for Later feature is built on the same architecture as Product Lists.

Save for Later provides the following functionality:

- Items in the Save for Later list are restored with each login session.
- By default, users can change the quantities in the Save for Later list.
- When an item in a Save for Later list goes out of stock, that item can no longer be added to the Cart and an out of stock message is displayed.
- When a shopper is not logged in they are redirected to the Login page when the Save for Later button for an item in the cart is clicked.

For details see the following topics:

- User Experience
- Setup
- Upgrading from Previous Version
- Customizing Save For Later

## User Experience

The Save for Later feature is accessed in the Cart view. Items can be moved to the Save for Later list and then back to the Cart by logged in users.

## Setup

- Prerequisites
- Feature Architecture
- Configuration Options

### Prerequisites

By default the Save For Later module is enabled in Reference ShopFlow 1.05 and above. You must have the Product Lists Custom Record bundle installed in your account. This bundle installs the custom records used to store and manage items in Save for Later lists for each logged in user.

Before shoppers can move an Item to the Save For Later list, they are prompted to login.

## Feature Architecture

Save for Later is a type of Product List and therefore uses the same architecture and fileset as is used for Product Lists with some additions and modifications as described in this section.

### Module

The primary module used for Save for Later is ProductList.js. A new method with the Product List module controls the behaviour for Save for Later:

- `getSavedForLaterProductList`: Retrieves the Save for Later list for the logged in user.

### Router

Save for Later uses one router defined in the **Cart.Router.js** file. The following methods allow the Cart view to render the Save for Later section:

- `initialize`: the `isSavedForLater` parameter sets whether the list is a Save For Later list. Its value is stored as a local property.
- `showCart`: If `isSavedForLater` is set to true, a new view named Cart.SavedForLater.View is instantiated. If false, the standard view is rendered.

### Views

**Cart.SavedForLater.View.js** defines the view implemented for Save for Later. This view inherits from the existing Cart.Detail.View appending the new section.

This file contains the following important methods:

- `render`: Calls the parent render and also renders the SFL view
- `renderSaveForLaterSection`: If the user is logged in, it loads the SFL list and renders the section if successful
- `renderSaveForLaterSectionHelper`: This is the method which contains all the plumbing for rendering the SFL section. It basically instantiates a new ProductList.Details view with the SFL product list model. The view template must contain a data-type place holder named saved-for-later-placeholder for injecting the rendered HTML content
- `addToCart`: This is the callback which is called when an action over the SFL list occurs. It basically renders the cart for possible changes
- `saveForLaterItem`: Handles the click event of the SFL button. Removes the item from the cart and adds it to the SFL list. Action buttons are disabled while processing for preserving consistency
- `saveForLaterItemHelper`: a helper for the previous method.
- `addItemToList`: Another helper which adds the item to the product list collection. It also creates the SFL in the backend if the custom record does not exist already
- `getItemOptions`: Formats existing item options for later storage in the Product List custom record. This method should be moved to a common location
- `doAddItemToList`: Adds the item to the list

- `validateLogin`: Checks if the user is logged in or not

The following two template files define the rendered output of the Save for Later feature:

- **product_list_details_later.txt**: located at templates/product_lists/
- **product_list_details_later_macro.txt**: located at templates/product_lists/macros/

View templates: product_list_details_later.txt: Cart.SavedForLater.View.js related template. product_list_details_later_macro.txt Cart.SavedForLater.View.js related item macro.

## Services

The Save for Later feature uses the **product-list.ss** service. On a `GET`, if `id` is set to `later`, then the list with of type of later is loaded.

> **Note:** The service is coded like this since we do not know the Save for Later internalid in advance. Also, a Product List may not have been created yet when the GET is requested for the first time.

```
case 'GET':
   if (id)
   {
      if (id === 'later')
      {
         Application.sendContent(ProductList.getSavedForLaterProductList());
      }
      else
      {
         Application.sendContent(ProductList.get(id));
      }
   }
   else
   {
      Application.sendContent(ProductList.search('name'));
   }
```

# Configuration Options

By default, a Save for Later template is defined within the `product_lists` object in the **backend.configuration.js** file. You can modify the `description` and `name` parameters as desired. The type id must be set to 2 for the list to function as a Save for Later list.

```
{
   templateid: '2'
,   name: 'Saved for Later'
,   description: 'This is for the cart saved for later items'
,   scope: {
      id: '2'
   ,   name: 'private'
   }
,   type: {
```

```
      id: '2'
    ,   name: 'later'
    }
}
```

## Upgrading from Previous Version

Save for Later is a new feature as of Reference ShopFlow 1.05. No special upgrade requirements specific to this module are needed.

## Customizing Save For Later

Currently, there are no specific customization techniques used for the Save for Later feature.

However, Save for Later can be customized as needed following standard customization procedures.

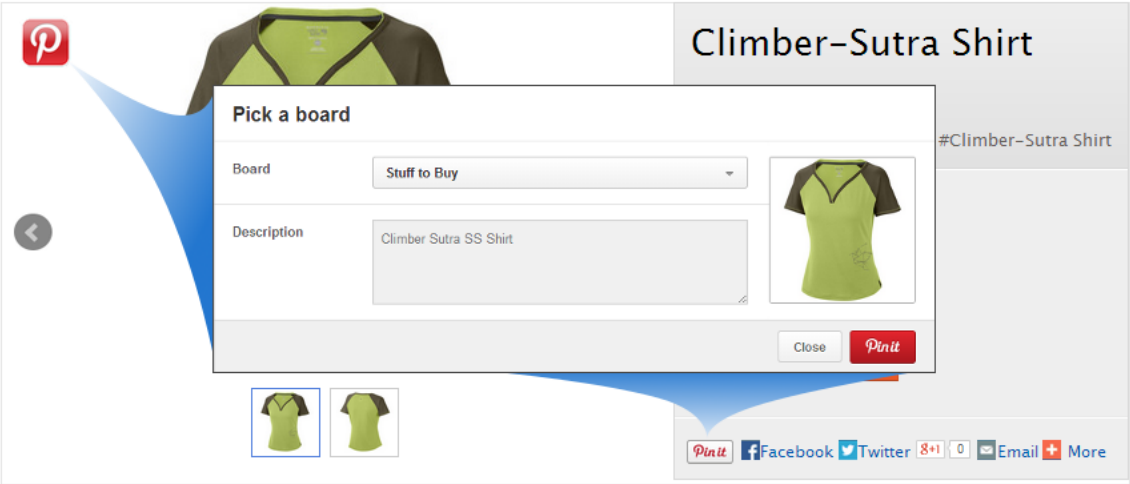
For complete details on how to customize reference implementations, see Customizing Implementations.

# Social Sharing

- Pinterest
- Twitter Product Cards

## Pinterest

Reference ShopFlow supports Product Rich Pins from Pinterest in the product details page. By default, shoppers can hover over an image and then click on a Pinterest icon, or click a Pinterest button from within the cart details region. The Pinterest entry automatically populates with item details such as the product image, pricing, or availability.



The data collection method used for Pinterest is Open Graph semantic markup. With this method, product metadata is included in the HTML header of the product page. The following metadata tag mappings for Pinterest are defined in the Configuration.js file:

```
metaTagMapping: {
         // [Open Graph](http://ogp.me/)
         'og:title': seo_title
      ,   'og:type': seo_type
      ,   'og:url': seo_url
      ,   'og:image': seo_image
      ,   'og:site_name': seo_site_name
      ,   'og:description': seo_description
      ,   'og:provider_name': seo_provider_name
      ,   'og:price:standard_amount': seo_price_standard_amount
      ,   'og:price:currency': seo_price_currency
      ,   'og:availability': seo_availability
```

For more information and details on supported fields, refer to the Pinterest developer documentation at https://developers.pinterest.com/rich_pins/.

The following screenshot shows Open Graph data collected in the HTML header output.

```
<meta property="og:title" content="Topout Shorts">
<meta property="og:type" content="product">
<meta property="og:url" content="http://coolsuite.com/product/426">
<meta property="og:image" content="http://coolsuite.com/Images/TopoutShorts_01.jpg?
resizeid=5&resizeh=150&resizew=150">
<meta property="og:site_name" content="CoolSuite">
<meta property="og:description" content="Our popular, classically-styled hiking short is better than ever.
The Canyon Short remains a favorite with its timeless combination of comfort, performance and attention to
detail. The seams at the hip have been rotated away so they'll never pinch under a pack. The integrated
webbing belt adjusts easily. UPF 50 fabric blocks out UV rays.">
<meta property="og:provider_name" content="CoolSuite">
<meta property="og:price:standard_amount" content="24.99">
<meta property="og:price:currency" content="USD">
<meta property="og:availability" content="InStock">
```

## Configuring Pinterest

Pinterest settings can be edited from within the Configuration.js file at js > src > app. You can enable Pin It Hover or the Pin It button or both options. Other configuration options are available as shown in the following code snippet. If editing the `image_size` parameter, use the available options defined from within the `imageSizeMapping` function. You can also edit the button image in the img folder. When editing this you should also edit the corresponding CSS in Shopping > skins > standard > site > home.less.

> **Note:** The Configuration.js file can NOT be edited directly from the Reference ShopFlow SSP application. Copy the file to the same location in the Custom ShopFlow SSP application folder and edit the file there. For details, see Using the Custom Folder.

```
      // hover_pin_it_buttons
   ,   hover_pin_it_button: {
         enable_pin_it_hover: true
      ,   enable_pin_it_button: true
      ,   image_size: 'main' // Select resize id to show on Pintrest
      ,   popupOptions: {
            status: 'no'
```

```
        ,   resizable: 'yes'
        ,   scrollbars: 'yes'
        ,   personalbar: 'no'
        ,   directories: 'no'
        ,   location: 'no'
        ,   toolbar: 'no'
        ,   menubar: 'no'
        ,   width: '680'
        ,   height: '300'
        ,   left: '0'
        ,   top: '0'
        }
    }
```

**Important:** Within the Configuration.js file, there is also a Pinterest setting within the `addThis.servicesToShow` function. Enable Pinterest using the `hover_pin_it_button` function instead of the `addThis.servicesToShow` function. Due to the single page configuration of Reference ShopFlow, enabling Pinterest from the `addThis` function can result in incorrect product information displayed in the users Pinterest board.

## Validating Rich Pins

To validate your Pinterest Rich Pins, use the Rich Pin Validator tool at https://developers.pinterest.com/rich_pins/validator/. Enter the URL for the product to test. The validator returns a success or error message. If the test is successful, item details that are displayed when shoppers Pin that product are shown.

# Twitter Product Cards

Reference ShopFlow supports Twitter Product Cards which enable the ability to attach media experiences to Tweets that link to product detail pages. Shoppers can Tweet your products by clicking the Twitter button on the product details page and a card visible to all of their followers is automatically added to their Tweet. This Card can show item details including an image, a description, and two other key item details such as price, availability, sizes or colors.

For more information on Twitter Product Cards see https://dev.twitter.com/docs/cards/types/product-card.

## Configuring Twitter Product Cards

Required Twitter Product Cards properties are automatically configured within the Configuration.js file. These include product details such as title, description, and image:

```
        // [Twitter Product Card](https://dev.twitter.com/docs/cards/types/product-card)
    ,   'twitter:card': seo_type
    ,   'twitter:site': seo_twitter_site
    ,   'twitter:creator': seo_twitter_creator
    ,   'twitter:title': seo_title
    ,   'twitter:description': seo_twitter_description
    ,   'twitter:image:src': seo_image
```

```
,   'twitter:domain': seo_domain
,   'twitter:data1': seo_twitter_price
,   'twitter:label1': seo_twitter_lavel_one
,   'twitter:data2': seo_availability
,   'twitter:label2': seo_twitter_lavel_two
```

The following image shows the Twitter data collected in the HTML header output.

```
<meta property="twitter:card" content="product">
<meta property="twitter:site" content="@YourAthleticSho">
<meta property="twitter:creator" content="@fedmun">
<meta property="twitter:title" content="Trekkin Thermal Henley Shirt">
<meta property="twitter:description" content="A performance twist on a classic. We designed
this shirt to combine the look and comfort of a classic long underwear top....">
<meta property="twitter:image:src" content="http://coolsuite.com/Images/Trekkin-Thermal-
Henley-Shirt_01.jpg?resizeid=5&resizeh=150&resizew=150">
<meta property="twitter:domain" content="http://coolsuite.com">
<meta property="twitter:data1" content="$49.99 USD">
<meta property="twitter:label1" content="PRICE">
<meta property="twitter:data2" content="InStock">
<meta property="twitter:label2" content="AVAILABILITY">
```

In addition to the required card properties Twitter Product Cards allow support for two custom data types. By default, these are configured to return the item price and item availability. It is recommended that you keep the default configuration for these properties.

To disable Twitter Product Cards in your web store, edit the Configuration.js file at js > src > app. Set the `enable` parameter to false in the `twitter` function. This removes the Twitter button from the product details page and disables Twitter functionality.

> **Note:** The Configuration.js file can NOT be edited directly from the Reference ShopFlow SSP application. Copy the file to the same location in the Custom ShopFlow SSP application folder and edit the file there. For details, see Using the Custom Folder.

# Reference ShopFlow Templates

This topic identifies the templates used to render each page of the Reference ShopFlow experience.

> **Important:** Web development experience is required to modify any template file installed by the Shop Flow Reference Implementation. Also, many basic customizations can be done by customizing the **Configuration.js** file at js > src > app.

All the files described in this section can be found in the templates folder after the Reference ShopFlow bundle has been installed. To access template files, go to Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Shopping > Reference Shopping > templates.

Note that all the template files in the Reference ShopFlow application folder are locked. To modify these files, copy them to the Custom ShopFlow folder. For details, see Customizing Implementations.

The **layout.txt** template is the container for all of the Reference ShopFlow pages and is used to generate all of the content between the header and footer of each page.

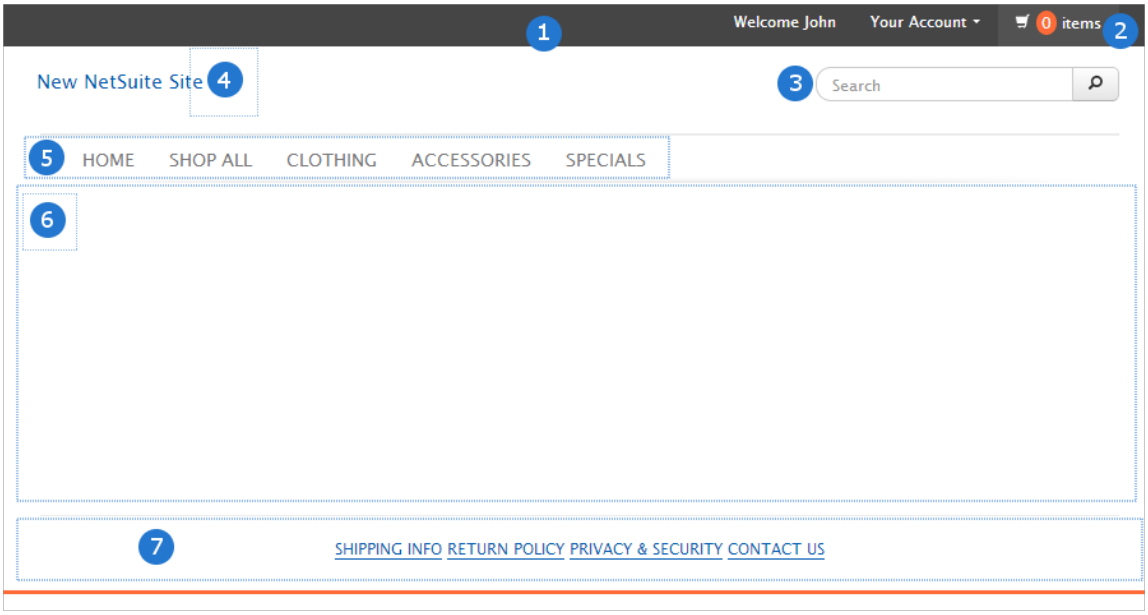
The topics below include more information about each template installed with Reference ShopFlow:

- Home Page
- Product Display Page

- Product Details Page
- Product Reviews Pages
- Shopping Cart Page

# Home Page

The page customers see when they visit the home page of your website is generated using the *home.txt* template.



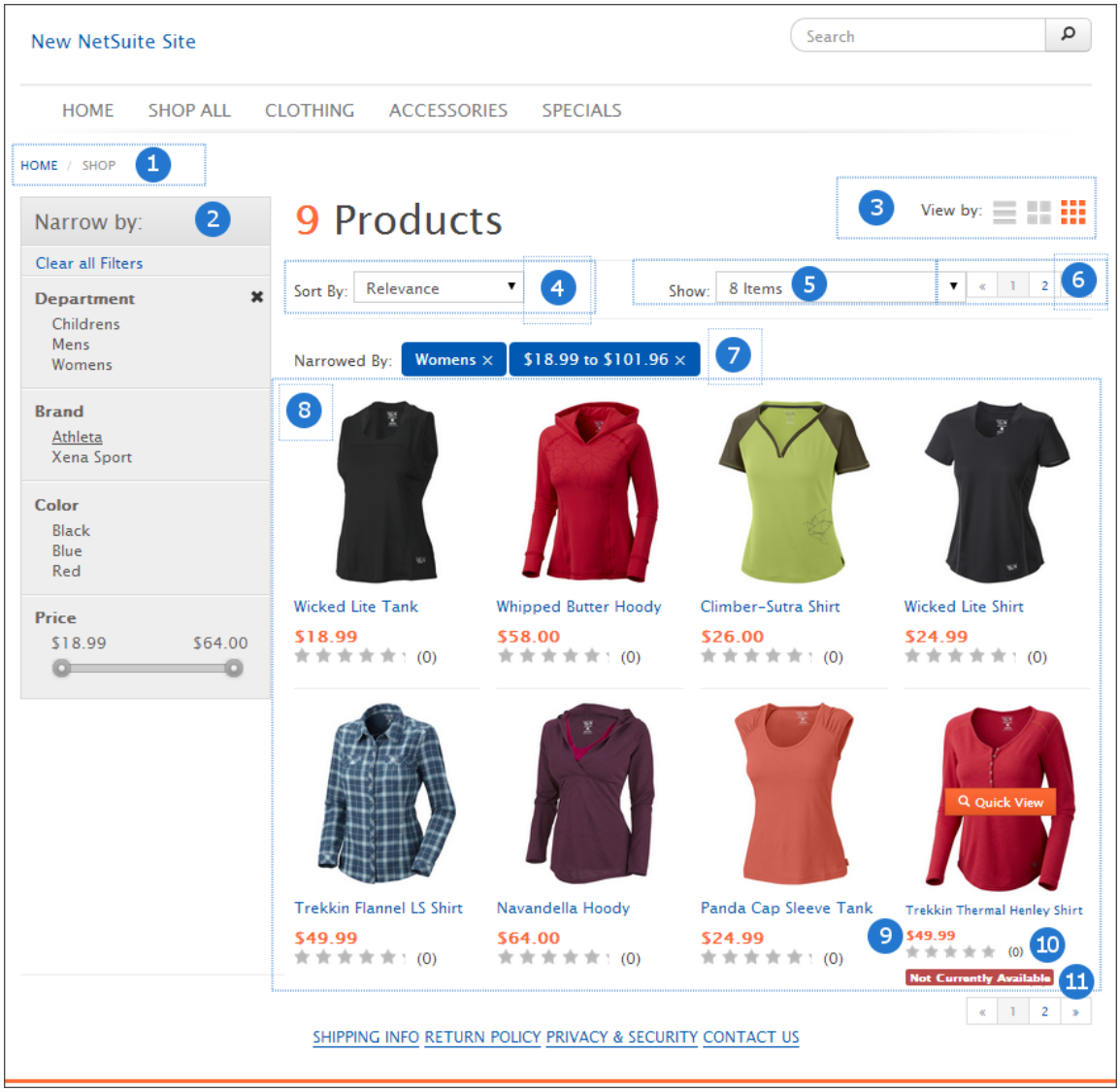

| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. Header | If Reference My Account is installed, Customers can access links to all My Account pages. | header/macros/ header_profile.txt | macro |
| 2. Mini Cart | This macro displays a shopping cart image with the number of items in the cart. Clicking this image opens the mini cart with a list of items and their prices. | mini_cart_macro.txt | macro |
| 3. Site Search | This macro presents the search field customers can use to search your website. | site_search_macro.txt | macro |
| 4. Logo | By default, displays the Web Site name as defined on the Web Site record. To display a custom logo, set the `logoUrl` parameter in the Configuration.js file using an absolute URL. For example: http://www.mysite.com/customlogo.png<br>**Note:** You must specify an **absolute URL** for custom logos to appear correctly on your site. | header/macros/ logo_macro.txt | macro |
| 5. Navigation Tabs | By default, Home and Shop tabs are displayed. Tabs can be configured in the Configuration.js file. | header/macros/ navigation_tabs_macro.txt | macro |

| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 6. Home Page Content | By default, this area is empty. | home/ home.txt | home_tmpl |
| 7. Footer | By default, this area is empty. You can configure standard footer navigation links in the Configuration.js file. | footer/ footer_macro.txt footer_navigation.txt | macro |

# Product Display Page

When customers search for items or navigate to a category, they can narrow results on this page by using faceted navigation. From this page, customers can narrow the results by selecting values from the facets in the Narrow By pane, change how results are displayed, and sort the results.

The main body template for this page is **facet_browse.txt**. The number of products displayed in each row of results is controlled by **display_in_rows_macro.txt**.



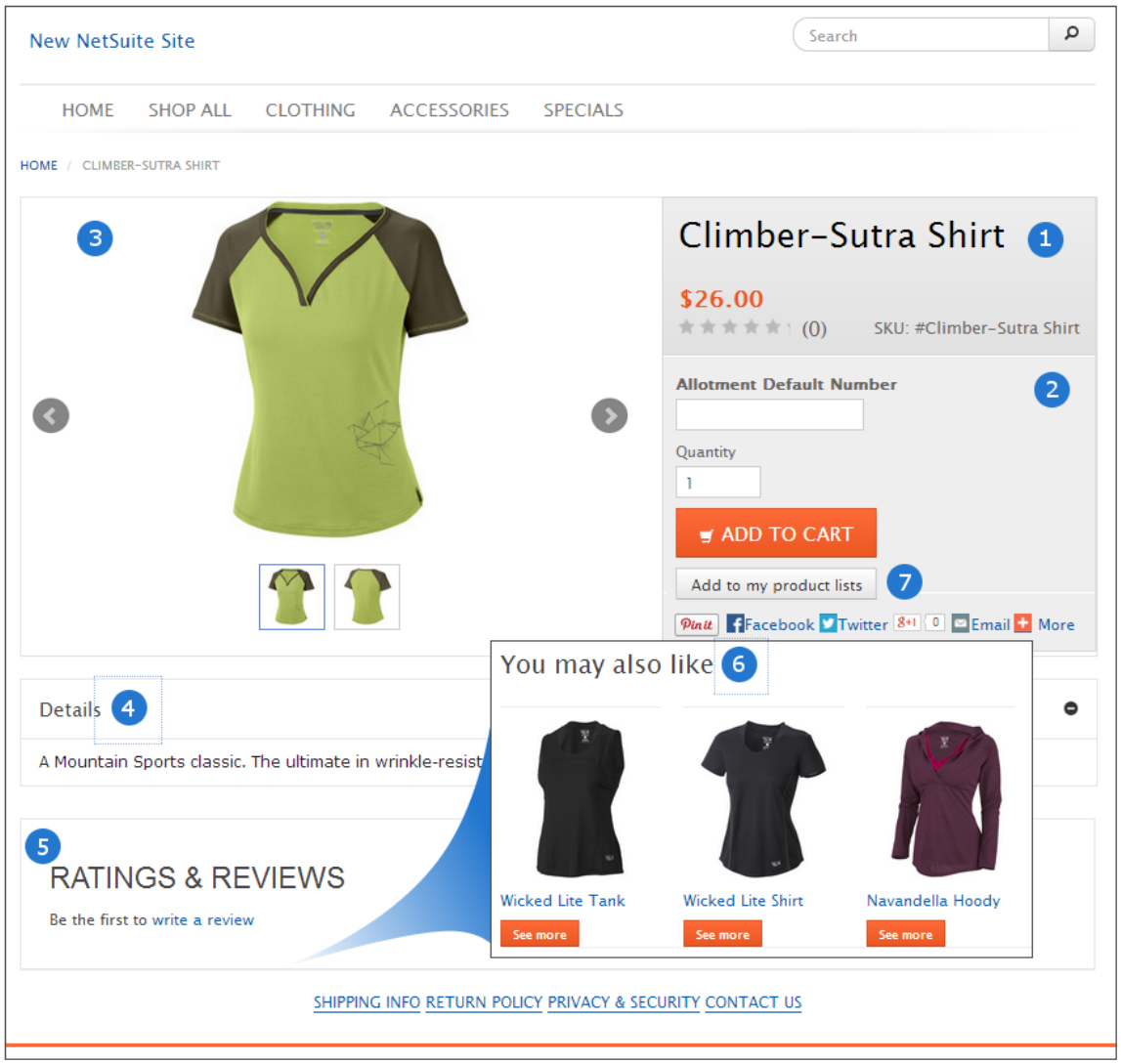
| Element | Description | Template File(s) | Metadata/Tag |
|---|---|---|---|
| 1. Breadcrumb Navigation | Updates based on active page within the ShopFlow experience. Clicking a breadcrumb link directs the customer to the target page. The active page is not a hyper link. | globals/macros/breadcrumb_macro.txt | macro |
| 2. Narrow By | Contains the facets and the values available for the products shown. Selecting or changing options updates the list of products instantly. Facets can be single or multi-select. Multi-select facets have check boxes for each option. In cases where there are many options, the list of values will have a scroll bar.<br>For detailed information on working with this panel, see Working with Faceted Navigation. | There are multiple facet macros available for configuration as indicated by the prepended `facet_` at: items_lists/macros. | macro |
| 3. View By | The customer can choose to display the results as a list, table, or grid. Grid is the default. | items_list/macros/item_listing_display_selector.txt | macro |
| 4. Sort By | Customers can sort the results by the following:<br>▪ Relevance<br>▪ Price, Low to High<br>▪ Price, High to Low<br>You can add custom sorting options by creating search API sort filters and adding them to the `sortOptions` parameter in the Configuration.js file. | items_list/macros/item_listing_sort_selector.txt | macro |
| 5. Show | Customers can set the number of items to display. The available options can be configured in the `resultsPerPage` parameter of the Configuration.js file. | items_list/macros/item_list_show_selector_macro.txt | macro |
| 6. Pagination | Pagination is only displayed if the number of results is greater than the maximum number of results per page you have set.<br>The page of results you are viewing is shown first followed by the total number of pages of results. Customers can enter a page to navigate to or click the forward and back arrows. | globals/macros/pagination_macro.txt | macro |
| 7. Facet Display | Displays the currently selected filter criteria for the page. | items_list/macros/facets_display_macro.txt | macro |
| 8. Products Displayed | All products are displayed based on the current Search API results.<br>**Note:** Certain actions within NetSuite initiate a rebuild of the search index. Actual search results may reflect cached information until a refresh of the index or local browser cache is performed. | One of the following:<br>▪ items_list/macros/item_listing_display_grid.txt<br>▪ items_list/macros/item_listing_display_list.txt<br>▪ items_list/macros/item_listing_display_table.txt | macro |
| 9. Price | This area displays the currently selected filter criteria. | globals/macros/show_item_price_macro.txt | macro |
| 12. Rating | For detailed information on configuring reviews, see Product Reviews. | item_details/macros/item_ | macro |

| Element | Description | Template File(s) | Metadata/Tag |
|---|---|---|---|
| | | details_reviews_macro.txt | |
| 11. Out of Stock | Out of stock messaging is configured on a per item basis. You can define the out of stock behavior for each item on the Web Store tab of the Item record at Lists > Web Site > Items. | item_details/macros/item_details_stock_macro.txt | macro |

# Product Details Page

The main body template for the product details page is **item_details.txt**.

On this page, customers can view product images, select product options, add the product to the shopping cart or wish list, and share the product through email or social networking sites like Facebook, or Pinterest. They can also see a preview of product reviews.



| Element | Description | Template File(s) | Metadata/Tag |
|---|---|---|---|
| 1. Product Information | The product information panel includes all of the information needed for the customer to make a purchase.<br>There are several macros included in the reference implementation that you can use for product options. These options are configured in Configuration.js file. | item_details/ item_details.txt<br>There are multiple item detail macros available for configuration as indicated by the prepended `item_details_` at: items_details/macros. | product_details_tmpl macro |
| 2. Cart Information | Displays further product details and cart information using the same macros as those for the Product Information panel. | item_details/ item_details.txt<br>There are multiple item detail macros available for configuration as indicated by the prepended `item_details_` at: items_details/macros. | product_details_tmpl macro |
| 3. Product Image | The default product image is shown here, and alternate images are presented in an image carousel below. This image carousel loops around to the beginning after the last image.<br>If there are no alternate images, the carousel is not shown.<br>**Note:** Any videos included in the carousel are opened in a modal light box.<br>For detailed information on setting up images for your SuiteCommerce Advanced site, see the help topic SCA Product Images | item_details/ macros/ item_image_gallery.txt | macro |
| 4. Product Details | The product details are collapsible panes. If any of the other collapsible panes on this page contain no information, they are not shown. | item_details/ item_details.txt | product_details_tmpl |
| 5. Product Reviews | Customers can view or add product review from this section.<br>For detailed information on Product Reviews, see Product Reviews. | item_details/ macros/ item_details_reviews.txt | macro |
| 6. You may also like | If an image has related items or correlated items associated with them, those items are displayed here.<br>For detailed information on related items, see Correlated & Related Items. | globals/macros/ related_item_macro.txt | macro |

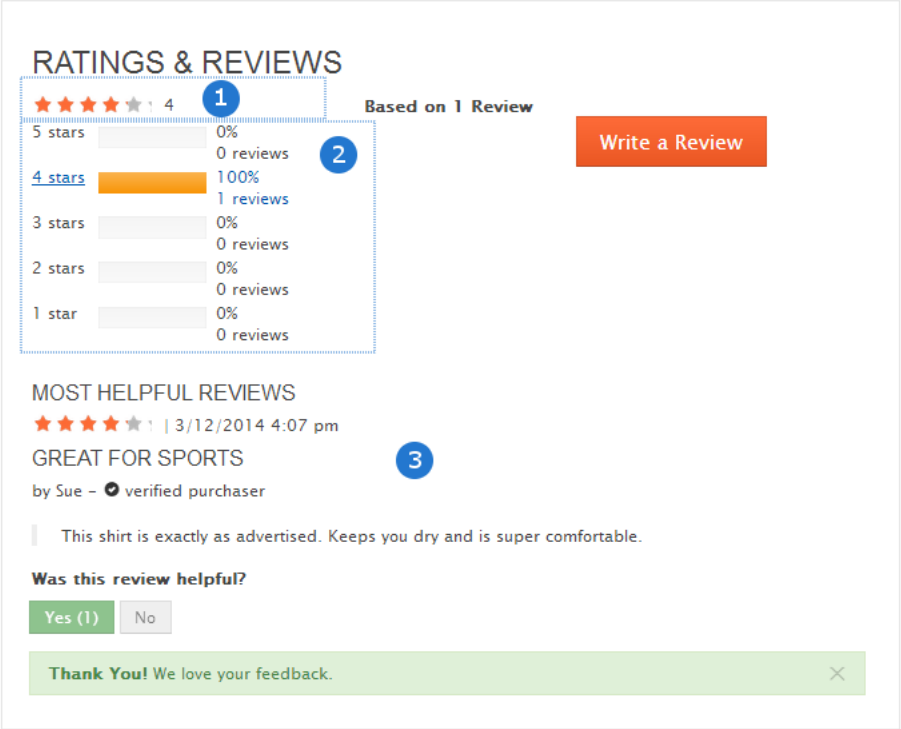
# Product Reviews Pages

If product reviews are available for an item, those reviews are displayed on the product details page. A customer can add a review or tag a review as helpful.

**Note:** Depending on your NetSuite account setup, reviews may need to be approved before they display on the site.

The main body template for this page is **item_details_reviews_macro.txt**.

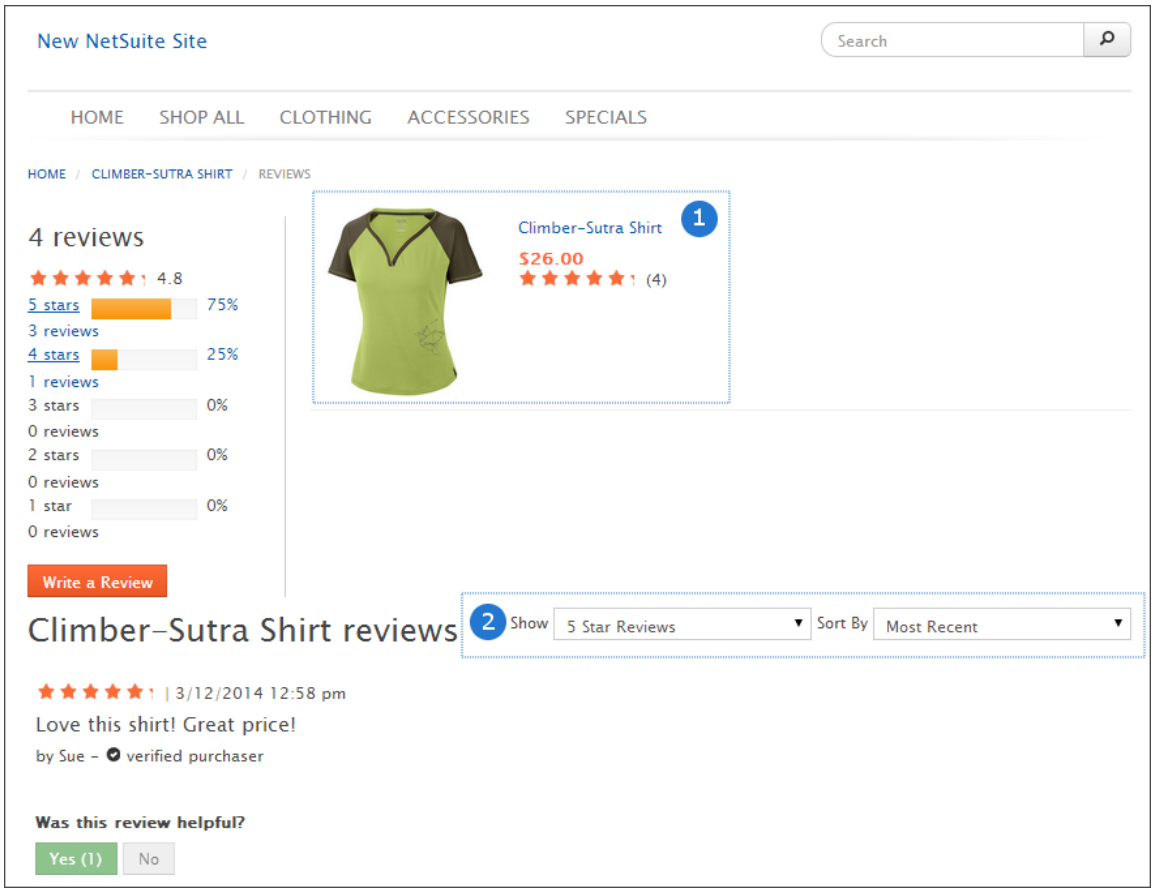
For detailed information on working with Product Reviews see Product Reviews.




| Element | Description | Template File(s) | Metadata/Tag |
|---|---|---|---|
| 1. Average Overall Rating | This rating averages all of the ratings for this product's reviews. | globals/macros/star_rating_macro.txt | macro |
| 2. Ratings by Star | This macro displays the percentage of ratings for each star. | globals/macros/rating_by_star_macro.txt | macro |
| 3. Review Details | Each product review is listed. Customers can filter reviews or sort them. Reviews are paginated. | globals/macros/show_review_macro.txt | macro |

## Product Review Details Page

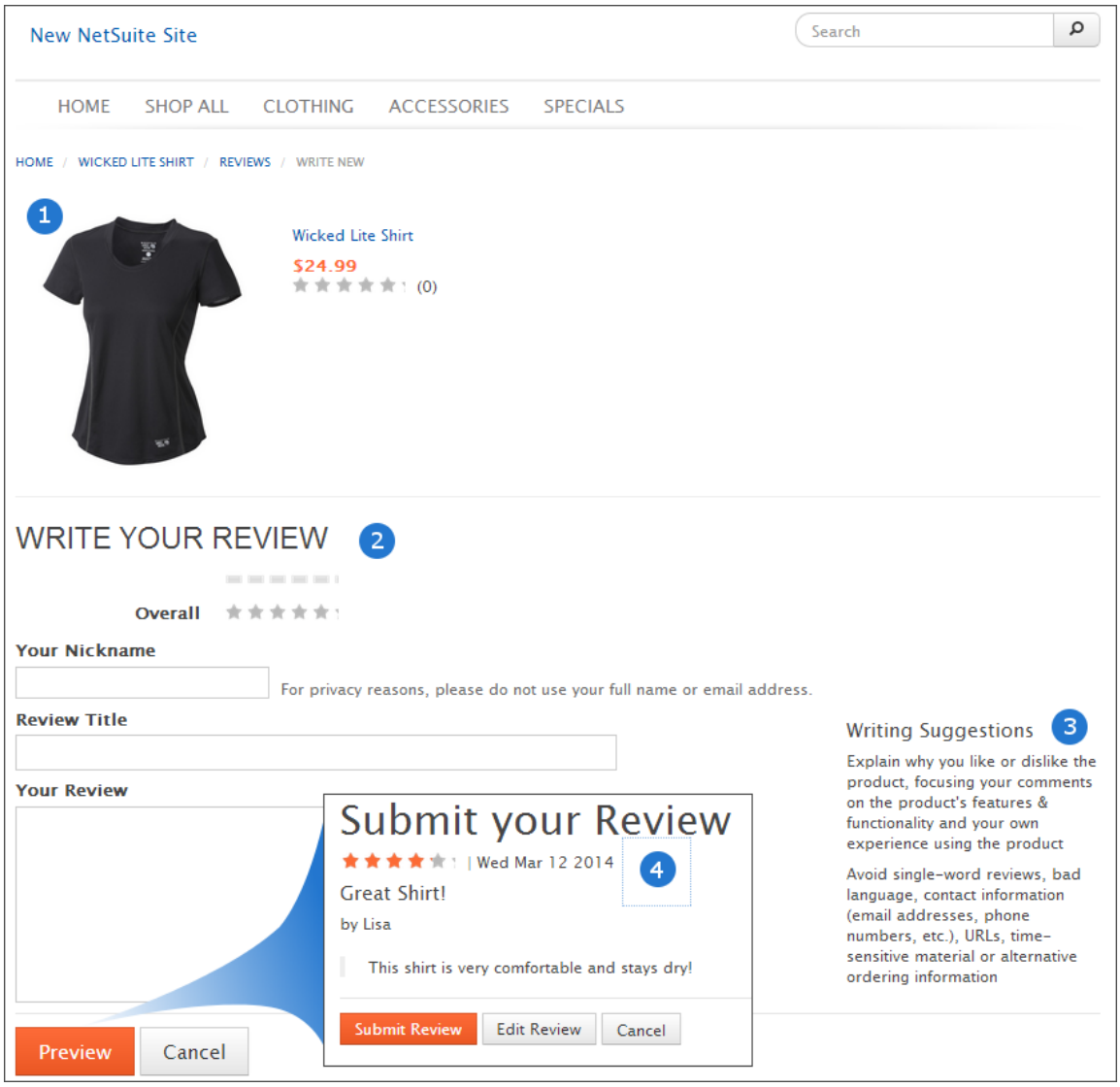
When a customer drills down into the star rating links, the Product Reviews Details page is displayed listing all reviews. Reviews are paginated if needed and can be filtered.




| Element | Description | Template File(s) | Metadata/Tag |
|---|---|---|---|
| 1. Product Details | Displays product details. | items_list/macros/ item_cell_list_macro.txt | macro |
| 2. Review Filters | Customers can filter and sort reviews. Available options for these can be set by editing the `productReviews` parameter in the Configuration.js file. | reviews/ reviews_center_for_item.txt | reviews_center_for_item_tmpl |

## Write a Review Page

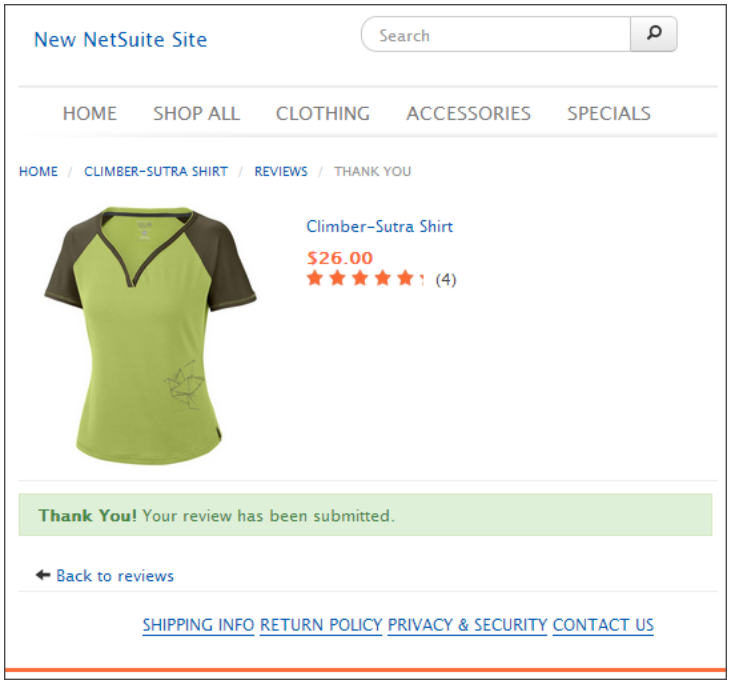
Clicking the Write a Review link takes the customer to the customer review page. Customers can assign star ratings, enter a nickname, and enter a title and text for their review. The main body template for this page is **review_form.txt**.




| Element | Description | Template File(s) | Metadata/Tag |
|---|---|---|---|
| 1. Product Display | The primary image for the product is shown along with the name, price, sales price, and a link back to the product details page. | items_list/macros/ item_cell_list_macro.txt | macro |
| 2. Review Form | Displays the form for inputting review comments. | reviews/review_ form.txt | review_form_ tmpl |
| 3. Ratings by Star | This macro displays the percentage of ratings for each star.<br>By default, the overall rating is calculated by averaging the three rating categories. You can disable the calculation in the `productReviews` parameter in Configuration.js file. | globals/macros/ star_rating_macro .txt | macro |
| 4. Review Preview | Customers can preview their review before submission. | reviews/ review_form_preview.txt | review_form_ preview_tmpl |

## Product Review Confirmation Page

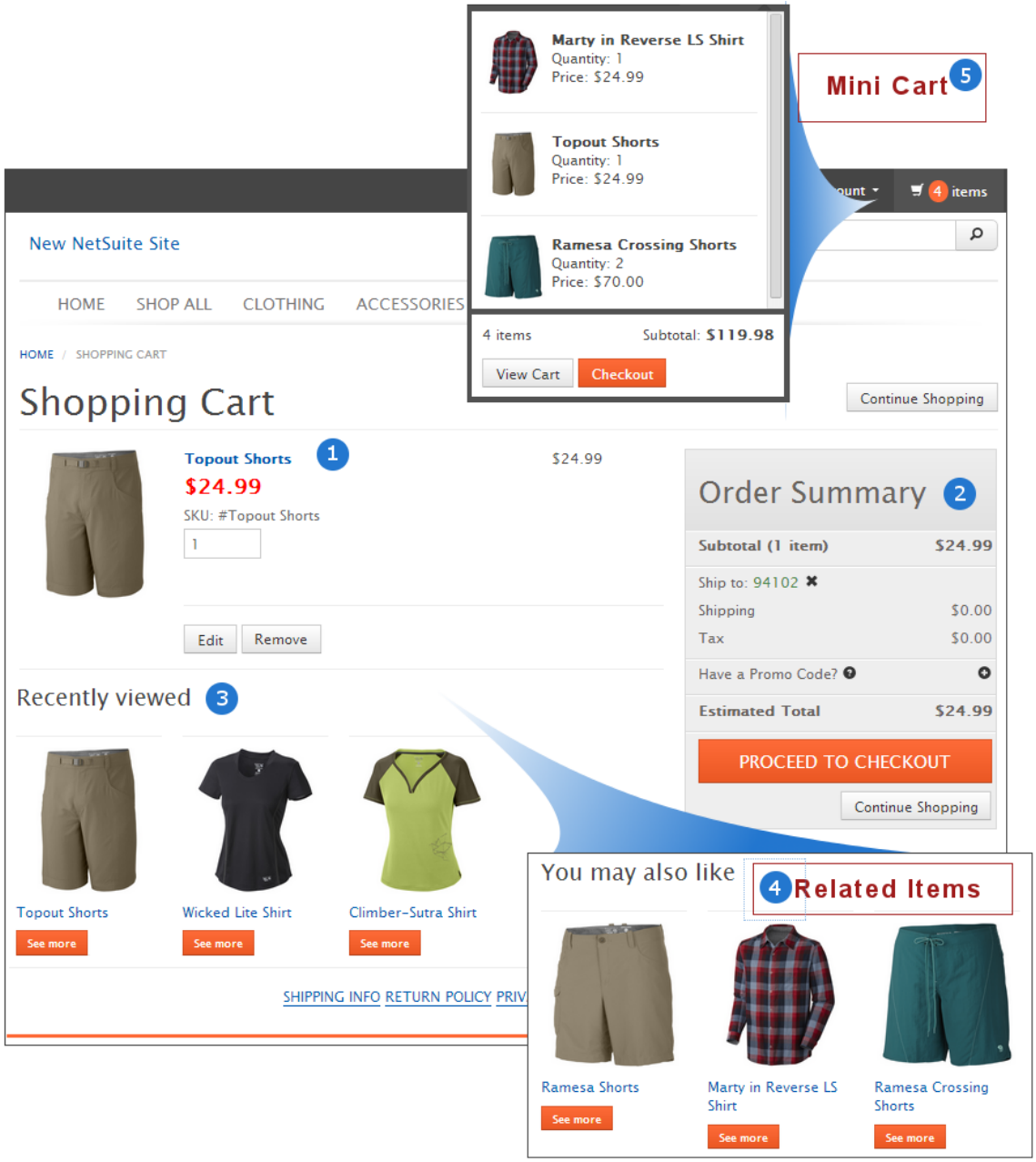
After customers have saved a product review, they are presented with a confirmation. The main body text for this page is **review_form_confirmation.txt**.




# Shopping Cart Page

The Reference ShopFlow shopping cart allows customers to view the items they are purchasing, estimate tax and shipping, and enter promotion codes. Customers are also presented with lists of items they might want to purchase based on the contents of the cart or view items they recently viewed.

The main body template for the Shop Flow shopping cart is **shopping_cart.txt**.



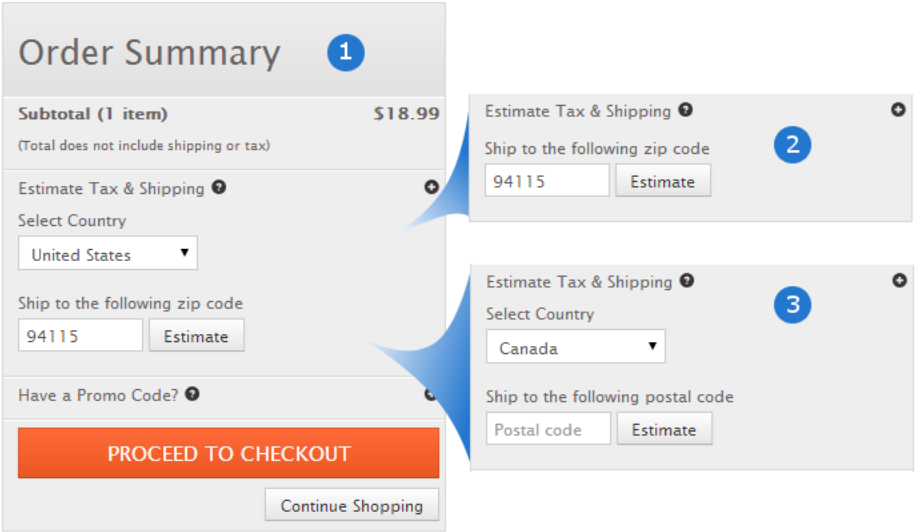

| Element | Description | Template File(s) | Metadata/Tag |
|---|---|---|---|
| 1. Shopping Cart Items | The shopping cart items details are displayed. | shopping_cart/ macros/ shopping_cart_item_macro.txt Additional item macros that can be used:<br>▪ shopping_cart/ macros/show_ item_price_ macro.txt<br>▪ shopping_cart /macros/cart_ | macro (all) |

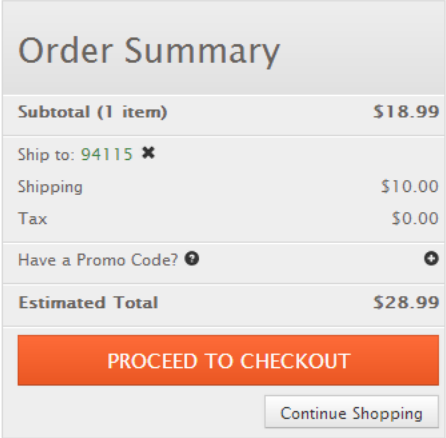
| Element | Description | Template File(s) | Metadata/Tag |
|---|---|---|---|
| | | option_default_macro.txt<br>■ cshopping_cart/macros/art_option_color_macro.txt | |
| 2. Order Summary | This macro shows Order Summary. Customers can expand sections to estimate tax and shipping and enter a promotion code. | shopping_cart/macros/ shopping_cart_summary_macro.txt | macro |
| 3. Recently Viewed | Displays products viewed in the current session. | globals/macros/ recently_viewed_macro.txt | macro |
| 4. You may also like | If an image has related items or correlated items associated with them, those items are displayed here.<br>For detailed information on related items, see Correlated & Related Items. | globals/macros/ related_item_macro.txt | macro |
| 5. Mini-Cart | While shopping, customers can click the cart in the upper right corner of the page to view the mini-cart and the items they have added. | header/macros/ mini_cart_macro.txt | macro |

## Tax and Shipping Estimate

In the Order Summary of the shopping cart, customers can expand the Estimate Tax & Shipping section to enter their country and zip/postal code to estimate tax and shipping costs. This section uses *shopping_cart_summary_macro.txt*.

If the zip/postal code entered is invalid, the customer is presented with an error message. The estimated total of the order is updated after shipping and tax are estimated.



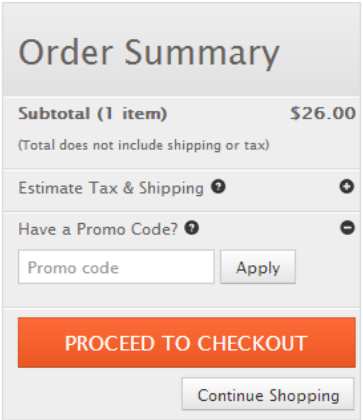



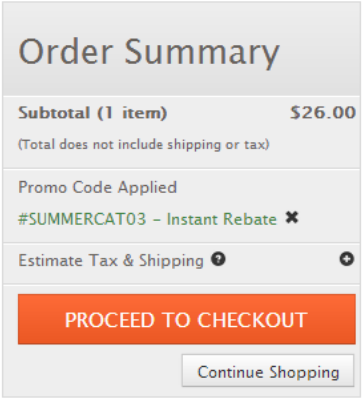

| Element | Description | Template File(s) | Metadata/Tag |
|---|---|---|---|
| 1. Order Summary | This template presents information on the items that have been added to the cart. | shopping_cart/macros/ shopping_cart_summary_macro.txt | macro |
| 2. US Only Shipping | Country drop down is removed for US only sites. | | |
| 3. International Aware Shipping | Changes zip code to postal code. | | |

## Promotion Code Entry

Customers can expand the Have a Promo Code section of the Order Summary to apply promotion codes on the shopping cart page. This section uses the **shopping_cart/macros/promocode_form_macro.txt** macro.

If customers enter an invalid promotion code, they are presented with an error message. When a promotion code is successfully applied, the estimated total of the order is updated.


Order Summary
Subtotal (1 item)
$26.00
(Total does not include shipping or tax)
Estimate Tax & Shipping
Have a Promo Code?
Promo code
Apply
PROCEED TO CHECKOUT
Continue Shopping


Order Summary
Subtotal (1 item)
$26.00
(Total does not include shipping or tax)
Promo Code Applied
#SUMMERCAT03 – Instant Rebate
Estimate Tax & Shipping
PROCEED TO CHECKOUT
Continue Shopping

# Reference My Account

**Important:** When upgrading to a new bundle version, refer to the Denali Release of SuiteCommerce Advanced for information on enhancements and fixes in that bundle.

The SuiteCommerce Reference My Account Reference Implementation delivers all the components you need to create a personalized experience for users on your e-commerce website. You can use Reference ShopFlow as the basis for creating your own SuiteCommerce website and then use Reference My Account to handle account management for that site. Reference My Account can also be used with Site Builder websites.

With Reference My Account, customers can track their orders, view order details, view and edit their shipping preferences, and view and edit their billing and credit card information. They can also create and manage Product Lists, pay against invoices and print statements.

This bundle includes:

- Responsive Design Layout (desktop, tablet, mobile) using SSP architecture and modern JavaScript Libraries
- Pages for Order History & Returns, Receipts, and Reorder Items
- Product List (WishList) Management
- Billing Tools (Account Balance, Invoicing, Transaction History and Printing Statements)
- Profile Information (update name, phone number, email)
- Address Book (multiple shipping and billing addresses)
- Credit Card (storage of credit cards)
- Email Preferences

**Important:** Features available vary depending on whether you have the Reference My Account or Reference My Account Premium edition. For details, see Reference My Account Editions. Also, multi-currency is not supported in Reference My Account.

Read the topics listed below to learn how install Reference My Account, and how to identify and work with the components installed:

- Managing Reference Implementation Installations
- SuiteCommerce Reference Implementation Web Assets
- Touch Points Available for Reference Implementations
- Working with Permissions
- Product Lists
- Billing
- Reference My Account Templates

## Working with Permissions

By default, Reference My Account honors all server-side NetSuite roles and permissions available for the currently logged in user. Permissions attributes are set for each view as needed and then in the template files those permissions attributes are used to determine which links to print client-side. For example, if a user does not have permission to view the Order History in NetSuite, the Order History & Returns link is not displayed in the My Account menu for that user.

Permissions can be customized as desired. You can customize permission levels in existing templates by defining or modifying permission attributes for any HTML segment to be rendered. You can also add permissions attributes to custom views you create. Multiple data-permission attributes can be applied, separated by a comma or a space. All listed permissions must be met before the user is given access. For example, in the following code snippet the div to be rendered will only display when the permissions for the currently logged in user matches both data-permissions attributes listed.

```
<div class="recentOrdersWidget" data-permissions="transactions.tranFind.1,transactions.tranSale
sOrd.1">
```

The `transactions.tranFind.1` attribute defines the role (transactions), the permission (tranFind), and the level of access (1). Access can be set from 0–4, where:

- 0 — no permission
- 1 — view only
- 2 — create
- 3 — edit
- 4 — full

**Important:** When no permissions attribute is set, all users will have access to the view. It is important that you set permissions for every new custom view implemented when restricted access to the rendered data is needed.

To learn more about permissions see the help topic NetSuite Permissions Overview.

## Permissions Function

The permissions function which sets the permission for the current user is found in the **commons.js** file in the ssp_libraries folder. This function checks each permission available in the customer center. You can add any of these permissions to HTML in template files as needed and set the minimum access level required.

```
result.permissions =
      {
         transactions:
         {
               tranCashSale : context.getPermission('TRAN_CASHSALE')
            ,   tranCustCred : context.getPermission('TRAN_CUSTCRED')
            ,   tranCustDep : context.getPermission('TRAN_CUSTDEP')
            ,   tranCustPymt : context.getPermission('TRAN_CUSTPYMT')
            ,   tranStatement : context.getPermission('TRAN_STATEMENT')
            ,   tranCustInvc : context.getPermission('TRAN_CUSTINVC')
            ,   tranItemShip : context.getPermission('TRAN_ITEMSHIP')
            ,   tranSalesOrd : context.getPermission('TRAN_SALESORD')
            ,   tranEstimate : context.getPermission('TRAN_ESTIMATE')
            ,   tranRtnAuth : context.getPermission('TRAN_RTNAUTH')
            ,   tranDepAppl : context.getPermission('TRAN_DEPAPPL')
            ,   tranSalesOrdFulfill : context.getPermission('TRAN_SALESORDFULFILL')
            ,   tranFind: context.getPermission('TRAN_FIND')
         }
      ,   lists:
         {
               regtAcctRec: context.getPermission('REGT_ACCTREC')
```

```
            ,   regtNonPosting: context.getPermission('REGT_NONPOSTING')
            ,   listCase: context.getPermission('LIST_CASE')
            ,   listContact: context.getPermission('LIST_CONTACT')
            ,   listCustJob: context.getPermission('LIST_CUSTJOB')
            ,   listCompany: context.getPermission('LIST_COMPANY')
            ,   listIssue: context.getPermission('LIST_ISSUE')
            ,   listCustProfile: context.getPermission('LIST_CUSTPROFILE')
            ,   listExport: context.getPermission('LIST_EXPORT')
            ,   listFind: context.getPermission('LIST_FIND')
            ,   listCrmMessage: context.getPermission('LIST_CRMMESSAGE')
         }
      };
      return result;
   }
```

## Example Permissions — Order History

**Order History Module Code:**

In the following code snippet, permissions are defined such that the user must at least have view permissions for transaction find (`transactions.tranFind.1`) and transaction sales order (`transactions.tranSalesOrd.1`).

```
// OrderHistory.js
// -----------------
// Defines the Order  module (Model, Collection, Views, Router)
define('OrderHistory', ['OrderHistory.Views','OrderHistory.Router'], function (Views,  Router)
{

   'use strict';

   return   {
      Views: Views
   ,   Router: Router
   ,   MenuItems: {
         id: 'ordershistory'
      ,   name: _('Order History &amp; Returns').translate()
      ,   url: 'ordershistory'
      ,   index: 3
      ,   permission: 'transactions.tranFind.1,transactions.tranSalesOrd.1'
      }

   ,   mountToApp: function(application) {
         return new Router(application);
      }
   };
});
```

**Layout.txt Template Code:**

The Layout.txt template uses the permissions set for each module to determine if a link for that view should be rendered in the sidebar menu.

```
<ul id="sidebar" class="nav nav-tabs nav-stacked">
<% _.each(view.options.Layout.getMenuItems(), function (item, index) { %>
```

```
   <li data-label="<%= item.id %>" data-permissions="<%= item.permission %>">
      <a href="/<%= item.url %>" data-hashtag="#<%= item.url %>" data-touchpoint="customercente
r" target="<%= item.target %>">
         <%= item.name %>
      </a>
   </li>
<% }) %>
</ul>
```

**header_profile_macro.txt Code:**

The header_profile_macro.txt macro defines permissions to determine if a link for that view should be rendered in the My Account menu in the header.

```
<li class="my-account-item-cell" data-permissions="transactions.tranFind.1,transactions.tranSal
esOrd.1">
   <a href="#" data-touchpoint="customercenter" data-hashtag="#ordershistory">
      <%= _('Order History & Returns').translate() %>
   </a>
</li>
```

## Customizing the Forbidden Access Error Page

The forbidden_error.txt template at templates/globals defines the page to render when permissions are not sufficient for the currently logged in user for that entire page. By default this templates renders a message like the following. You can customize this template as needed for your web store.

*"Sorry! You have no permission to view this page. Please contact the website administrator, click here to continue."*

> **Note:** This page is rendered only in cases where a user may have accessed a link to the page even though they do not have permissions to view that page. Normally, links do not render for these users since the links themselves have permissions attributes tied to them as described above.

# Product Lists

Product Lists (Wish Lists) provide the ability to group lists of items to be purchased from your web store. Using product lists, users in a B2B or B2C can do the following:

- Flag Items to be added to a product list as they are browsing in your web store.
- Create new lists on the fly when flagging items.
- Edit Item order details from within the Product Lists pages.
- Move Items between Product Lists.
- Add an entire list of products to the shopping cart with a single click.
- Use Product Lists to Reorder lists of items.

> **Note:** Product Lists are supported for SuiteCommerce Advanced Sites only.

With Product Lists, users can add items to either a predefined Product List (Wish List) or to their own private list:

- **Predefined Product Lists**: Predefined Lists are lists that are automatically available to your web store users. By default, a single predefined list, My List, is available. You can define multiple predefined lists for your store or configure your store with no predefined lists so that user must always create their own.
- **Private Product Lists**: By default, users can create their own private lists. They can add these lists on the Product Lists pages or create them on the fly when adding an item to a list. You can disable the ability for users to create their own private lists.

For example, you could provide two predefined lists for your web store, such as Gifts and Favorites. A user could then add to either of these predefined lists or they could create a new list, such as Christmas, and add the item to that list.

**Note:** Currently, both Predefined and Private product lists can only be viewed by the currently logged in user within a web store.

For more information on configuring a site for Predefined and Private lists, see Configuring Product Lists.

## Product List Custom Records

When using the Product List feature, custom records are used to store and manage product lists for each Customer. These custom record are accessed from the custom **SuiteCommerce Extensions** tab of the Customer record at Lists > Relationships > Customers > [customer record]. The Custom record components are installed as a separate bundle, **Reference Product Lists Records**. For detailed information on installing this bundle, see Installing New Reference Implementations.

The following custom record components are added on installation of the bundle:

- Lists:
  - **Product Lists Items Priority**: Defines the priority of each list. Currently this field is unsupported but in future releases this field will be used to enable the ability for sorting items within a list based on priority.
  - **Product Lists Scope**: Defines the scope for the list as private, public or shared. Currently, only private lists are supported and each list is set as private by default. Private lists can be viewed and managed by the currently logged in user only. Public and shared lists will be supported in future releases.
  - **Product Lists Type**: Defines the list as default, later, or predefined. All lists added by a user are automatically set to **default**. Lists set as **predefined** are available as a list for the user by default without them needing to create it. The **later** type is not currently implemented.

    **Note:** We recommend defining **predefined** lists in the backend.configuration.js file as described in Configuring Product Lists. When predefined lists are defined via the backend.configuration.js file, the list is automatically available to all users and the list is not defined on any given customer record until that customer has added an item to that predefined list. When adding a predefined list to a customer record, it is available only for that customer.
- Record Types:
  - **Product Lists**: This record is used to manage product list data.
  - **Product Lists Items**: This record is used to manage item data associated with a product list record.

- **Subtabs**: A single SuiteCommerce Extensions entity subtab is used to display Product List data on the Customer record. This subtab is also used to display Product Review data on Item records.

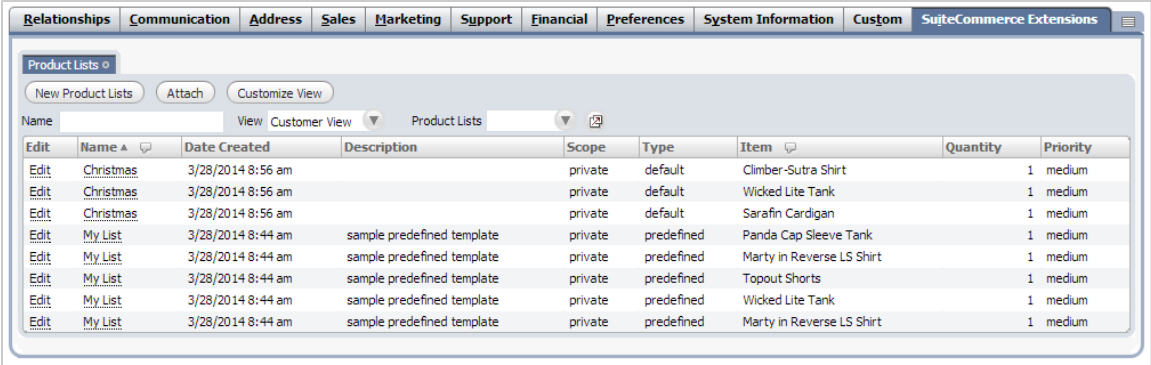
You can customize these records as needed. For detailed information on working with custom records, see the help topic SuiteBuilder (Customization).

> **Note:** If the Product List Custom Record bundle is not installed, Product Lists are not available in the Product Lists menu of Reference My Account and errors are received when a user attempts to add an Item to a Product List with Reference ShopFlow.

## SuiteCommerce Extensions Subtab

From the SuiteCommerce Extensions > Product Lists Subtab on the Customer Record you can manage Product Lists. You can drill down into each List and modify list attributes or modify Item attributes as needed.

> **Important:** Generally, these records are intended to store and display data gathered from the web store front end and in most cases should not be edited directly.




| Edit | Name | Date Created | Description | Scope | Type | Item | Quantity | Priority |
|---|---|---|---|---|---|---|---|---|
| Edit | Christmas | 3/28/2014 8:56 am | | private | default | Climber-Sutra Shirt | 1 | medium |
| Edit | Christmas | 3/28/2014 8:56 am | | private | default | Wicked Lite Tank | 1 | medium |
| Edit | Christmas | 3/28/2014 8:56 am | | private | default | Sarafin Cardigan | 1 | medium |
| Edit | My List | 3/28/2014 8:44 am | sample predefined template | private | predefined | Panda Cap Sleeve Tank | 1 | medium |
| Edit | My List | 3/28/2014 8:44 am | sample predefined template | private | predefined | Marty in Reverse LS Shirt | 1 | medium |
| Edit | My List | 3/28/2014 8:44 am | sample predefined template | private | predefined | Topout Shorts | 1 | medium |
| Edit | My List | 3/28/2014 8:44 am | sample predefined template | private | predefined | Wicked Lite Tank | 1 | medium |
| Edit | My List | 3/28/2014 8:44 am | sample predefined template | private | predefined | Marty in Reverse LS Shirt | 1 | medium |

Note the following:

- Editing Product List attributes modifies that list for the current Customer only even if the List is a predefined template. To modify available predefined templates attributes for all users you must edit the attributes in the backend.configuration.js file as described in Configuring Product Lists.

  Also, once a predefined template is modified or an item is added to a predefined product list in a Customer record, the existing template attributes take precedence over the corresponding predefined template attributes in the backend.configuration.js file.

  For example, to change the Christmas list name to Christmas 2014 for the Customer record below, drill down into the Christmas list field and change the name field to Christmas 2014. Similarly, to change the My List list predefined template name for the current Customer only, drill down into the My List record and change the name field as needed. To change the My List record name attribute for all users, edit the `name` parameter for that template in the backend.configuration.js file.

  > **Note:** Predefined templates are not intended to be modified on a per customer basis. To modify predefined templates, you should always edit the attributes in the backend.configuration.js file to ensure changes are available for all customers.

- The scope attribute for a Product List is not supported by default in the Reference ShopFlow or Reference My Account front end. Changing this setting has no impact.

- The Item Options field on the Product Lists Items record reflects selected item options by the user. It is not intended to be directly edited. For example, for gift certificates, the recipient email and address are stored in this field.

# Configuring Product Lists

Configuration of Product List parameters is controlled by the backend.configuration.js file in the My Account and Reference ShopFlow applications at ssp_libraries > backend.configuration.js. You can add default Product List templates and define the control a user can have over lists. You can also set the item display options.

**Important:** For SuiteCommerce Advanced sites, when modifying the default settings, you must modify the backend.configuraiton.js file in **BOTH** the ShopFlow and My Account reference applications. Modifications in the ShopFlow application affect what is available to the user when in the ShopFlow site whereas modifications in the My Account application affect what is available in the customer center.

## Configuring Predefined Templates

To add predefined templates, duplicate the list_templates function and rename the list.

```
   // product list configuration.
,   product_lists: {

      // can the user modify product lists ?
      additionEnabled: true
   ,   loginRequired: true
   ,   list_templates: [
      {
         'templateid': '1'
      ,   'name': 'My List'
      ,   'description': 'sample predefined template'
      }

,   {
         'templateid': '1'
      ,   'name': 'New List'
      ,   'description': 'new sample predefined template'
      }
   ]
```

**Note:** Only the default predefined list, My List, is automatically translated. When adding additional predefined lists, ensure that you edit the translation files as needed. Private lists created by users are not translated. For detailed information on translation see Handling Localization in Reference Implementations.

## Configuring Item Display Options

When a user is viewing Product Lists in their My Account page, items can be viewed in various formats similar to viewing items in the product display pages. For example, users can view items in a condensed

list without images or in a list layout with images. By default, the condensed and list views are enabled. These item display options can be configured in the backend.configuration.js file. For example, you can also enable table and grid views.

> **Note:** Item display options use the current width of the screen device, not the width of the browser, when the user loads the page. To reset the display, the user must resize the window and refresh the browser, regardless of device.

```
   ,   itemsDisplayOptions: [
         {id: 'list', name: 'List', macro: 'productListDisplayFull', columns: 1, icon: 'icon-th
-list', isDefault: true}
      // ,   {id: 'table', name: 'Table', macro: 'itemCellTable', columns: 2, icon: 'icon-th-la
rge'}
      // ,   {id: 'grid', name: 'Grid', macro: 'itemCellGrid', columns: 4, icon: 'icon-th'}
      ,   {id: 'condensed', name: 'Condensed', macro: 'productListDisplayCondensed', columns: 1
, icon: 'icon-th-condensed'}
      ]
```

# Product List Experiences & Wireframes

This section includes standard Product List experiences available for users depending on configuration options set. For detailed information on corresponding macros for page areas refer to Product Lists in the template reference section.

- Creating and Adding to Lists
- List Options
- List Options

## Creating and Adding to Lists

Users can add product lists on the fly while adding items or can create a list from the Product Lists page and then add items to that list. A user cannot add new lists when the site is configured to have a single predefined list.

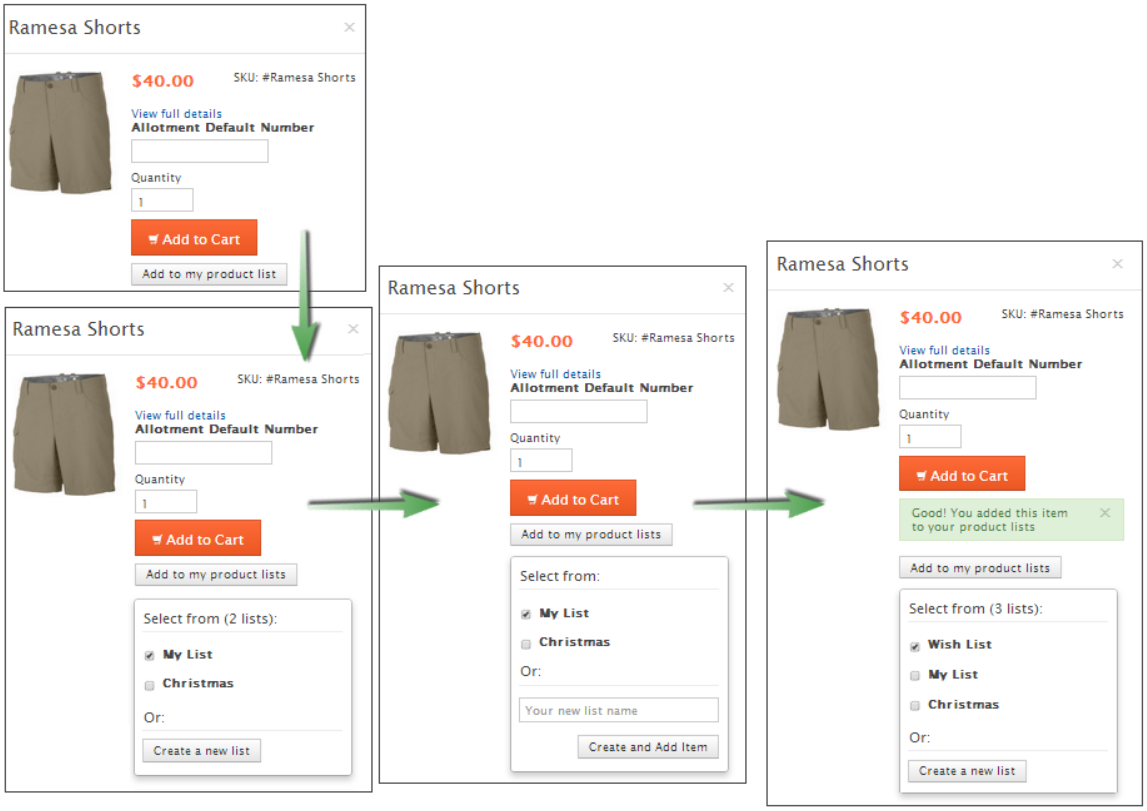
For more information on configuring a site for a single predefined list, see Configuring Product Lists.

This sections describes the following user experiences:

- Creating a New List
- Single Predefined List Available
- Adding Items When No Lists Exist
- Add to Product List from QuickView

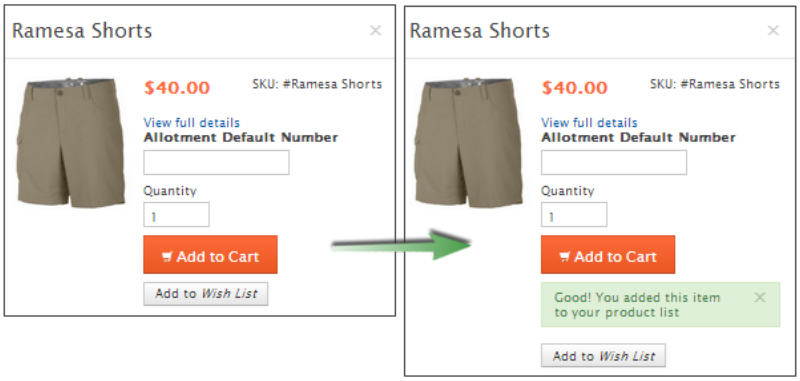
### Creating a New List

When adding items, the user can select from any predefined lists, private lists previously created, or they can create and add to a new list on the fly.



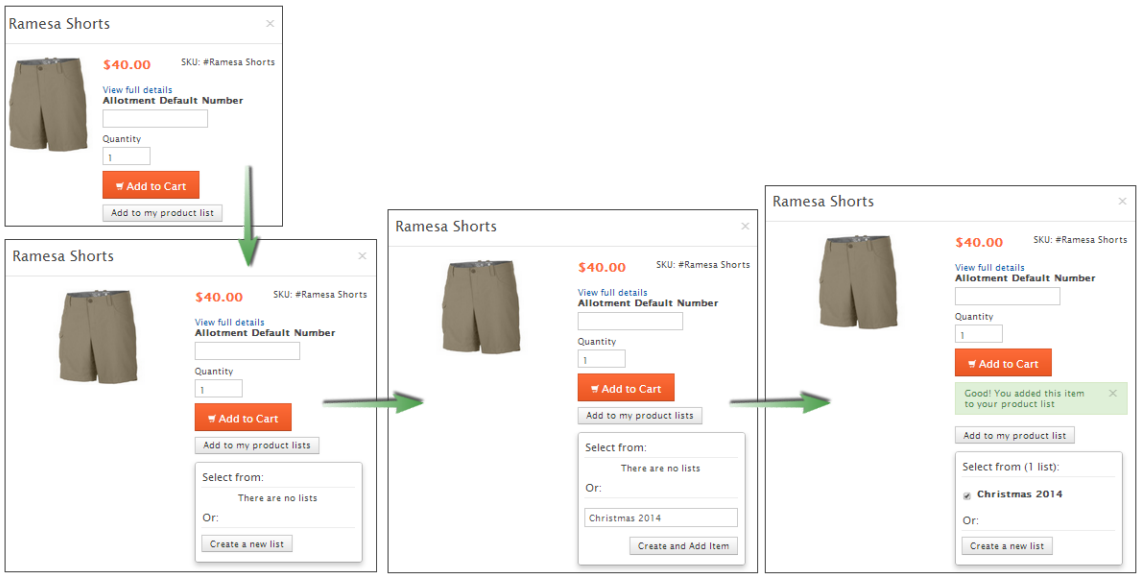

## Single Predefined List Available

When a single predefined list is available, the user can only add to that list. In this example only a single predefined list, Wish List, is available.



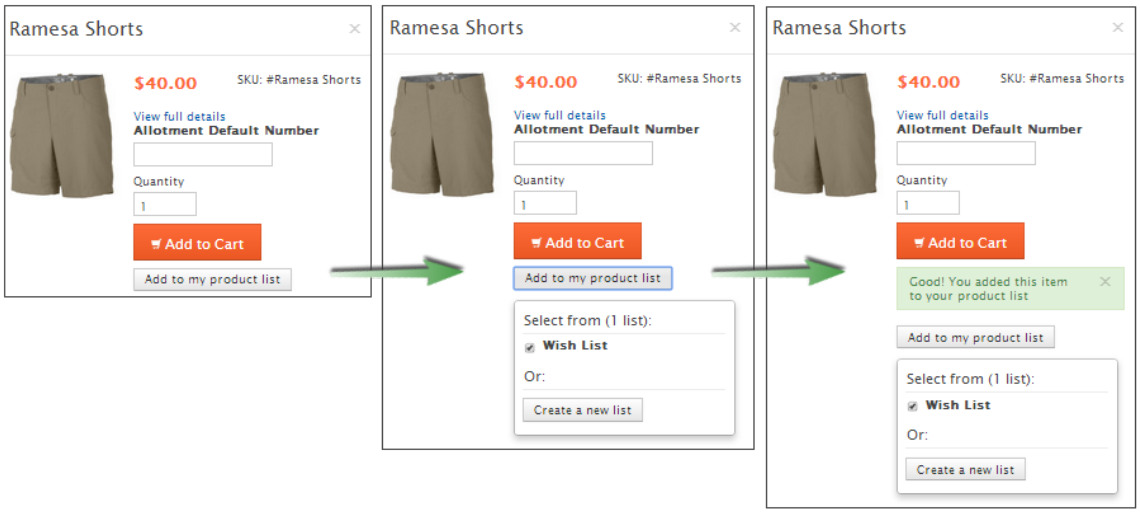

## Adding Items When No Lists Exist

When no predefined lists are configured and a user has not created a list, the user can immediately create a new list when adding an item.




## Add to Product List from QuickView

Users can add Items to their Product Lists from the QuickView modal dialog in the same manner as when adding them from the Item details page.




# List Options

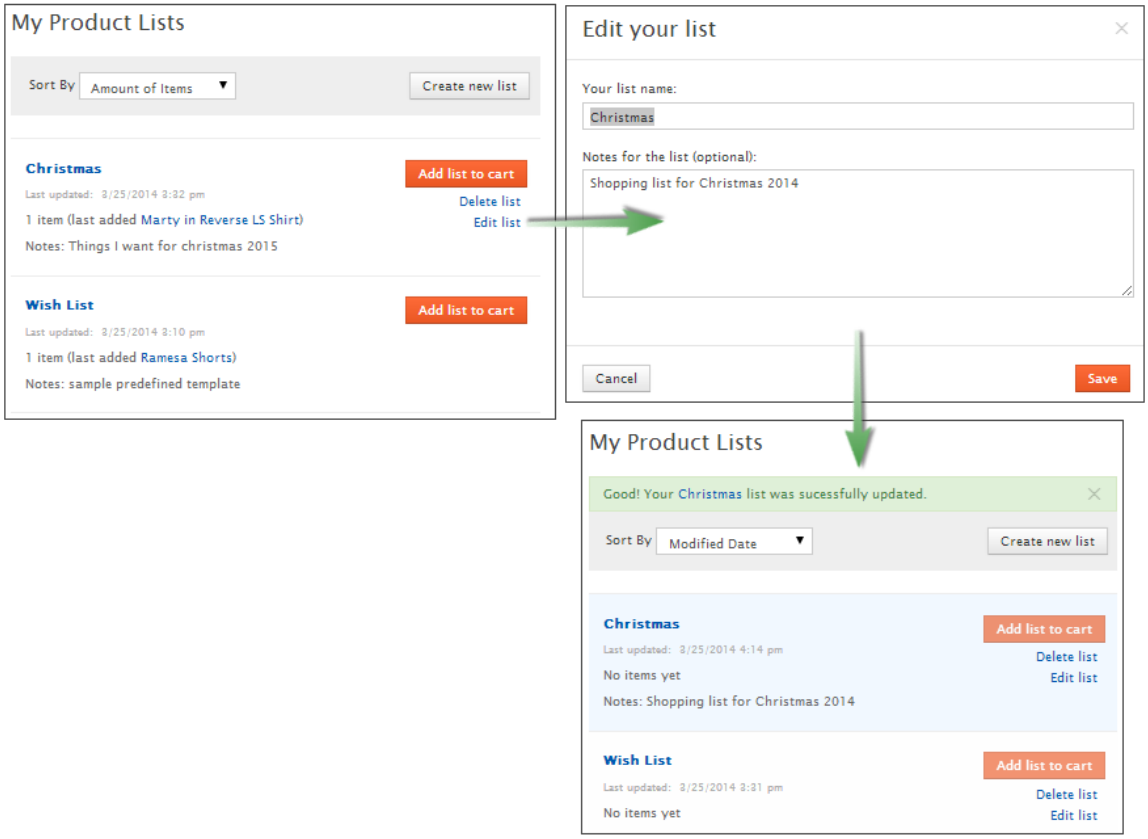
From the My Account Product Lists page, users can Edit, Add, or Delete Product Lists.

This sections describes the following user experiences:

- Editing a List
- Adding a List to the Cart
- Deleting a List

## Editing a List

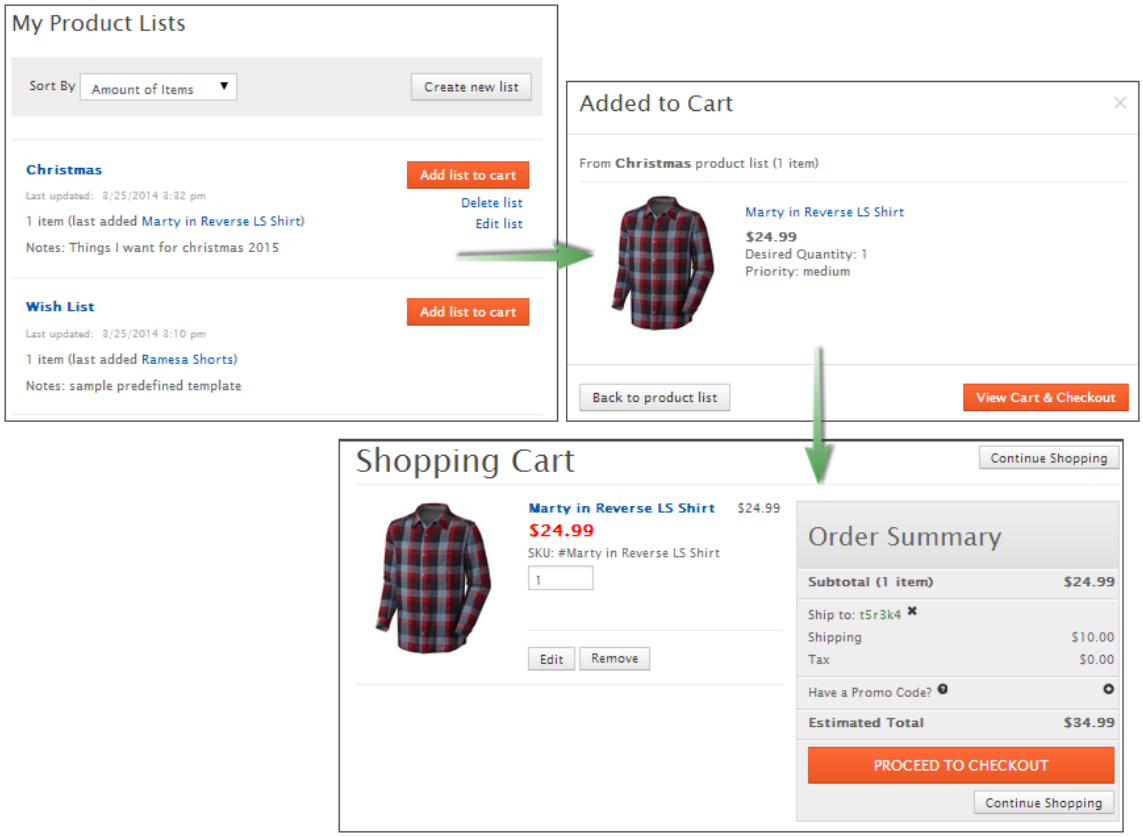
Users can edit list attributes for private lists by clicking Edit List and opening the Edit your list attributes in a modal dialog.. Product lists from predefined templates can not be edited.




## Adding a List to the Cart

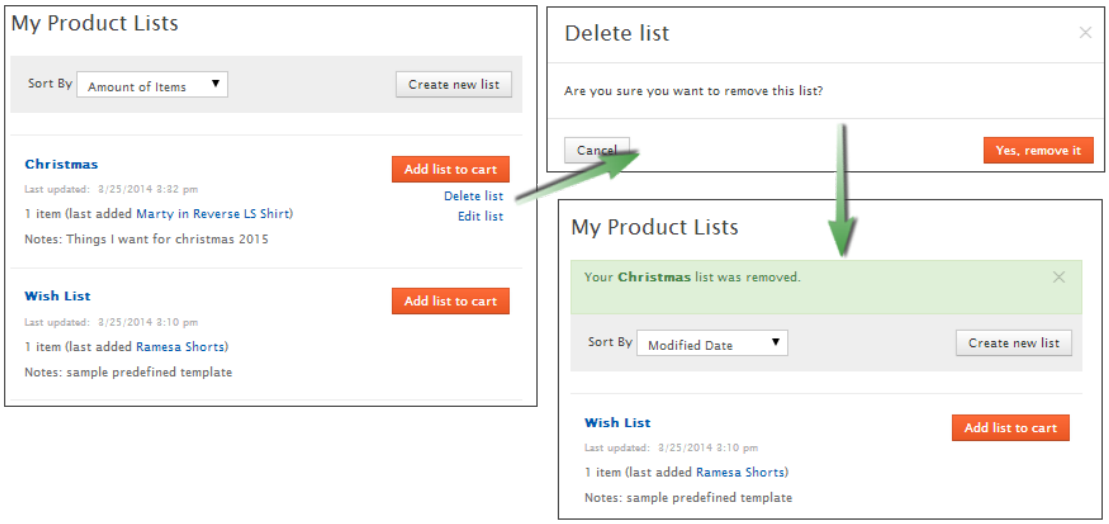
From the My Product List page, a user can add an entire list to the cart by clicking the **Add list to cart** button. The Added to Cart confirmation is displayed in a modal dialog. Closing the modal or clicking Back to product List returns the user to the My Product Lists page.

**Note:** When an item is not available for purchase, that Item is not added to the cart.




## Deleting a List

Users can delete a private product list by clicking Delete List. The confirmation is displayed in a modal dialog. Product lists from predefined templates can not be deleted.



# Item Options

From the My Account Product Lists page, users can manage items within those lists.

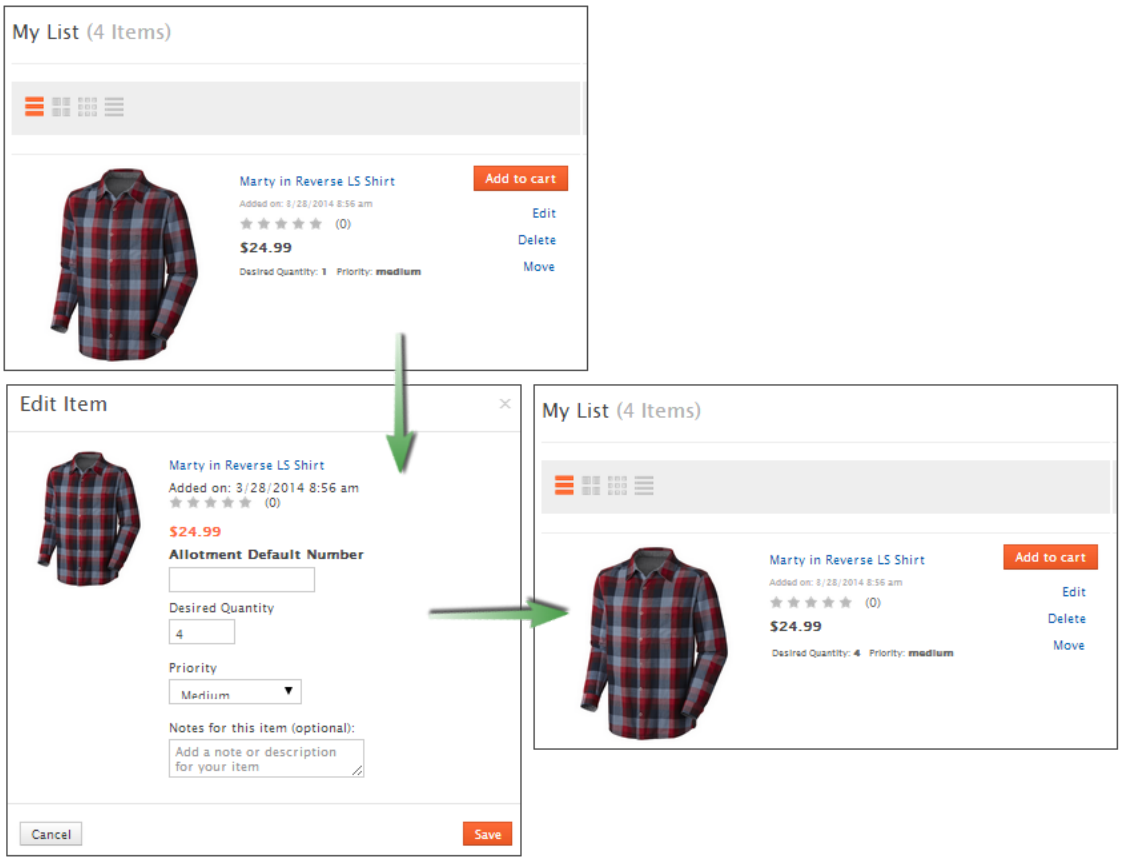
This sections describes the following user experiences:

- Editing Items in a List
- Moving Items to Another List
- Deleting Items from the List

## Editing Items in a List

Users can edit available item options by clicking Edit. All configurable item options for the current item are displayed in a modal dialog. Once saved, the user is returned to the Product List page where the updated item details are displayed. In the example below, the quantity is updated for the item.

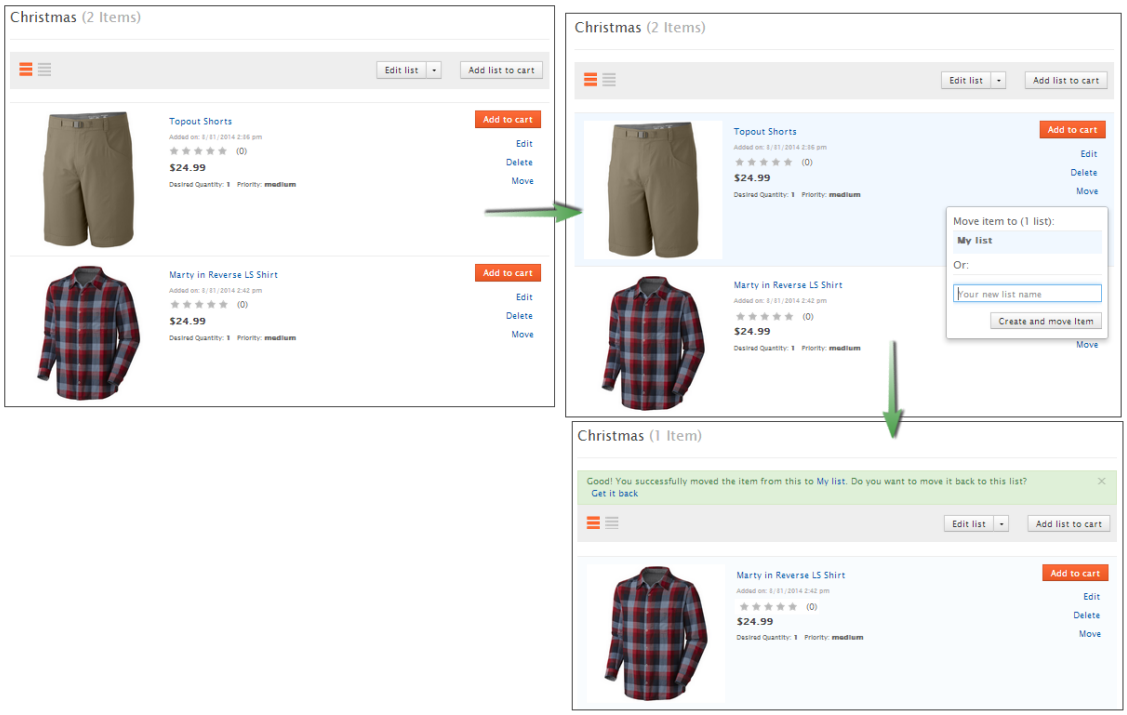
> **Note:** If you are unable to save changes when editing matrix items options, there may be an issue with your site setup. Ensure that the matrixchilditems_detail field is defined for the order fieldset as described in Minimum Setup Requirements.




## Moving Items to Another List

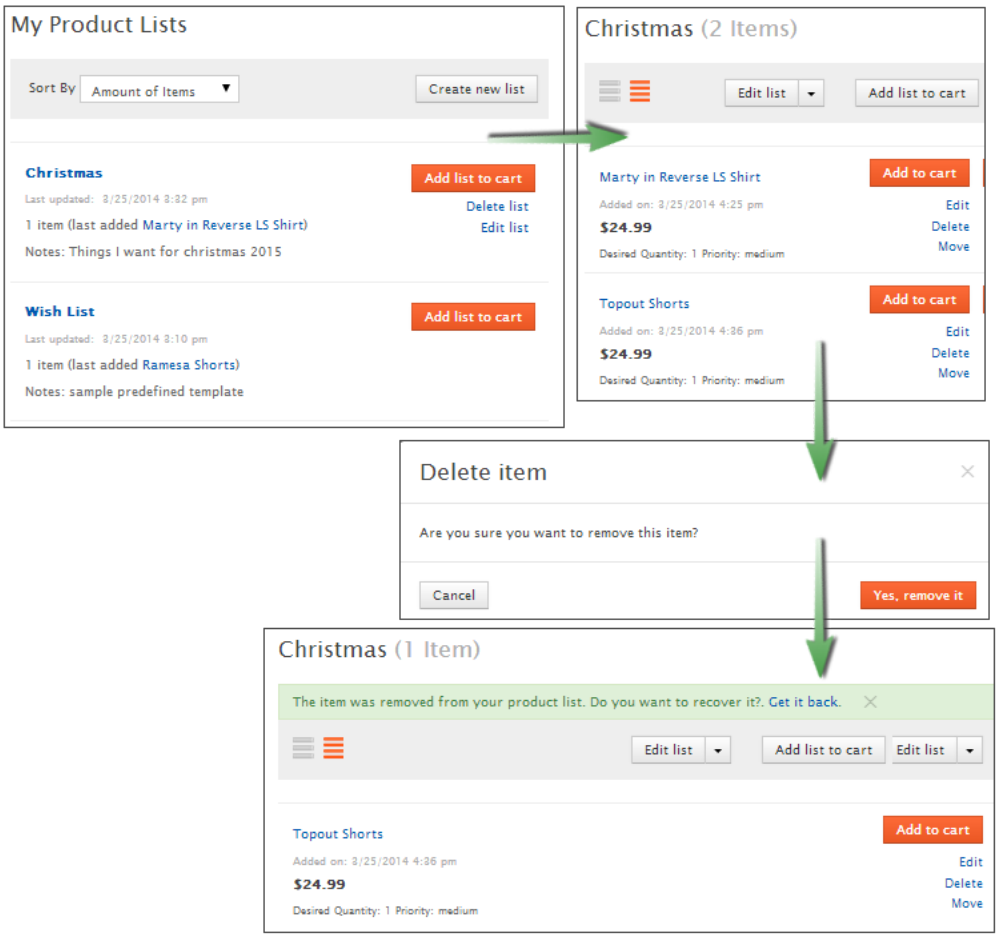
Users can move items from one list to another. They an move can item to another existing list or create a new list on the fly and then move the item.

**Note:** An item can not be moved to a list it has already been added to. If an item already exists in a second list, the second list does not display as an option in the move dialog box.




## Deleting Items from the List

Users can delete items from a product list by clicking Delete next to the desired item. The confirmation is displayed in a modal dialog. Users can also immediately retrieve a deleted item by clicking the Get it back link. This message is displayed temporarily after confirming the deletion.




# Billing

With the Billing functionality, invoicing and payments are supported through the web store. Users can view account balance and details, all open or closed invoices with due dates, and a full history of transactions such as Payments, Deposits, and Credits. They can then proceed to make payments against individual or multiple invoices for full or partial amounts. Users can also print a PDF of Invoices, Statements, Deposits, and Credit Memos.

**Note:** Billing feature are available for Reference My Account Premium editions only.

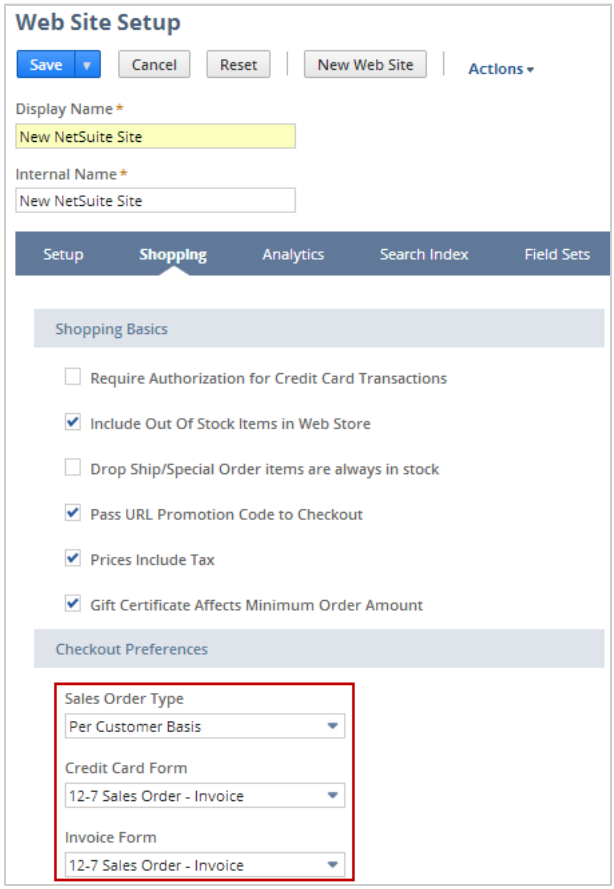
## Prerequisites

The Invoices page is designed for users who have payment terms set up. For detailed information on setting up payment terms, see the help topics Creating Terms of Payment and Using Terms of Payment

In order to view transaction history, you must also configure your web site Checkout Preferences on the Web Site Setup record at Setup > SuiteCommerce Advanced > Set Up Web Site as follows:

- Set the Sales Order Type to a Per Customer Basis

- Set the Credit Card Form and Invoice Form to an invoice form




# Billing Experiences & Wireframes

This section describes the following user experiences for the Billing pages available in a SuiteCommerce Advanced or Site Builder site. For detailed information on understanding Billing and Invoices within NetSuite, see the help topic Billing and Invoices.

- Account Balance Page
- Make a Payment
- Print a Statement
- Transaction History

## Account Balance Page

On the Account Balance page, users can see the account balance and related information. Details include:

- Account Balance information:
  - Current balance and available credit
  - Buyer's company name

> **Note:** If the company name is not entered on the Entity record, the area is left blank.

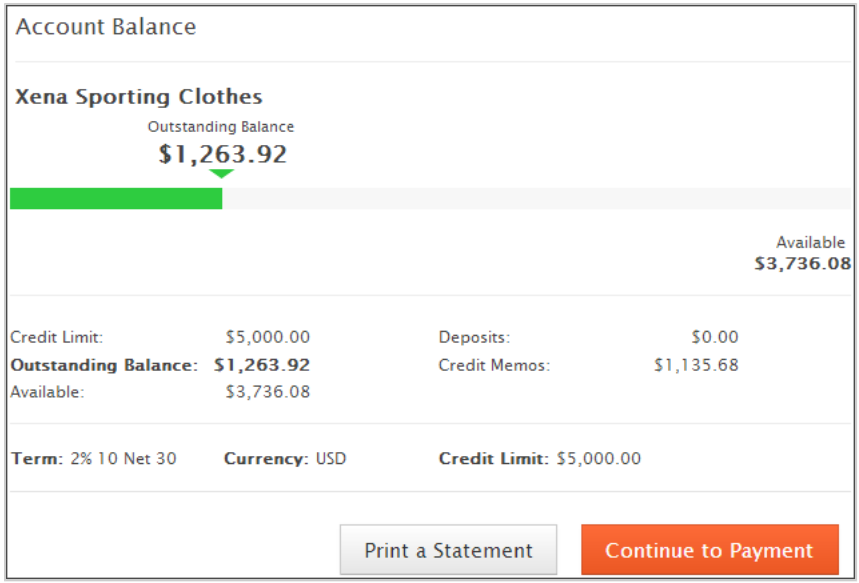
- Summary of credits: deposits, credit memos
- Contract related information: Customer term, Credit limit, Currency

> **Note:** The Customer term must be defined on the Entity record to display here.

- Make a Payment

  The payment flow is started by clicking on the **Continue to Payment** button.

> **Note:** When there are no outstanding invoices, the Make a Payment button is disabled with a **No payment due** label.




## Make a Payment

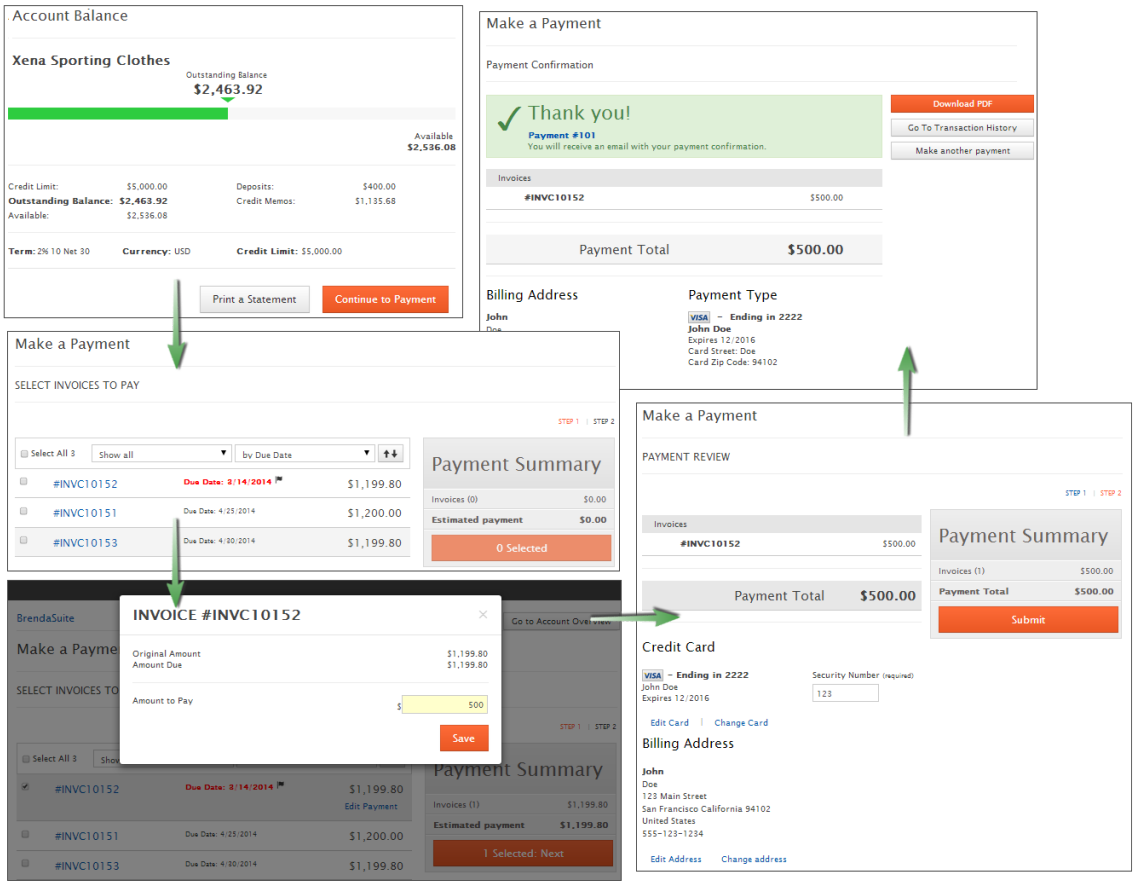
Users can begin a Payment process against invoices from a couple different locations within the My Account pages. They can choose to pay against single or multiple invoices and pay partial amounts on any given invoice.

This sections describes the following user experiences:

- Payment from Account Balance Page
- Payment From the Invoice Page

### Payment from Account Balance Page

From the Account Balance page, users can select Continue to Payment to go directly to the Make a Payment page. There users can view all Open or Paid in Full invoices. They can filter invoices by due date and sort by Due Date, Invoice Date, Invoice Number or Amount Due. Once an invoice is selected, users can edit the total amount to pay in a modal dialog. In the payment review page, users can edit address and billing information in modal dialogs. Once the payment is completed, users can download a PDF statement.

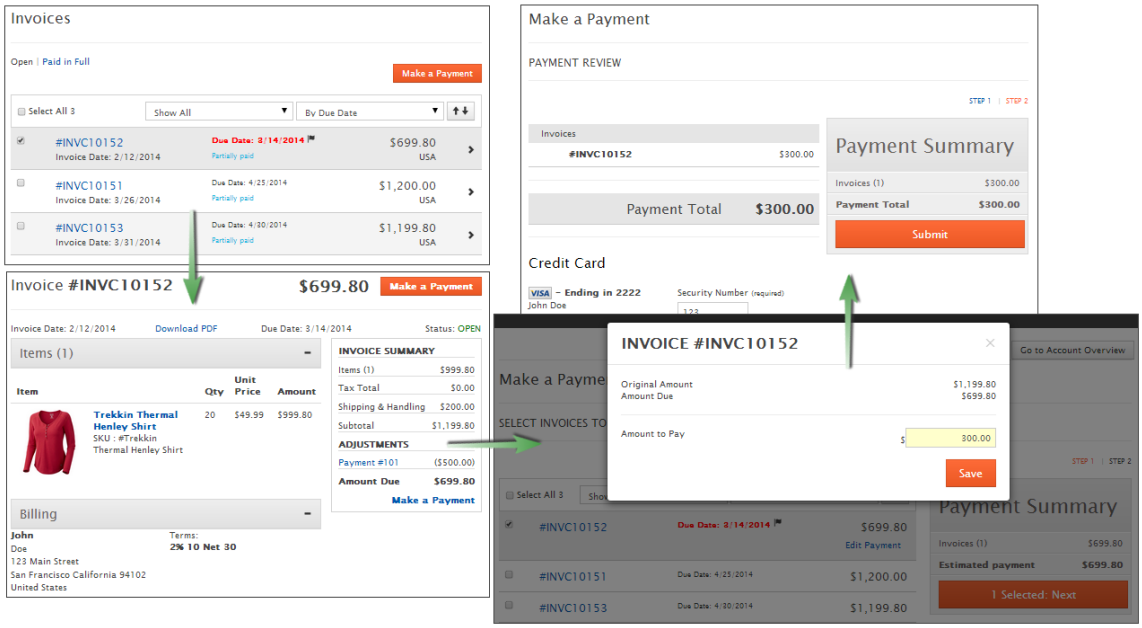



## Payment From the Invoice Page

From the Invoices page, users can view all Open or Paid in Full invoices. They can filter invoices by due date and sort by Due Date, Invoice Date, Invoice Number or Amount Due.

Users can then drill into a single invoice and enter a payment for that invoice. When a user has Terms with discounts, the discount does not display since it is only available once the Payment Flow starts. In the Payment Flow, users see the discount on a label in the invoice row and the detailed amounts in the **Edit amount** modal.

**Note:** If the user selects the Make a Payment button instead of drilling down into a single invoice, they can make payments for multiple open invoices as selected. This is a similar experience as described above in Payment from Account Balance Page.




## Payment Using a Credit Memo or Deposit

From Reference My Account, users can pay invoices using the following payment methods:

- Existing credit memos
- Deposits

**Note:** Reference My Account also enables the user to enter credit card and billing information. This is the same as Reference Checkout.

These payment options are based on the Accept Customer Payments feature in NetSuite. The workflow for paying invoices is:

1. Logs in to Reference My Account.
2. Selects Invoices.
3. Select the invoice to be paid.
4. Select Credits or Deposits

   After selecting an invoice, Reference My Account displays a list of credits and deposits that are available to the user. On the right side of the Invoices page, the user can view summary information which includes the amount due for all of the select invoices.
5. Click Make a Payment.

   From this page, the user can select other invoices to pay as necessary. They can also edit the payment amount.
6. Click Next.

   This page displays the list of invoices being paid and the total amount due.
7. Click Submit Payment.

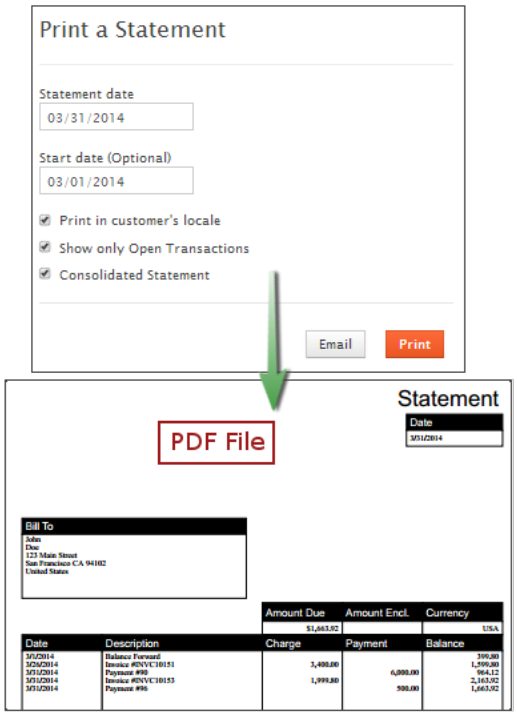
## Print a Statement

From the Print a Statement page, users can print account statements.

Statements printed show account status and balances that summarize the following:

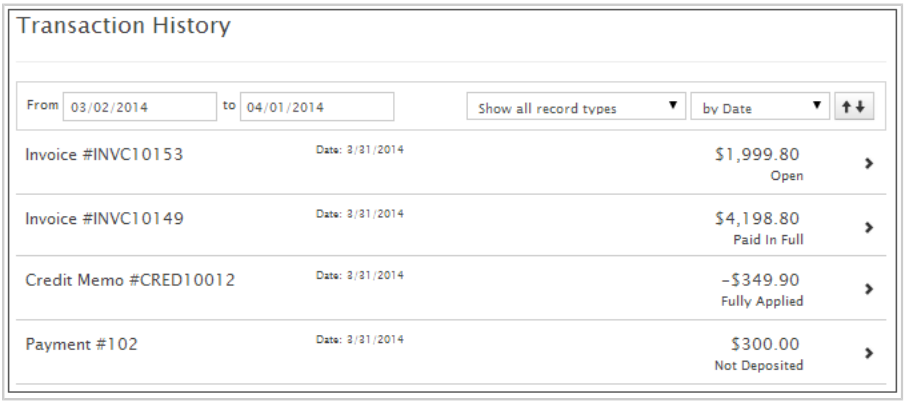
- Invoices
- Finance charges
- Credit Memos
- Payments received

Users can specify date ranges, choose to print open and closed transactions, and choose to print consolidated balance information for customer-subcustomer hierarchies into a single output.



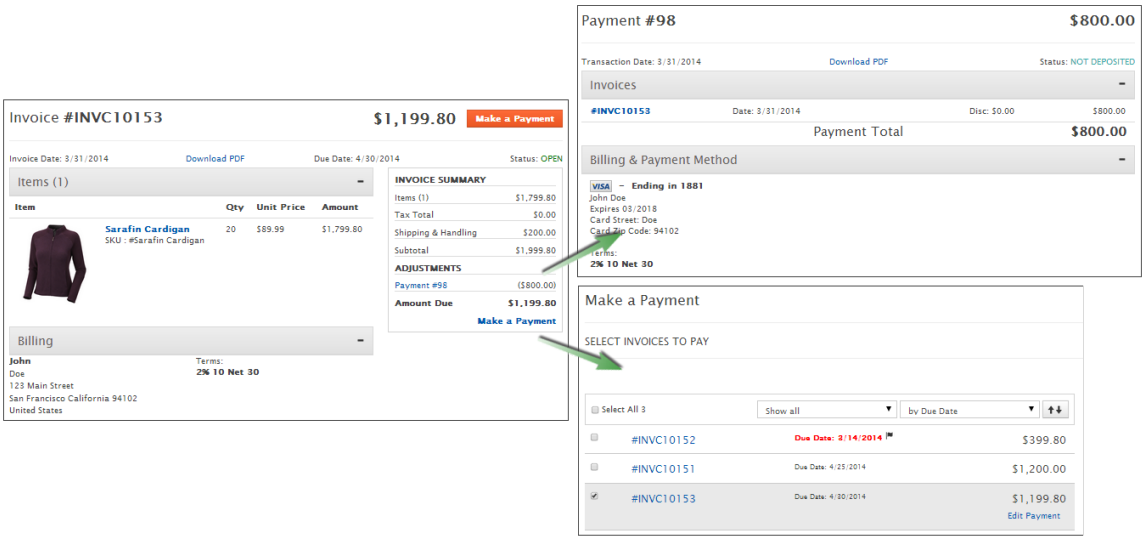

## Transaction History

In the Transaction History pages of My Account, users can filter records by type and date and then sort the results. Users can then drill down into individual records.



## Viewing Open Invoices from the Transaction History

Users can view remaining invoice balances from Open transactions lines. The user can then view prior payments or make another payment for that invoice.




## Viewing Paid Invoices from Transaction History

Users can view paid invoices and then drill down to payments made for that invoice. They can download a PDF of the invoice or payment.

Invoice #INVC10149 $4,198.80

Invoice Date: 3/31/2014 Download PDF Due Date: 4/30/2014 Status: PAID IN FULL

Items (1)

| Item | Qty | Unit Price | Amount |
|---|---|---|---|
| Wicked Lite Shirt<br>SKU : #Wicked Lite Shirt | 120 | $24.99 | $2,998.80 |

| INVOICE SUMMARY | |
|---|---|
| Items (1) | $2,998.80 |
| Tax Total | $0.00 |
| Shipping & Handling | $1,200.00 |
| Subtotal | $4,198.80 |
| ADJUSTMENTS | |
| Payment #90 | ($4,114.82) |
| Payment #96 | ($83.98) |
| Amount Due | $0.00 |

Billing

John
Doe
123 Main Street
San Francisco California 94102
United States

Terms:
2% 10 Net 30

Payment #90 $6,000.00

Transaction Date: 3/31/2014 Download PDF Status: NOT DEPOSITED

Invoices

| | | | |
|---|---|---|---|
| #INVC10147 | Date: 3/31/2014 | Disc: $0.00 | $1,249.50 |
| #INVC10149 | Date: 3/31/2014 | Disc: $0.00 | $4,114.82 |
| | Payment Total | | $6,000.00 |

Billing & Payment Method

VISA – Ending in 2222
John Doe
Expires 12/2016
Card Street: Doe
Card Zip Code: 94102

Terms:
2% 10 Net 30

## Viewing Credit Memos from Transaction History

Users can view Credit Memos and then drill down to related invoices or item records. They can also download a PDF for the current Credit Memo.

Credit Memo #CRED10012 $349.90

Date: 3/31/2014 Download PDF Status: FULLY APPLIED

Returned Items (1)

| Item | Qty | Unit Price | Amount |
|---|---|---|---|
| Marty in Reverse LS Shirt<br>Marty in Reverse LS Shirt | 10 | $24.99 | $249.90 |
| | | Items Subtotal | $249.90 |

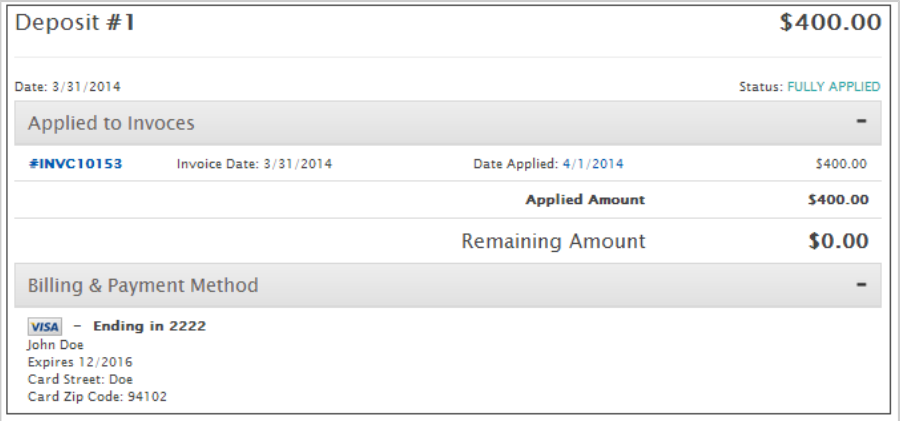
| ITEMS SUMMARY | |
|---|---|
| Items Subtotal | $249.90 |
| Discount | $0.00 |
| Tax Item | $0.00 |
| Shipping Cost | $100.00 |
| Total | $349.90 |

Applied to Invoices

| | | |
|---|---|---|
| #INVC10150 | Transaction Date: 2/20/2014 | $349.90 |
| | Applied Subtotal | $349.90 |
| | Remaining Amount | $0.00 |

### Viewing Deposits from Transaction History

Users can view Deposits and then drill down to related invoices.




# Returns

With Returns, users can initiate and track a Returns request from an existing Purchase Order or Invoice. The Return request is automatically linked to a purchase order or invoice. On the Returns views, users can do the following:

- Submit a Return from an Order or Invoice
- Monitor Returns Process
- Cancel Returns
- Receive Email Notification

For more details, see the following topics:

- User Experience
- Setup
- Upgrading from Previous Version
- Customizing Returns

## User Experience

This section describes the user experience for Returns pages available in a SuiteCommerce Advanced or Site Builder site. For detailed information on understanding Returns within NetSuite, see the help topic Customer Return Management.

Refer to the *SuiteCommerce Advanced User Experience Design Document* for a complete overview of the user experience.

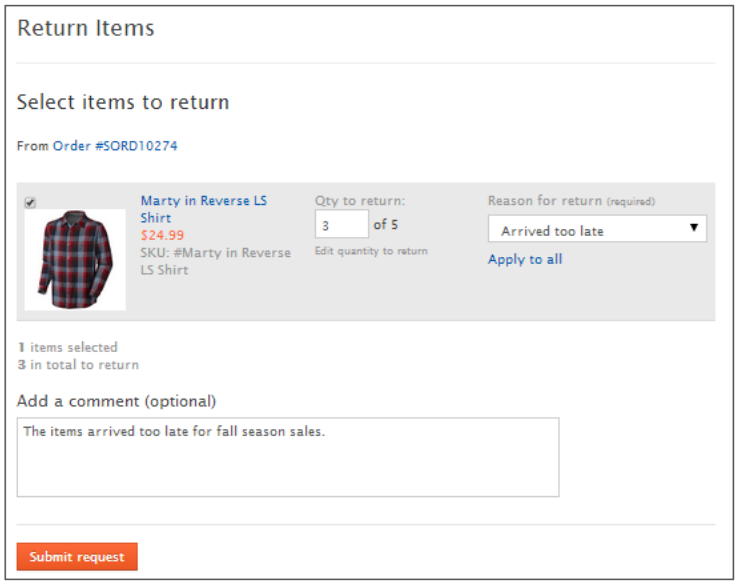
With the Returns feature, users can initiate a return against an existing invoice or order, view a list of return items, and view details of a specific returns request. To perform these tasks, the Returns feature uses three main views:

- Request a Return
- List Return Requests
- Return Request Details

## Request a Return

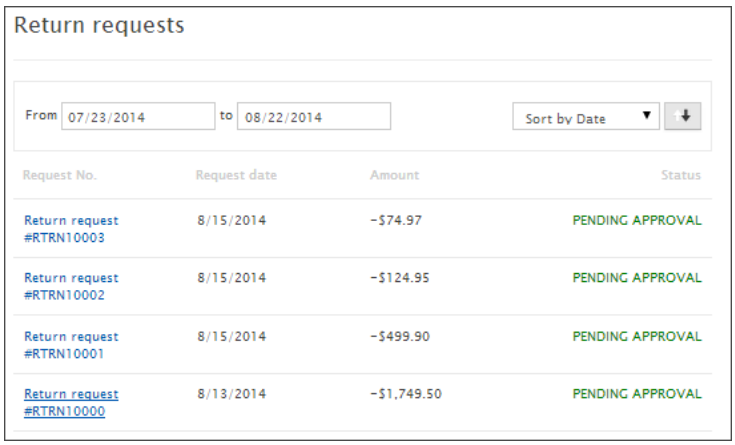
Users access the Request a Return page from an existing invoice or order. Here, users can select the items to return from the item list. Once selected, the quantity and reason for the return can be edited. Users can also add comments.

All items not eligible for return are shown in a secondary accordion that is closed by default on desktop and mobile.




## List Return Requests

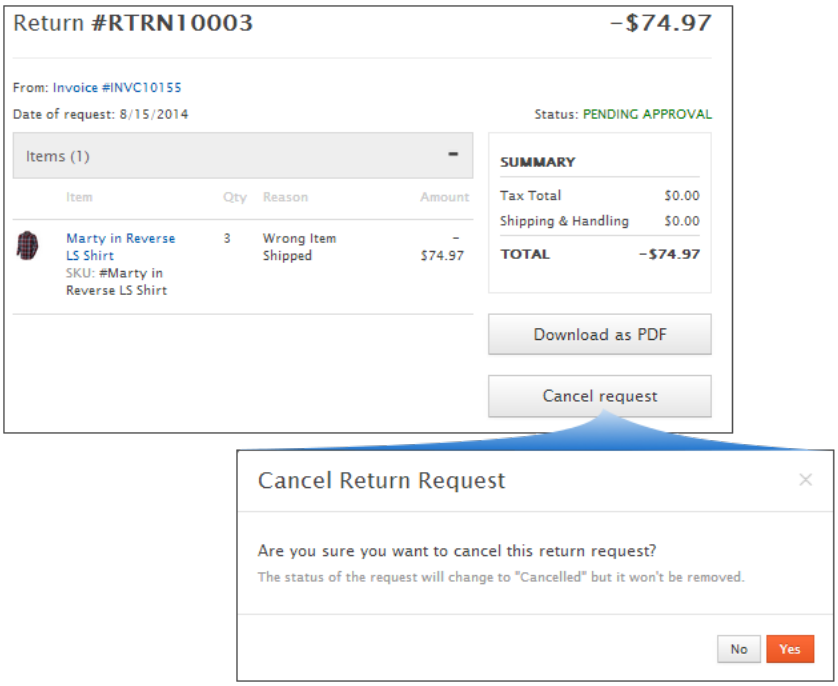
Users can view a list of all Return Requests. They can filter by a range of dates or sort return requests by data or number.




## Return Request Details

From the Request Details view, users can see details of a specific return request. Users can download a PDF of the details or cancel a return request.

**Note:** Only return requests that have not been approved can be cancelled.




# Setup

- Prerequisites
- Feature Architecture
- Configuration Options
- Known Limitations

## Prerequisites

Returns is available in Reference My Account 1.05 and Reference My Account Premium 1.04 and above.

In order for users to access Return Authorizations, they must be logged in and have permissions for Return Authorizations records.

## Feature Architecture

The Return Authorization module uses the standard architecture shared across many Reference My Account modules with some modifications as described in this section.

### Modules

The primary module files used to define the behavior of Returns include the following files at js/src/app/modules/ReturnAuthorization:

- **ReturnAuthorization.js**: also referred to as the index file. This file references all of the sub-modules and files used for Return Authorizations. This file also contains the code to define the display and position of the Return item within the Reference My Account menu (see Modifying the Menu).
- **ReturnAuthorization.Model.js**: this is a standard Backbone module that listens to ensure that its lines are an instance of **OrderLine.Collection**.
- **ReturnAuthorization.Collection.js**: defines the required list header methods, `parse` and `update`. The `update` method is called each time a header filter is changed to re-sync data from the backend.
- **ReturnAuthorization.Router.js**: instantiates the various views depending on the given URL.
- **ReturnAuthorization.Views.js**: this is the index file that references all views. The following views are referenced:
  - ReturnAuthorization.Views.Cancel.js
  - ReturnAuthorization.Views.Confirmation.js
  - ReturnAuthorization.Views.Detail.js
  - ReturnAuthorization.Views.Form.js
  - ReturnAuthorization.Views.List.js

## Services

Return Authorizations uses the **return-authorization.ss** service to control backend calls and logic.

## ReturnAuthorization Model

The `ReturnAuthorization` model defined within the Models.js file is unique from most other Reference My Account backbone models. Rather than using an `nlapiLoadRecord` in the `get` function which returns all data for a given record, the `nlapiSearch` functions are used to retrieve data for a specified set of data. This improves performance. The set of data to be returned is defined in a columns array. The results returned from the search are then mapped as needed.

Within NetSuite a Return Authorization record consists of a single record internal ID that corresponds to the main record information and all order line items for that record. Each order line item does not have a separate internal ID. Because of this, there may be multiple results corresponding to a single internal ID. When a search for a specific Return Authorization record is done, one result for the mainline data is returned as well as a separate result for each order line item. When discount items exist, these are also returned as a separate result.

In the code, each result is parsed to return mainline data such as the internal id, data, comment etc. Then the order line item data is parsed to return item data.

```
return {
   internalid: main_return_authorization.getId()
,   type: main_return_authorization.getRecordType()

,   date: main_return_authorization.getValue('trandate')
,   tranid: main_return_authorization.getValue('tranid')
,   comment: main_return_authorization.getValue('memo')

,   status: {
      id: main_return_authorization.getValue('status')
   ,   label: main_return_authorization.getText('status')
   }

,   isCancelable: this.isCancelable(main_return_authorization)
```

```
    ,   createdfrom: this.getCreatedFrom(return_authorizations)
    ,   summary: this.getSummary(main_return_authorization)
    ,   lines: this.getLines(return_authorizations)
    };
}
```

Important methods used to process Return Authorization data include:

- `getCreatedFrom`: loops through the results to filter for a record that has a value for the createdfrom column. The `return` object is then built from the data set for that record. Since only the mainline record has a createdfrom value, this object contains the data for the mainline record.
- `getLines`: loops through the results to find a record that has a value for the item column. The `return` object is then built from the data set for that record. Since only order line item records have item data, this object contains data for a given order line item.
- `getSummary`: formats floating point numbers returned from the search to the appropriate currency output.
- `update`: uses the `nlapiRequestURL` API to directly access the current Return Authorization record and set the status while passing the request header for the currently logged in user.

  **Note:** The URL accessed uses a **system.netsuite.com** domain by default as defined in the backend.configuration.js file. Some accounts do not use this domain. For example, the account may be using system.na1.netsuite.com. In these cases the backend.configuration.js file must be customized to reflect the correct domain.

- `create`: creates a Return Authorization request record from an existing Invoice or Order and populates that record with all line items from the referenced Invoice or Order.

  **Note:** The memo field on the Return Authorization record is populated with the text input by the user in the memo field on the front-end interface.

- `setLines`: removes or resets item/quantity details on lines in the Returned Authorization record created by the `create` method.
- `list`: used to return the list of records returned from the `nlapiSearch` API call. This method includes calls to several List Header filters.

  **Note:** In order to maintain backwards compatibility with a previously released Placed Order module, there are two separate list methods. The first, `getPaginatedSearchResults`, is the standard List Header pagination method. The second, `getAllSearchResults`, returns all of the search results for applied filters and columns. The second method is used by the Placed Order module.

```
if (data.page)
{
   filters.push(new nlobjSearchFilter('mainline', null, 'is', 'T'));

   return_authorizations = Application.getPaginatedSearchResults({
...
...
else
{
   return_authorizations = this.parseResults(Application.getAllSearchResults('returnauthorizati
on', filters, columns));
}
```

## Configuration Options

In the Configuration.js file, the default values corresponding to the Reason for Return field are defined. This field is accessed by the user in the Return Authorization request form on the web site.

```
   // Return Authorization configuration
,   returnAuthorization: {

      reasons: [
         _('Wrong Item Shipped').translate()
      ,   _('Did not fit').translate()
      ,   _('Quality did not meet my standards').translate()
      ,   _('Not as pictured on the Website').translate()
      ,   _('Damaged during shipping').translate()
      ,   _('Changed my mind').translate()
      ,   _('Item was defective').translate()
      ,   _('Arrived too late').translate()
      ]
   }
```

## Known Limitations

The following are known limitations for Returns:

- Return Authorizations can only be created from **existing** Invoices or Orders.
- Shipping Slips are not supported.
- Return Status per line is not supported.

# Upgrading from Previous Version

Returns is a new feature as of Reference My Account 1.04. No special upgrade requirements specific to this module are needed.

# Customizing Returns

> **Note:** When customizing reference implementations, the files mentioned in this section can NOT be edited directly from the reference application. Copy the file to the same location in the Custom SSP application folder and edit the file there. For details, see Using the Custom Folder.

## Modifying the Menu

By default, the Returns menu item in Reference My Account is nested within the Orders menu. You can customize the Returns menu item by editing the ReturnAuthorization.js file at js/src/app/modules/ReturnAuthorization.

The following code snippet shows the parameters set by the `MenuItems` object:

```
,   MenuItems: {
      parent: 'orders'
```

```
  ,   id: 'returns'
  ,   name: _('Returns').translate()
  ,   url: 'returns'
  ,   index: 2
  ,   permission: 'transactions.tranFind.1,transactions.tranRtnAuth.1'
  }
```

- `parent`: sets the nested position of the Returns item. To display Returns at the top level, comment this line out. You can set this to the list item data-id of any existing menu item. For example, to nest the Returns menu item within the Billing menu, set `parent` to `billing`. Note that you can determine the data-id for any list item within the menu by inspecting the element html code within your browser.
- `id`: sets the id for the current list item. This should be kept as `returns` for this module.
- `name`: sets the text to be displayed on the web site. If you customize this value and your site uses multiple languages, edit the language files with the corresponding translation. For more information, see Handling Localization in Reference Implementations.
- `url`: sets the default URL path for the page.
- `index`: sets the level within the navigation menu tree. When customizing this value, you should compare against the level set for other modules appended to the same menu tree to ensure that each module within a group has a unique index value.
- `permission`: sets the permissions required by the logged in user to view this menu item. By default, this permission parameter is set to the permissions required by the standard NetSuite Return Authorization record. This may need to be edited when a custom record with custom permissions is used. For more information on permissions, see Working with Permissions.

## Modifying Data Returned

As described in the section ReturnAuthorization Model, the `nlapiSearch` API is used to return a specified set of data from the Return Authorization record. To retrieve more or less data from the record, you can modify the columns array in Models.js to include or exclude fields from the record as needed.

```
,   columns = [
      // Basic info
      new nlobjSearchColumn('mainline')
  ,    new nlobjSearchColumn('trandate')
  ,    new nlobjSearchColumn('status')
  ,    new nlobjSearchColumn('tranid')
  ,    new nlobjSearchColumn('memo')

      // Summary
  ,    new nlobjSearchColumn('total')
  ,    new nlobjSearchColumn('taxtotal')
  ,    new nlobjSearchColumn('currency')
  ,    new nlobjSearchColumn('shippingamount')

      // Created from
  ,    new nlobjSearchColumn('internalid', 'createdfrom')
  ,    new nlobjSearchColumn('tranid', 'createdfrom')
  ,    new nlobjSearchColumn('type', 'createdfrom')

      // Items
```

```
,   new nlobjSearchColumn('internalid', 'item')
,   new nlobjSearchColumn('type', 'item')
,   new nlobjSearchColumn('quantity')
,   new nlobjSearchColumn('options')
,   new nlobjSearchColumn(amount_field)
,   new nlobjSearchColumn('rate')
]
```

## Customizing the Domain for Cancelling Returning Authorizations

To cancel a Return Authorization, the `update` method in the `ReturnAuthorization` model uses a URL to access the record and set the status to cancelled. By default, the URL accessed uses a **system.netsuite.com** domain. Some accounts do not use this domain. For example, the account may be using system.na1.netsuite.com. In these cases the URL used must be customized to reflect the appropriate domain. To customize the domain, override the `cancelUrlRoot` setting in the `ReturnAuthorizations` object within the **backend.configuration.js** file.

```
,   returnAuthorizations: {
      // Cancel Return System Path
      cancelUrlRoot: 'https://system.netsuite.com'
      // cancelUrlRoot: 'https://system.f.netsuite.com'
      // cancelUrlRoot: 'https://system.na1.netsuite.com'
      // cancelUrlRoot: 'https://system.beta.netsuite.com'
      // cancelUrlRoot: 'https://system.sandbox.netsuite.com'
}
```

## Using a Custom Return Authorization Record

By default the standard returnauthorization record is used to create Return Authorizations. If you are using a custom Return Authorization record, you can leverage the existing `ReturnAuthorization` model to create a new model. You will need to edit the following line in the `ReturnAuthorization` model code, replacing returnauthorization with the internal id of the custom record you are using. You must also modify the columns array, and other logic referencing data returned based on the defined columns, to match the columns available in your custom record.

```
,   return_authorizations = Application.getAllSearchResults('returnauthorization', filters, col
umns)
```

# Case Management

Reference My Account enables users to submit questions or support queries directly from the My Account user interface. This is handled using Case Management. Case Management is a component of Support Management in NetSuite that enables tracking and managing of support cases. When a user reports an issue, question, or concern, you can creates a case in NetSuite. This case is assigned to the appropriate NetSuite user. After the case is assigned, the workflow begins and actions taken. See the help topic Working With Cases for more information.

For more information about using Case Management within Reference My Account, read the following topics:

- User Experience
- Setup
- Upgrading from a Previous Version
- Customizing Case Management


# User Experience

Refer to the *SuiteCommerce Advanced User Experience Design Document* for a complete overview of the user experience.

Customers can use the Case Management feature to perform the following from My Account:

- Submit a new case.
- View cases that have already been submitted.
- View the details of an existing case.
- Attach a message to a case.
- Close an open case.

To perform these tasks, Case Management implements three views:

- Create a Case
- Case List
- Case Detail

## Create a Case

This view is implemented by the CaseCreate.View.js file. It enables users to enter the following information:

- Subject
- Type of Inquiry
- Message
- Custom Email Address.

Possible values for **Type of Inquiry** are defined in NetSuite in Setup > Support > Case Types.

When a user submits the form, the case record is created in NetSuite using the information provided. When the case record is created, the case number is automatically generated.

The Case Management feature also sets the case status. You can configure the default value for the case status. See Configuration Options for more information.

## Case List

This view is implemented by the CaseList.View.js file. It enables users to view their case history.

It contains a list header which can:

- Filter by case status.
- Sort by case number, creation date, or last activity.

This list can be displayed across multiple pages. The page size is configured like any other list implementing pagination.

## Case Detail

This view is implemented by the CaseDetail.View.js file. It enables the user to view the following information about the case:

- Subject
- Message chain
- Creation date
- Last message date
- Status
- Case type

Using this view, a user can add a reply message or close a case. When a user adds a reply message, the Case Management feature adds the message to the case record. When closing a case, the case status is updated according to the status set in Configuration.js. See Configuration Options for more information.

# Setup

The Case Management feature is installed and configured by default when Reference My Account is installed. As with other Reference Implementation features, the Require.js file loads the Case Management module when the application starts.

- Prerequisites
- Feature Architecture
- Configuration Options

## Prerequisites

To use Case Management within Reference My Account, you must set up Case Management within NetSuite. See the help topic Setting Up Case Management.

## Feature Architecture

The Case Management feature is implemented across multiple files. The following sections describe each of these files.

### Main Module

Reference My Account contains one module for Case Management. This module is defined in the Case.js file. This file defines the menu items that appear in My Account and initializes all required modules. Case.js contains the following primary methods:

- isCaseManagementEnabled: checks to see if the Case Management module is enabled.
- mountToApp: appends all required modules to the application object. It also maintains a reference to these objects.

## Backbone Configuration

Case Management uses the following files to define and configure the Backbone model:

- CaseModel.js: defines the Backbone model for case instances. Backbone validation is also handled by this file.
- CaseCollection.js: defines the list header code that supports filtering, sorting, and server-side pagination. This file contains the following methods:
  - parse: processes the server-side search response containing the number of records per page, total records found, and data for each record.
  - update: retrieves a new record set based on list headers.
- CaseFields.Model.js: defines the initial values for a standard case. This file is invoked from the Create New Case and Case List views.

## Routers

Case Management uses one router defined in the Case.Router.js file. This file contains the following primary methods:

- showCase: requests the details of an individual case.
- showCasesList: requests a list of all cases.
- createNewCase: creates a new case.

## Views

Case Management implements the following views:

- CaseList.View.js: defines the view used to display a list of all open cases for the current user.

  This file contains the following important methods:
  - listenCollection: listens for changes to the case collection. This method is required by the list header module.
  - setupListHeader: initializes the list header and loads filters with all values retrieved from the back-end. This method is called from the initialize() method when creating the view.
  - setLoading: defines the List Header module if the record list is still being loaded.
  - initializeFilterOptions: defines the default filter options using values retrieved from the back-end.
  - sortOptions: defines an array containing the default sort options required by the list header.
  - showContent: renders the Case List view. This method is called from Case.Router.js.
  - render: redefines the Backbone model to support notifications when creating new cases after the view is loaded. When the user creates a new case, it is redirected to this view. This method is used to display a customized message to the user different than that provided by the standard Backbone model.
- CaseDetail.View.js: defines the view which displays details about a specific case to the user.

  This file contains the following important methods:
  - customSaveForm: redefines the generic saveForm method. This is used to provide a customized message to the user.
  - closeCase: updates the case status to closed. The value specified is defined in the cases section of the backend.configuration.js file. See Configuration Options for more information.
- CaseCreate.View.js: defines the view that enables a user to create a new case.

This file contains the following method:

- saveForm: saves the data contained in the view's form. This method is called from the Backbone.View.saveForm.js file.

See User Experience for more information about how these views appear to users in My Account.

### View Templates

Case Management uses the following view templates:

- case_list.txt: template that corresponds to CaseList.View.js
- case_detail.txt: template that corresponds to CaseDetail.View.js
- case_new.txt: template that corresponds to CaseCreate.View.js

## Configuration Options

From the backend.configuration.js file of My Account, you can configure the case status when a user performs one of the following actions:

- Creating a case
- Closing a case

Case statuses are defined in Setup > Support > Case Statuses.

You can also configure the value of the Origin Type for the case record. Origin Types are defined in Setup > Support > Case Origin Types.

the backend.configuration.js file contains an object called `cases`. By default this object contains the following parameters:

```
             cases:
{
   default_values:
   {
      status_start {
         id: '1'
      ,   name: 'Not Started'
      }
   ,   status_close {
         id: '5'
      ,   name 'Closed'
      }
   ,   origin {
         id: '-5'
         name: 'Web'
      }
   }
}
```

The `cases` object also contains a parameter called `default_values`. This parameter defines how the My Account Reference implementation sets the status and origin of the case record in NetSuite.

## Upgrading from a Previous Version

The Case Management feature is new in Reference My Account.

## Customizing Case Management

Currently, there are no specific customization techniques used for the Case Management feature. However, Case Management can be customized as needed following standard customization procedures. For complete details on how to customize reference implementations, see the help topic Customizing Implementations.

# Quotes

Reference My Account enables users to receive and view quotes directly from the My Account user interface. This is handled using the estimates functionality in NetSuite. Estimates are non-posting transaction that can be converted to a sales order, invoice, cash sale.

For more information about using Quotes within Reference My Account, read the following topics.

- User Experience
- Setup
- Upgrading from a Previous Version
- Customizing Quotes

## User Experience

Refer to the *SuiteCommerce Advanced User Experience Design Document* for a complete overview of the user experience.

Users can use the Quotes feature to perform the following from My Account:

- View a list of quotes provided to the user.
- View details of a specific quote.

To perform these tasks, the Quotes features uses two views.

### Quote List View

This view displays a list of the quotes that have been provided to a user. Users can filter the list by specifying a date range. Users can sort the list in ascending or descending order according to the following:

- Status
- Quote number

The Quotes List view displays the quote list in a table containing the following headings:

- Quote number
- Request date
- Expiration date

- Amount
- Status

The quote list can be split across multiple pages or displayed in a single page.

## Quote Details View

The Quote Details view displays the following information about a specific quote:

- Quote number: contains number of the quote. NetSuite generates this number when the estimate record.
- Request date: specifies the request date of the quote.
- Expiration date: specifies the expiration date of the quote as defined in the estimate record in NetSuite.
- Status: displays the current status of the quote.
- Related items: displays a list of all items included in the quote.
- Discount information: lists an discounts that have been applied to the quote.
- Address: displays the user's address.
- Shipping: displays shipping information for the quote.

The Quote Details view also provides summary information about the quote and a button that enables the user to download the quote as a PDF file.

# Setup

The Quotes feature is installed and configured by default when Reference My Account is installed.

- Prerequisites
- Feature Architecture
- Configuration Options

## Prerequisites

To use the Quotes feature within the Reference My Account, you must set up estimates within NetSuite.

## Feature Architecture

The Quotes feature is implemented across multiple file. The following sections describe each of these files.

### Main Application Module

The main application module for the Quotes features is the Quotes.js file. This file defines the menu items in My Account and loads all required modules. This file contains the following important methods:

- **menuItems**: adds the Quotes module to the menu and sets the appropriate permissions.
- **mountToApp**: appends all required modules to the application object. It also maintains a reference to these objects.

## Backbone Configuration

The Quotes feature uses the following files to define and configure the Backbone model:

- **Quote.Model.js**: defines the Backbone model for quotes and sets initial values.
- **Quote.Collection.js**: defines the required list header modules that support range, filtering, sorting, and pagination.

  This file implements the following important methods:

  - **parse**: processes the server-side search response containing the number of records per page, total records found, and data for each record.
  - **update**: retrieves a new record set based on list headers.

## Routers

The Quotes feature uses one router defined in the Quote.Router.js file. This file contains the following primary methods:

- **list**: returns the quotes list using the format: `/quotes + HTTP GET`.
- **list**: returns the quotes list using the format: `/quotes?:options + HTTP GET`.
- **details**: returns the details for a specific quote using the format: `/quotes/:id + HTTP GET`.

## Views

The Quotes feature uses the Quote.View.js file to define the views displayed in My Account. This file defines the following objects:

- Views.List object: extends the Backbone.View and defines the following important methods:
  - **initialize**: creates the default filter options retrieved from the back-end. This method calls listenCollection and setupListHeader.
  - **listenCollection**: listens for changes on the quotes collection. The list header module requires this method.
  - **setupListHeader**: initializes the list header and loads the filters with values retrieved from the back-end. This method specifies the range, sorting, and filtering of the list collection.
  - **setLoading**: defines the list header module if the record list is still being loaded.
  - **rangeFilterOptions**: specifies an object containing the default values of the range options filter.
  - **filterOptions**: contains an array of objects with the default values of the filter options filter.
  - **sortOptions**: contains an array of objects containing the default values of the sort options filter.
  - **showContent**: renders the Quotes List view. The Quotes.Router.js file calls this method.
- Views.Details object: extends Backbone.View and defines the following important methods:
  - **initialize**: creates the default filter options with option values retrieved from the back-end. This method calls listenCollection and setupListHeader.
  - **showContent**: renders the Quote Detail view. The Quotes.Router.js file calls this method.

## View Templates

The Quotes feature uses the following view templates:

- **quote_list.txt**: corresponds to the Views.List object defined in the Quotes.List.js file.
- **quote_details.txt**: corresponds to the Views.Details object definte in the Quotes.List.js file.

### Configuration Options

You can configure the default number of days before a quote expires. The backend.configuration.js file contains configuration parameters for the Quotes feature. This file defines an object called `quotes`. By default, this section appears as follows:

```
quote: {
     days_to_expire: 7
}
```

The `days_to_expire` parameter defines the number of days before a quote expires.

## Upgrading from a Previous Version

The Quotes functionality is new in this version of Reference My Account.

## Customizing Quotes

Currently, there are no specific customization techniques used for the Quotes feature. However, Quotes can be customized as needed following standard customization procedures. For complete details on how to customize reference implementations, see the help topic Customizing Implementations.

# Reference My Account Templates

This topic identifies the templates used to render each page of the Reference My Account experience.

> **Important:** Web development experience is required to modify any Reference My Account template file.

All the files described in this section can be found in the templates folder after the Reference My Account bundle has been installed. To access template files, go to Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - My Account > Reference My Account > templates. Template files are locked. To modify these files, copy them to the Custom My Account folder. For details, see Customizing Implementations.

The **layout.txt** template is the container for all of the My Account pages and is used to generate all of the content between the header and footer.

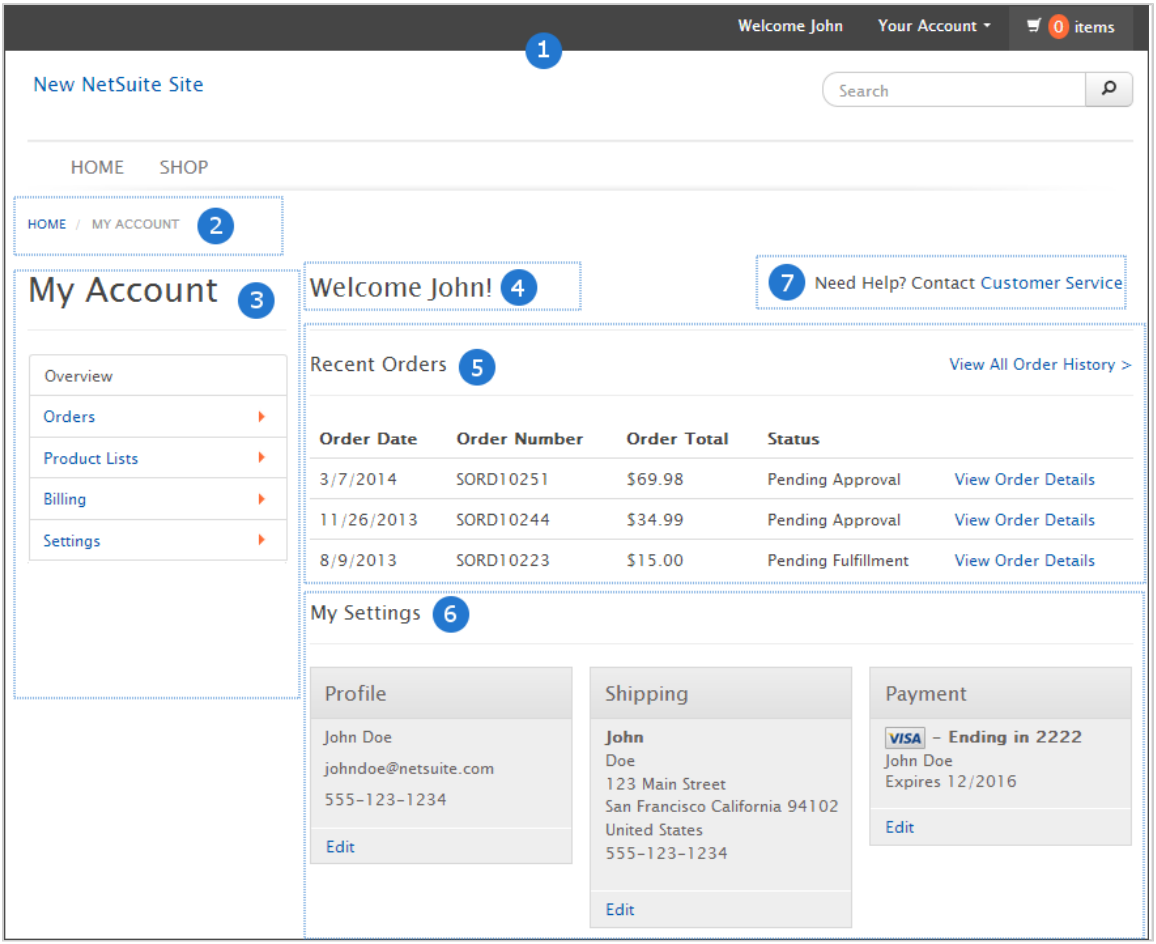
The topics below include more information about each template installed:

- Overview Page
- Orders
  - Order History & Returns Page
  - Receipts
  - Reorder Items Page
- Product Lists
- Billing
  - Account Balance Page
  - Invoices

- Transaction History
- Print a Statement
- Profile Information
- Address Book Page
- Credit Cards Page
- Reorder Items Page
- Email Preferences Page

# Overview Page

When customers successfully sign into the site, or navigate directly to their My Account area, the My Account Overview page is displayed. The main body template for this page is **home.txt.**




| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. Header | Can display your company logo, or a graphic of your choice. | home/home.txt | home_tmpl |
| 2. Breadcrumb Navigation | Updates based on active page within the My Account experience. Clicking a breadcrumb link | globals/macros/breadcrumb_macro.txt | macro |

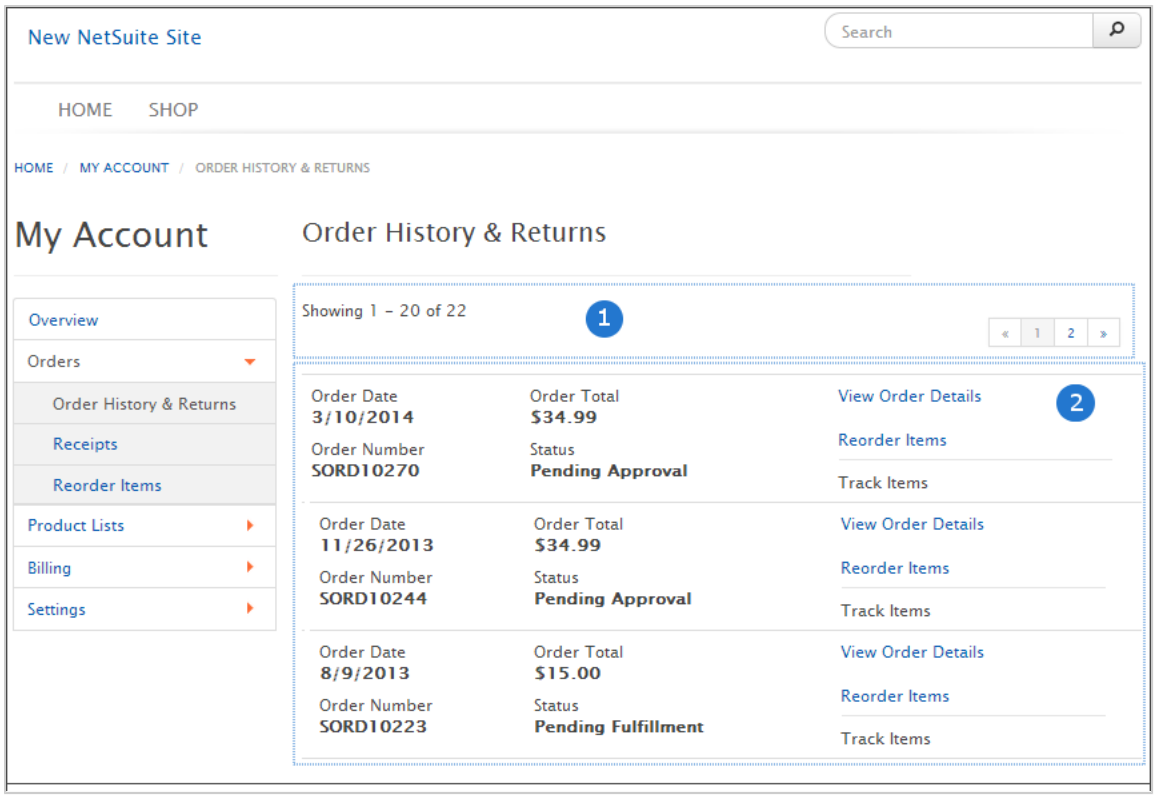
| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| | directs the customer to the target page. The active page is not a hyper link. | | |
| 3. Main Navigation | Is persistently displayed in the left column. Crosslinks are displayed based on the customer role within NetSuite. Clicking a crosslink loads new content into the man body template using an Ajax call. | layout.txt | layout_tmpl |
| 4. Welcome message | Recognizes the customer logged in, and then displays the customer's first name.<br>You can modify the Welcome content in **home.txt.** | home/home.txt | home_tmpl |
| 5. Customer Service | Logged in customers can click a link to contact customer service. This link is not displayed by default. To display the link in your website, set the **customerSupportURL** property in the Configuration.js file to the desired URL. This URL can be set to an absolute or relative link. The Configuration.js file can be found at Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - My Account > Reference My Account > js > src > app. | home/home.txt | home_tmpl |
| 6. Recent Orders | The recent orders section displays the three most recent orders by default. The View All Order History link takes the customer to the Order History page. If a customer ha not placed an order, then a default message is displayed communicating that are no orders in the system. | home/macros/recent_orders_widget_macro.txt | macro |
| 7. My Settings | Provides an overview of the customer's profile, shipping and payment information.<br>The **profile area** contains: Name, Email, Phone, and password. Click Edit to go to the profile information page to update information.<br>The **Shipping** and **Payment** areas contain default address and credit card information as defined on the customer record.<br>If the customer has not indicated a default shipping address or credit card, then The Shipping and Payments areas contain a default message and a link to add a new address or credit card.<br>Click Add New Address or Add a Credit Card to open up a light-box overlay to enter and save information. | home/home.txt<br>home/macros/home_payment_macro.txt<br>home/macros/home_shipping_macro.txt<br>home/macros/home_profile_macro.txt | home_tmpl |

# Orders

- Order History & Returns Page
- Receipts
- Reorder Items Page

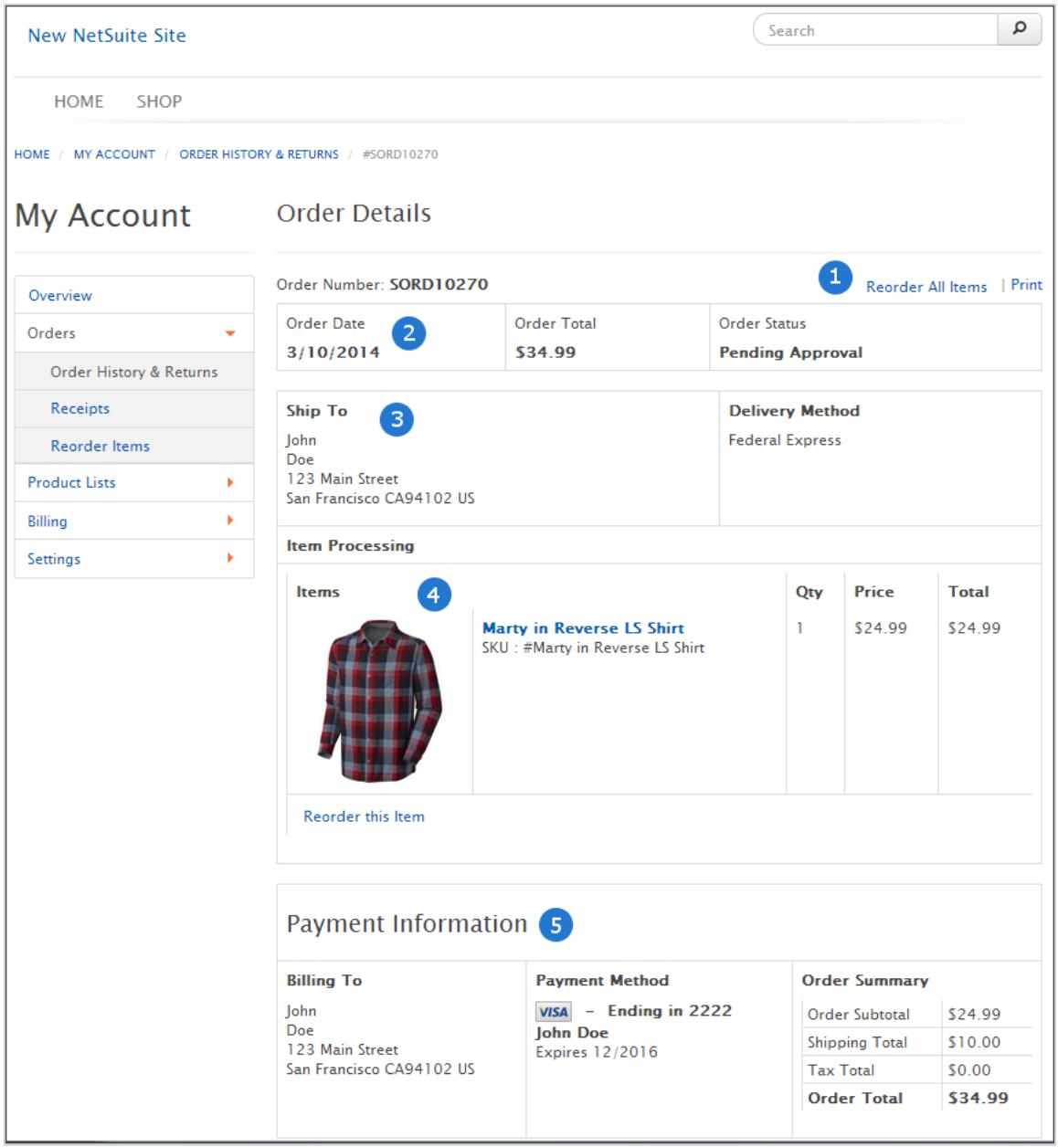
## Order History & Returns Page

The Order History page allows customers to view the orders they have placed and to make requests to return items. The main body template for this page is **order_history.txt.**




| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. View options | Customers can choose to view All Orders or Receipts only. | order/macros/transactions_navigation_macro.txt | macro |
| 2. Orders showing | The message containing the numbered results is in the following format:<br>**Showing <initial number> -<page max> of <total # of orders>**<br>The initial number is<br>**[(current page - 1) * maximum products per page] + (current page - 1)**<br>Page max is<br>**(current page * maximum products per page) + (current page - 1)**<br>If the customer is on the first page of the results, the message is<br>**Showing 1 - <maximum results per page> of <total # of orders>**<br>If there are fewer results than the maximum products per page, the total number of orders is shown as the maximum. | globals/macros/pagination_macro.txt | macro |
| 3. Order Information | One image for each product is shown. Clicking this image takes the customer to the order details page.<br>If there is tracking information for an item, clicking the Track Items link opens a page on a third party shipping website. If there are multiple tracking numbers, a tool tip is displayed that lists all tracking numbers. Customers can click one of these tracking numbers to view the tracking information. | order_history.txt | order_history_tmpl |

When customers click the View Order Details link, a detailed view of the order opens. Order details are available for single orders (with one shipping address or with multiple shipping addresses), in-store pick-up orders, and in-store purchases. The main body template for this page is **order_details.txt**. Each product's details are displayed using **product_details_macro.txt**.



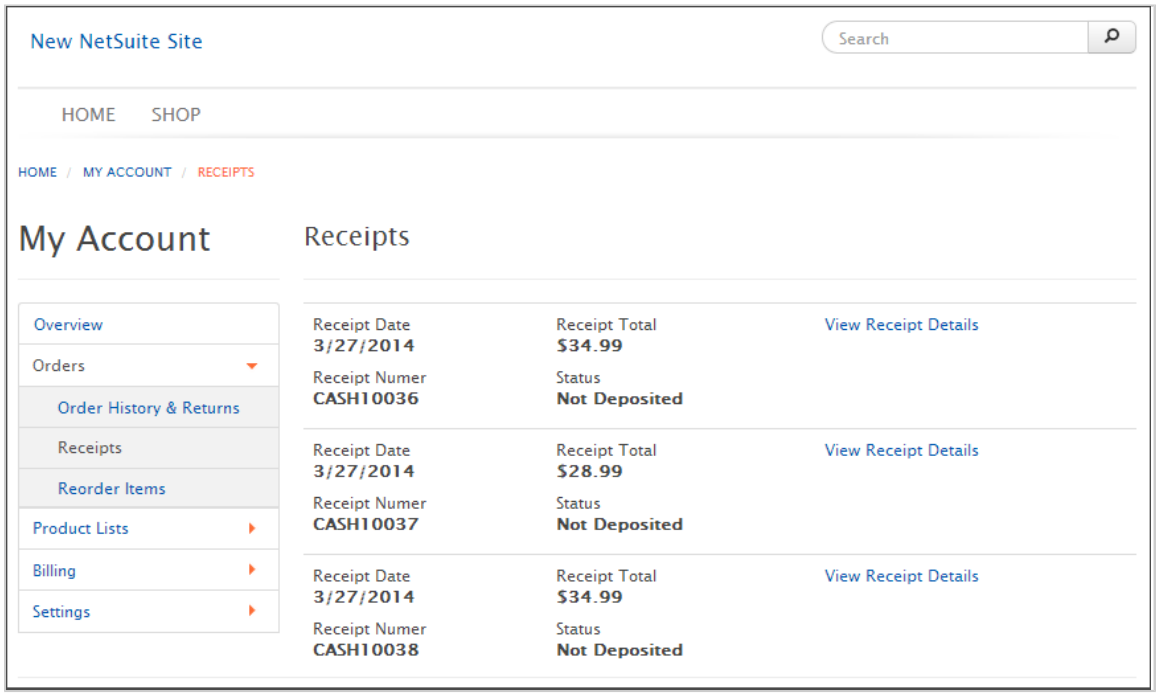

| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. Order links | Clicking the Re-Order All items link opens a new order with the same items as the order the customer is viewing.<br>Clicking the Return Items opens a page with the your company's return policy.<br>Clicking Print opens the order in a printer-friendly format. | order_details.txt | order_details_tmpl |

| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| | Clicking View Invoice opens a detailed invoice view of the order. | | |
| 2. Order header | The order header includes the order number, date, total, and status. | order_details.txt | order_details_tmpl |
| 3. Shipping information | Products are divided into shipments. Each shipment contains the shipping address, delivery method, estimated delivery (if available), and the list of products.<br>Two statuses are possible:<br>▪ **Processing** - Products that are not yet fulfilled are all grouped together and labeled as "Items Processing".<br>▪ **Shipped** - Items have been fulfilled and are grouped by shipment. Items in the same shipment can be fulfilled at different times. When items are fulfilled at different times, the shipment is split into multiple sections. Each section includes the shipping address, delivery method, and items and is ordered by fulfillment status. | order_details.txt | order_details_tmpl |
| 4. Product list | Each product in a shipping group includes:<br>▪ **image**<br>▪ **title**<br>▪ **item number**<br>▪ **configuration options (size, color, etc.)**<br>▪ **quantity purchased**<br>▪ **price (with sales tax if applicable)**<br>▪ **total price**<br>▪ **re-order link** - clicking this link adds the items to the customer's shopping cart and displays the mini-cart (if available). If the mini-cart is not available, the customer is sent to the Shopping Cart page. | order_details.txt | order_details_tmpl |
| 5. Payment information | The following sections are included:<br>▪ **billing address**<br>▪ **payment method**<br>▪ **payment transactions** - includes credit card name, last four digits of the card, and the amount and date of the purchase.<br>▪ **order summary** | order_details.txt | order_details_tmpl |

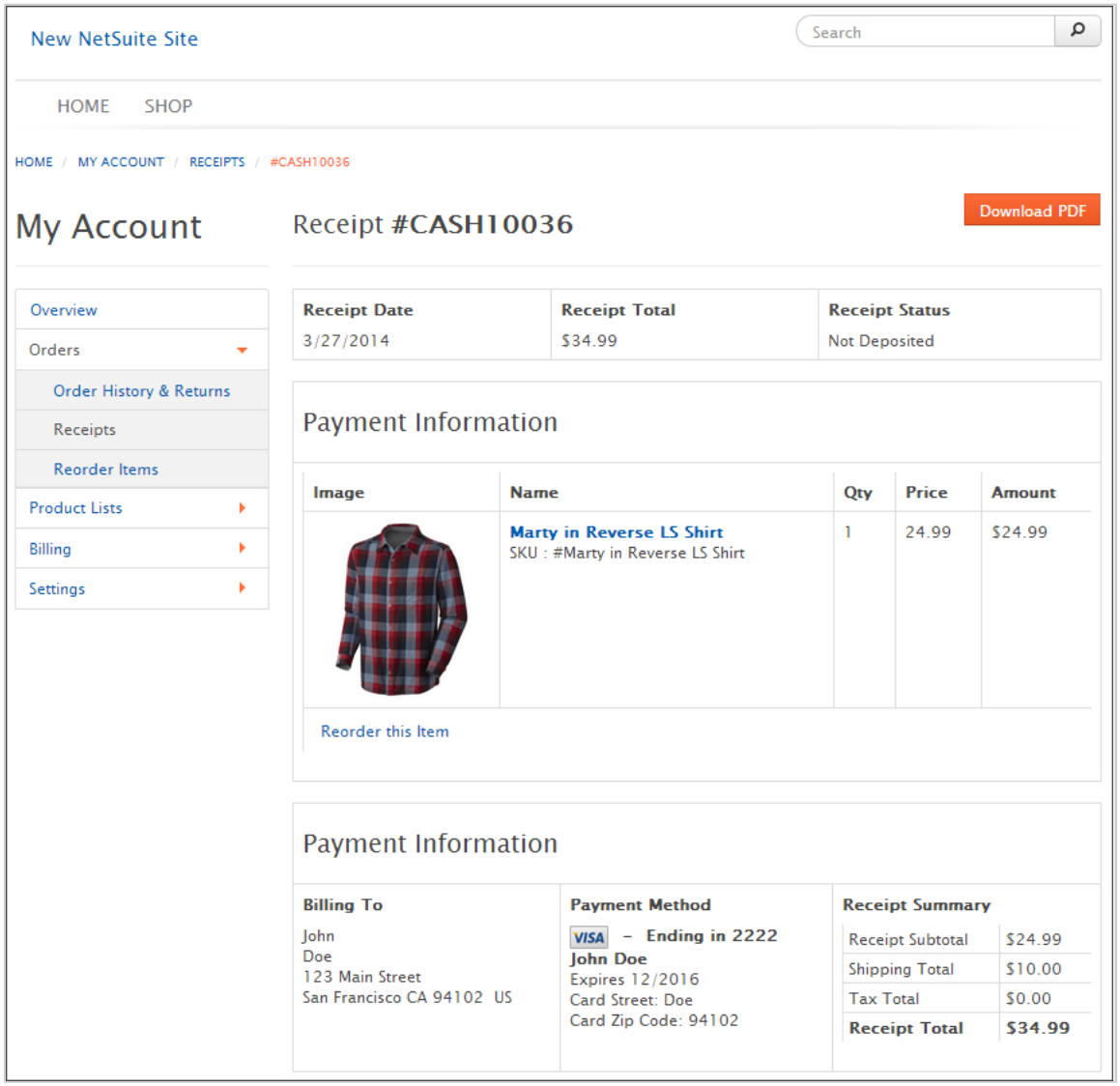
## Receipts

The Receipts page displays a list of all cash sales for the currently logged in customer where payments have been made via credit card or paypal.

**Note:** Purchases paid via an invoice are listed in the Invoices page under the Billing menu. See Invoices.

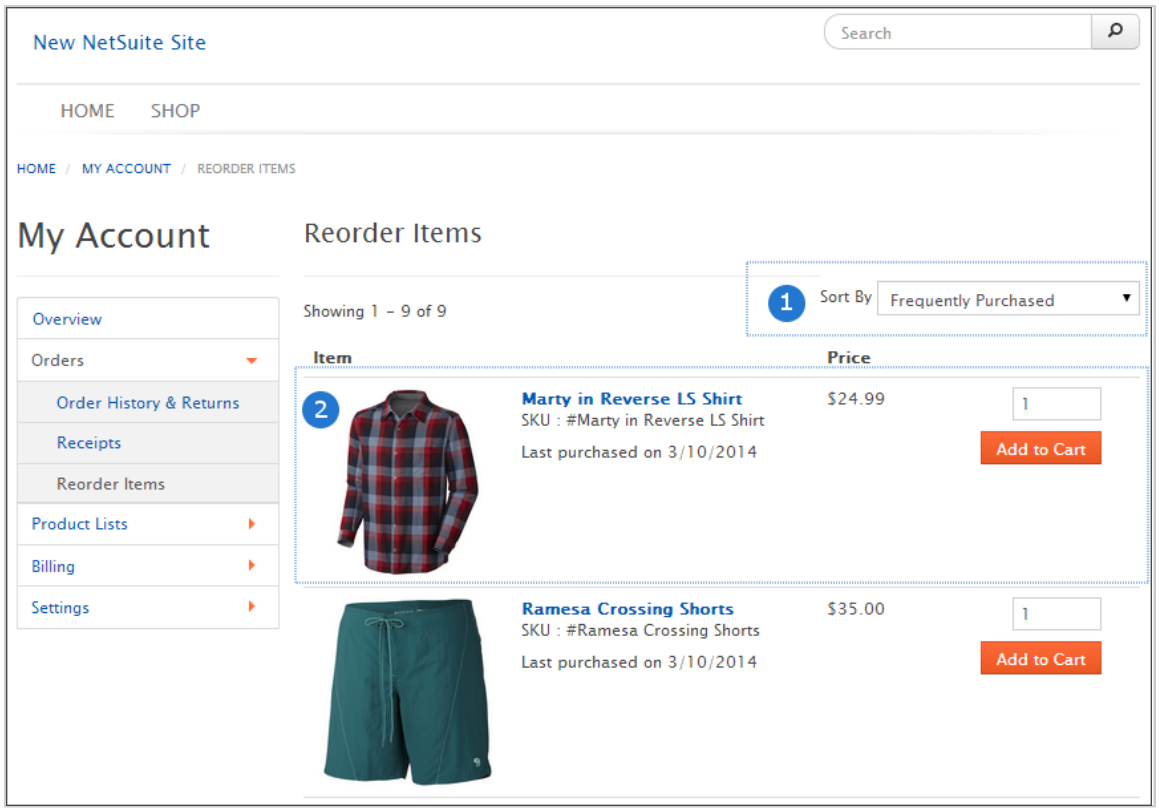


### View Receipt Details




## Reorder Items Page

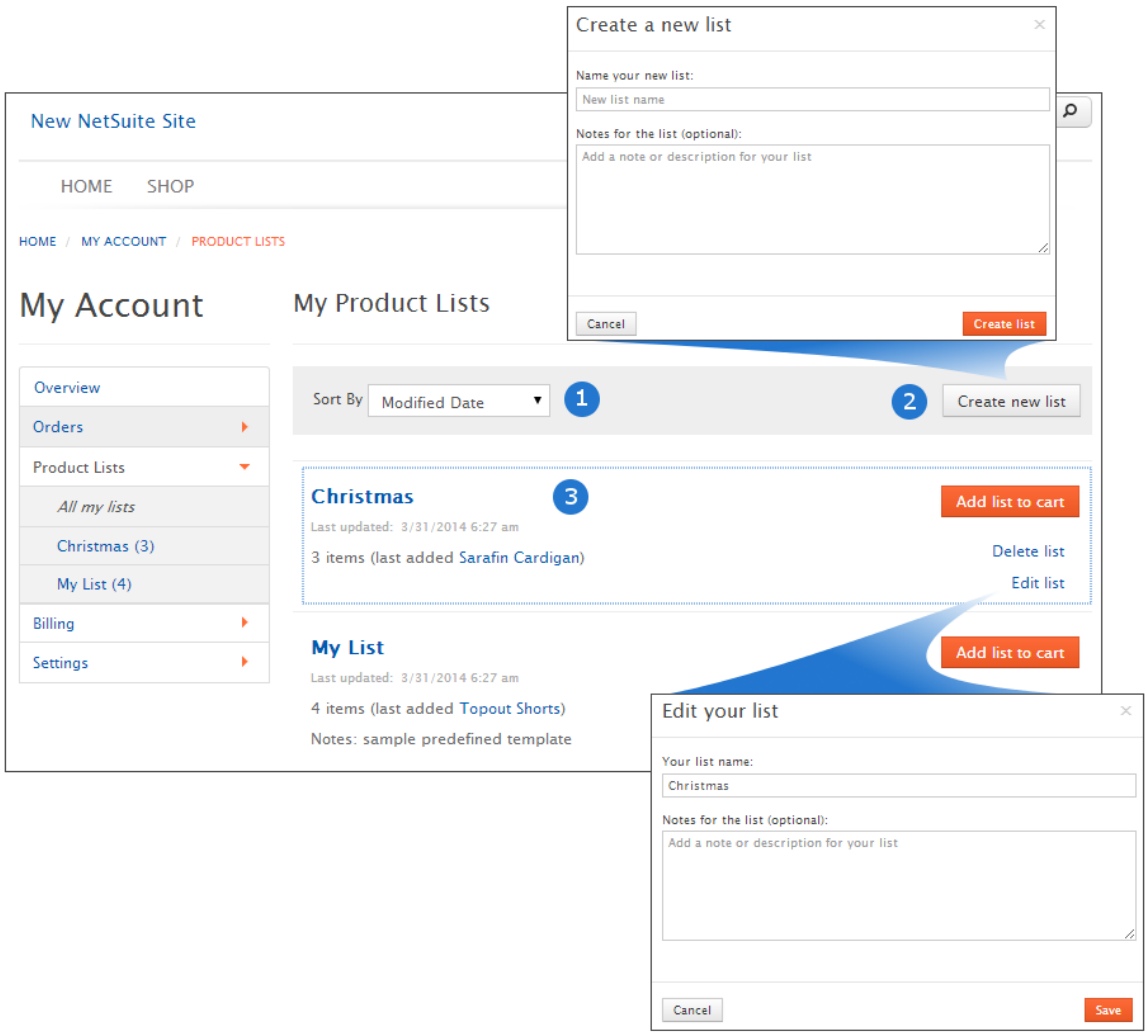
When customers click Reorder Items in the My Account navigation, they are taken to the Reorder Items page. This page lists items purchased and can be sorted. Customers can click Re-Order this Item to add it to the shopping cart. The main body template for this page is **reorder_items.txt**. The product details are displayed using **product_details_macro.txt**.




| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. Add to Cart | Customers can add items to their carts by choosing a quantity and clicking the Add to Cart button. If an item is not available on the website, the Add to Cart button and quantity select are not shown. The stock level of the item is not considered by this page. | product_details_macro.txt | macro |
| 2. Confirmation/ Error Message | This message is shown when an item is added to your cart. This message includes a link to the shopping cart.<br>A different message is displayed if an error occurs when the customer attempts to add an item to their shopping cart. | product_details_macro.txt | macro |

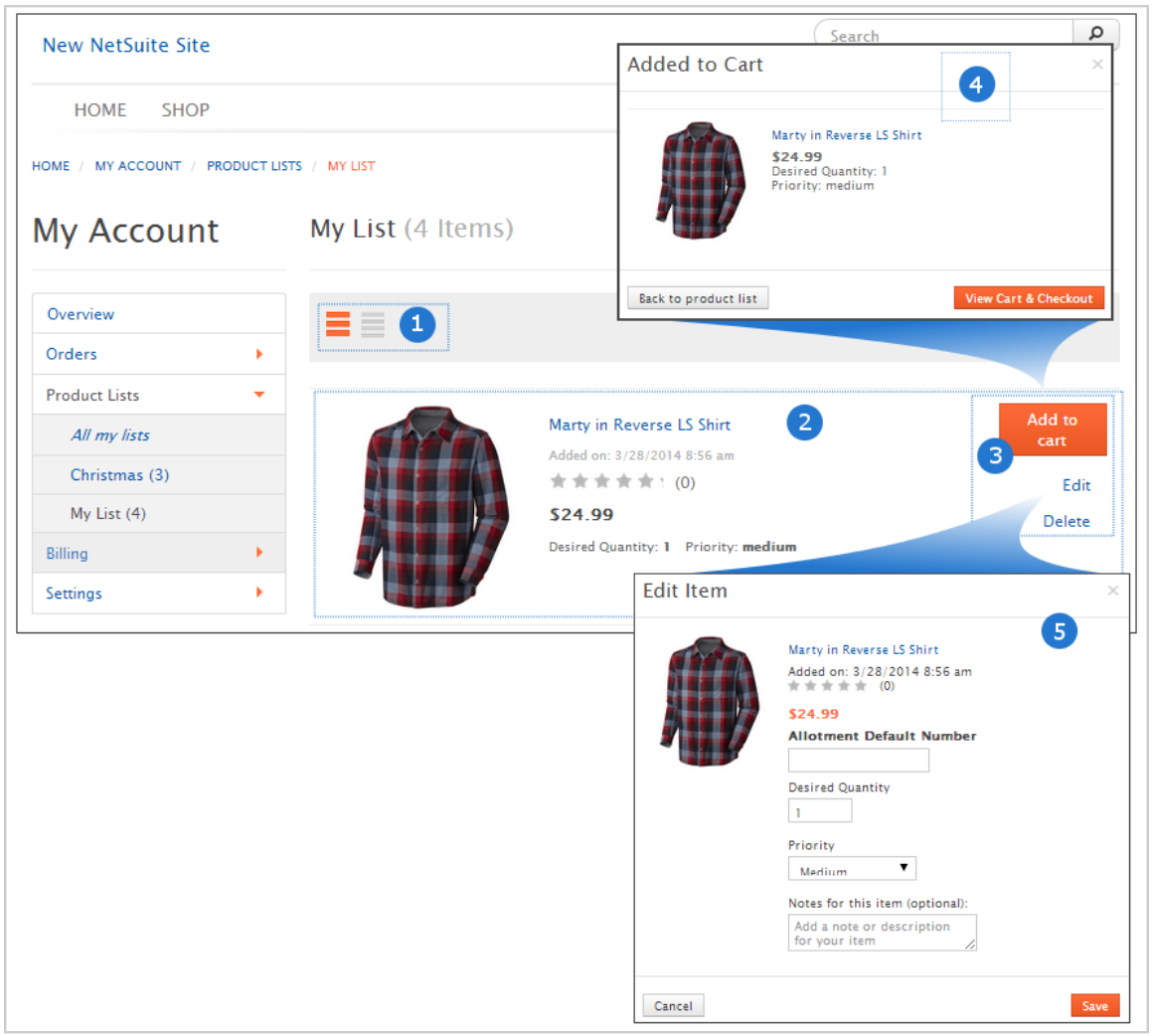
# Product Lists

The Product Lists pages allow users to manage and view product lists. Here users can add, edit, and delete product lists and items within those lists. The main body template for this page is **product_list_lists.txt**.




| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. Sort By | Additional options can be added in the product_lists_lists_details.txt template. The logic for these additional sort options also needs to be added to the Models.js file. | product_lists/ product_lists_lists_ details.txt | product_list_ lists_tmpl |
| 2. Create New List | Opens the create new list modal dialog. | product_lists/ product_list_new.txt | product_list_ new_tmpl |
| 3. List Details | Each available Product List is displayed. Predefined Lists cannot be edited or deleted by the user. Clicking list options (Edit/Delete) displays the options is modal dialogs. | product_lists/ macros/product_ lists_lists_details.txt | macro |

## List Details Page




| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. Listing Display Options | Allows users to toggle between display types. Listing options can be configured in the backend.configuration.js file. By default, the condensed and list views are enabled. You can also add table and grid views. | product_lists/ product_list_details.txt | product_list_ details_tmpl |
| 2. Item Details | Displays details for each item in the Product List. | product_lists/ macros/product_ list_details_full_ macro.txt | macor |
| 3. Options | Displays various options in modal dialogs. | product_lists/ macros/product_ lists_lists_details.txt | macro |
| 4. Add to Cart Confirmation | Displays confirmation of items added to the cart when the Add to Cart button is clicked. | product_lists/ product_list_added_ to_cart.txt | product_list _added_to_ cart_tmpl |
| 5. Edit Item | Displays the edit item modal dialog. | product_lists/ product_list_edit_ item.txt | product_list _edit_item_ tmpl |

| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| | **Note:** If you are unable to save changes when editing matrix items options, there may be an issue with your site setup. Ensure that the matrixchilditems_detail field is defined for the order fieldset as described in Minimum Setup Requirements. | | |

# Billing

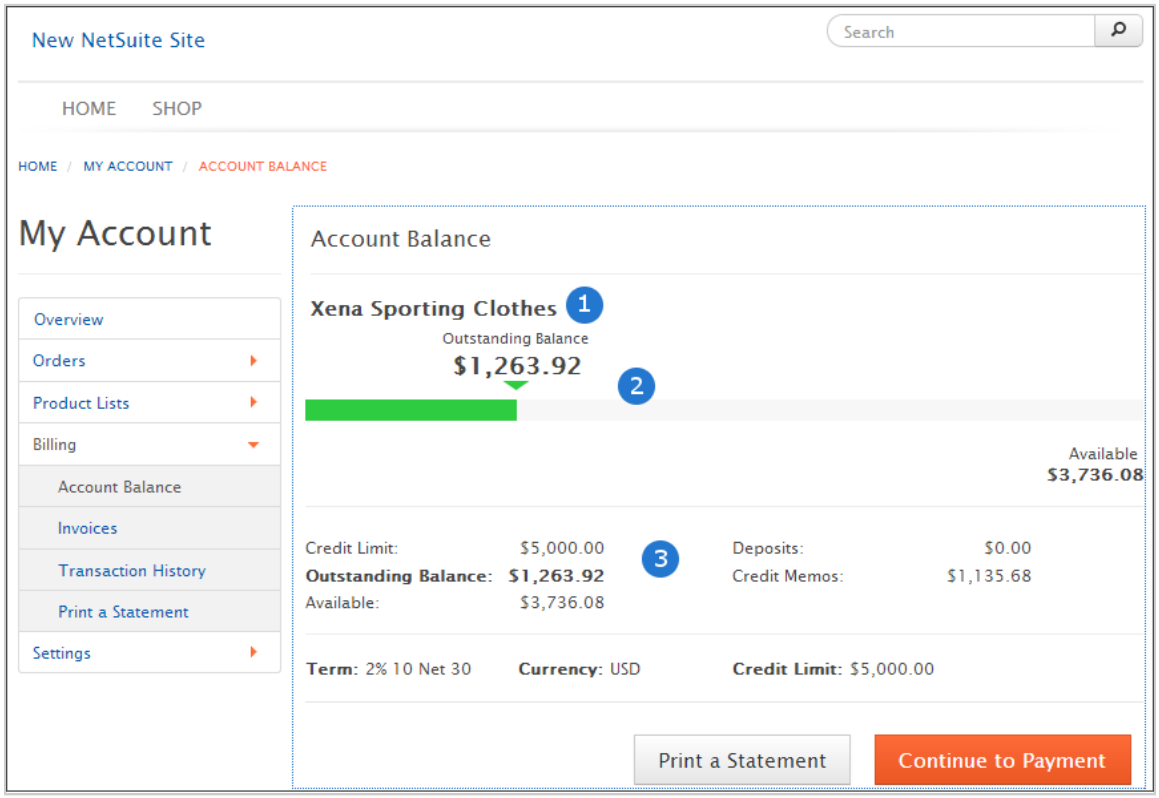
The Billing pages in Reference My Account provide access to billing and payment processes including the following:

- Account Balance
- Invoices
- Transaction History
- Print a Statement

The following sections provide details about the templates and macros used when rendering billing and payment pages. For more information on the user experience and default site workflows, see Billing Experiences & Wireframes.

## Account Balance

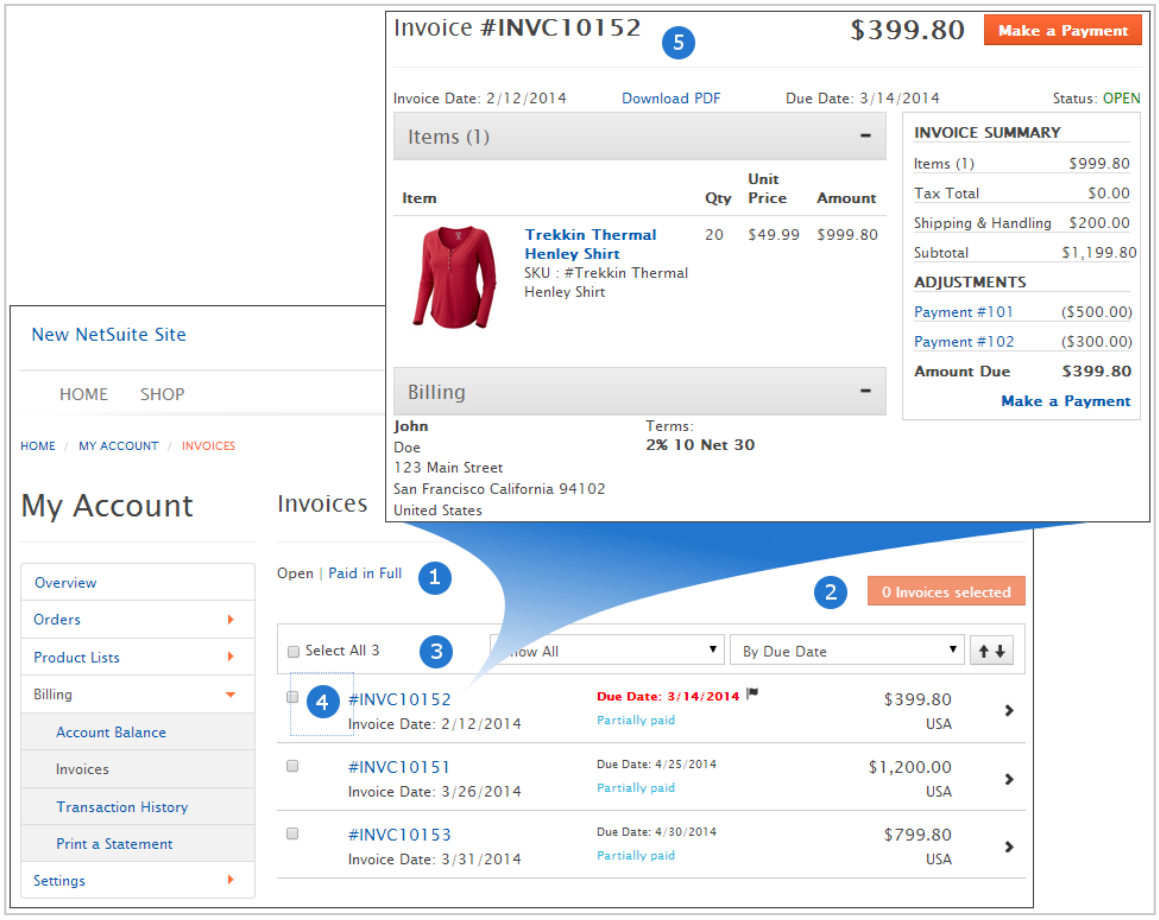
The account balance page displays general account information and allows users to initiate making payments. The main template for this page is **balance/balance.txt**.



| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. | Buyer's company name. This only displays if it is defined in the Customer record. | balance\balance.txt | balance_tmpl |
| 2. | Current outstanding balance bar. | balance\balance.txt | balance_tmpl |
| 3. | Billing Summary. | balance\balance.txt | balance_tmpl |

## Invoices

From the Invoices page, users can view invoices and initiate payments. The main template for this page is **invoice/invoices_list.txt.**

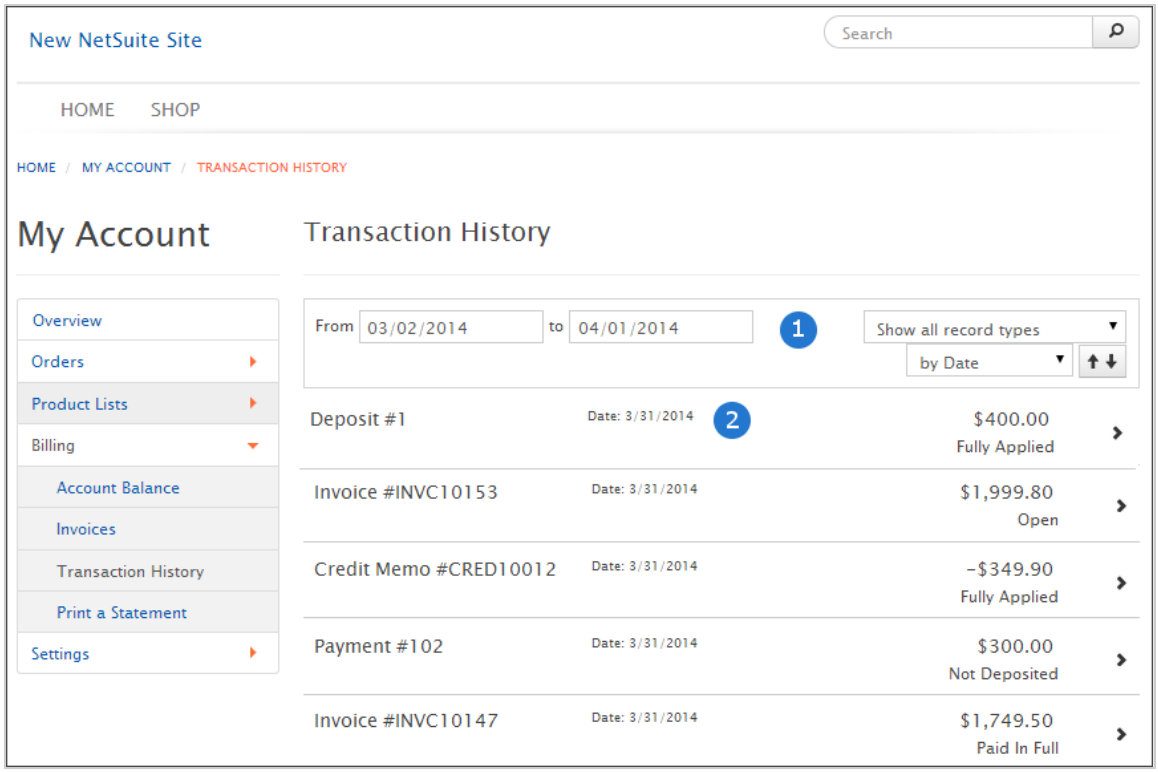



| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. Invoice Type | If the user selects Open, the open_invoices_list.txt template is used. If Paid in Full, the paid_invoices_list.txt template is used. | invoices/open_invoices_list.txt invoices/paid_invoices_list.txt | |
| 2. Filters | Make a Payment button. | | |
| 3. Header | List toolbar control | list_header/list_header.txt | list_header_tmpl |

| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 4. Invoice Line Item | Displays each invoice as defined by the selected filter and date criteria. | invoice/macros/ invoice_list_line.txt | macro |
| 5. Invoice Details | Details of selected invoice. | invoice/invoice_ details.txt | invoice_detail s_tmpl |

## Transaction History

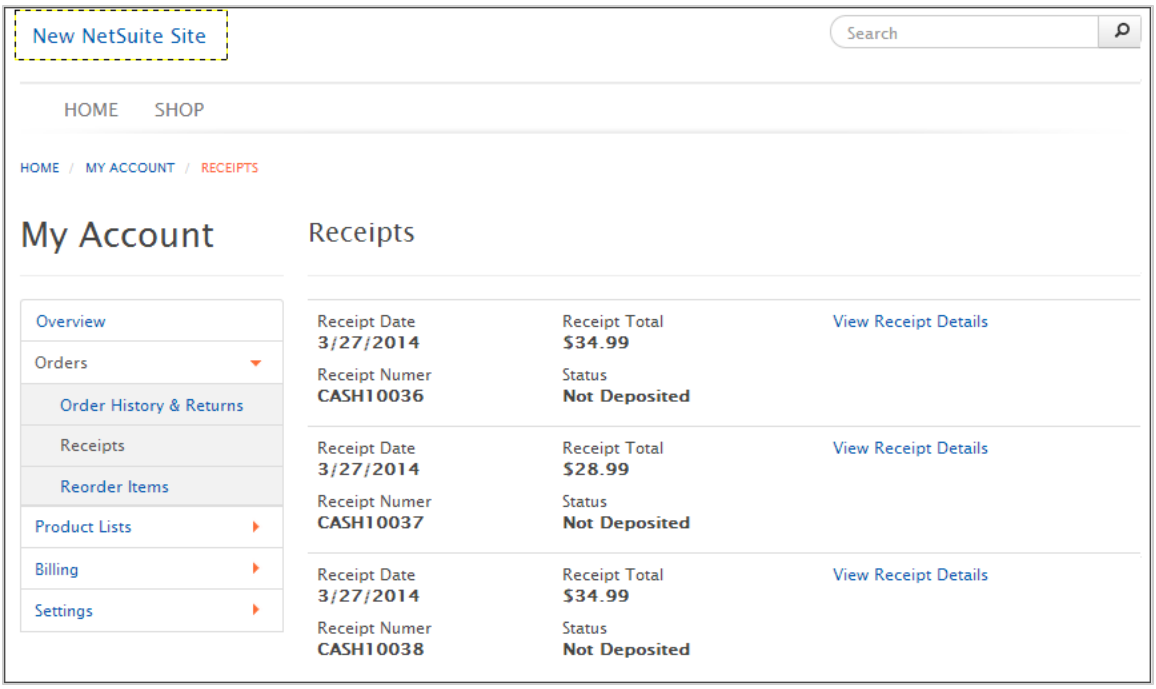
In the Transaction History pages of My Account, users can filter records by type and date and then sort the results. Users can then drill down into individual records. The main template for this page is **transaction_history.txt**.




| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. | List toolbar control | list_header/list_ header.txt | list_header_ tmpl |
| 2. | When no transaction terms are set for the customer the historical items are not displayed. | transaction_history/ transaction_history. txt | transaction_ history_tmpl |

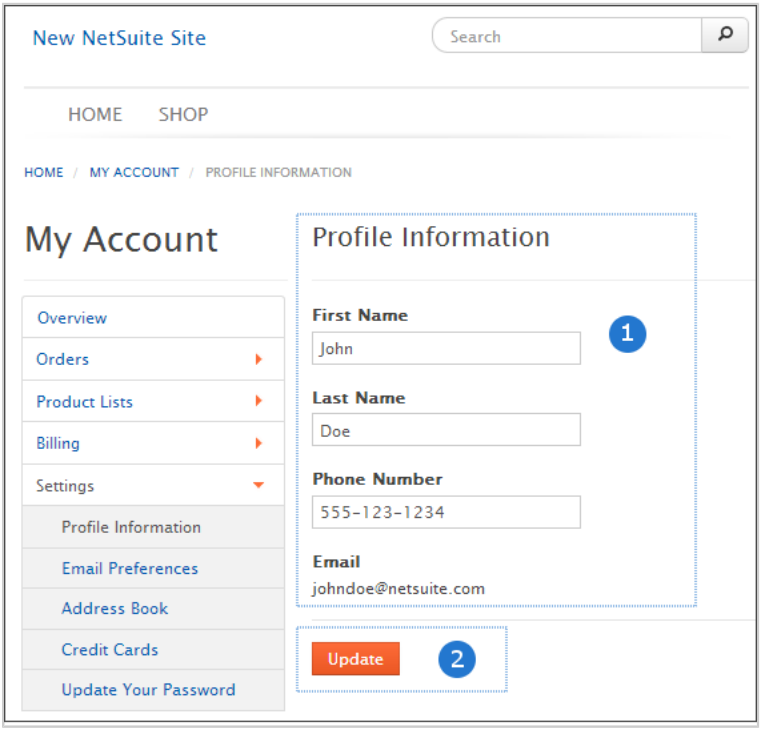
## Print a Statement

From the Print a Statement page, users can print account statements. The main body template for this page is **print_statement.txt**.




# Profile Information

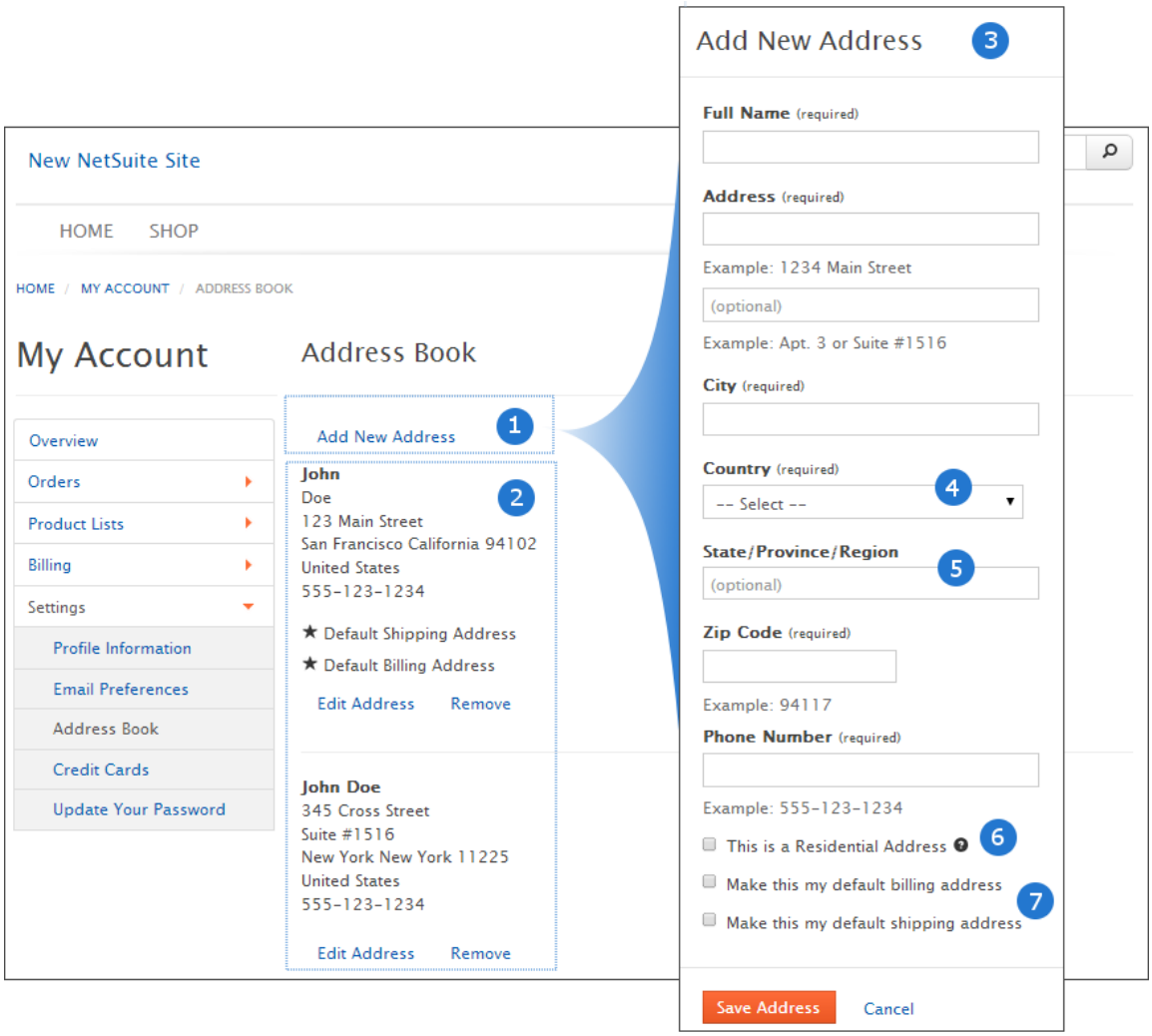
When customers view their profile information, they see the Profile page. Here customers can update their personal information like name, email address, phone number, and password. The main body template for this page is **profile_information.txt.**



| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. Update Your Contact Information | Customers can make changes to their personal information and reset their passwords. | profile/profile_information.txt | profile_information_tmpl |
| 2. Update button | Clicking this button submits the form and updates the customer's information. A confirmation message is displayed at the top of the page when the update is successful. | profile/profile_information.txt | profile_information_tmpl |

# Address Book Page

The Address Book page presents a list of all addresses associated with a customer. Here a customer can add, edit, or remove addresses. The main body template for this page is **address_book.txt**.

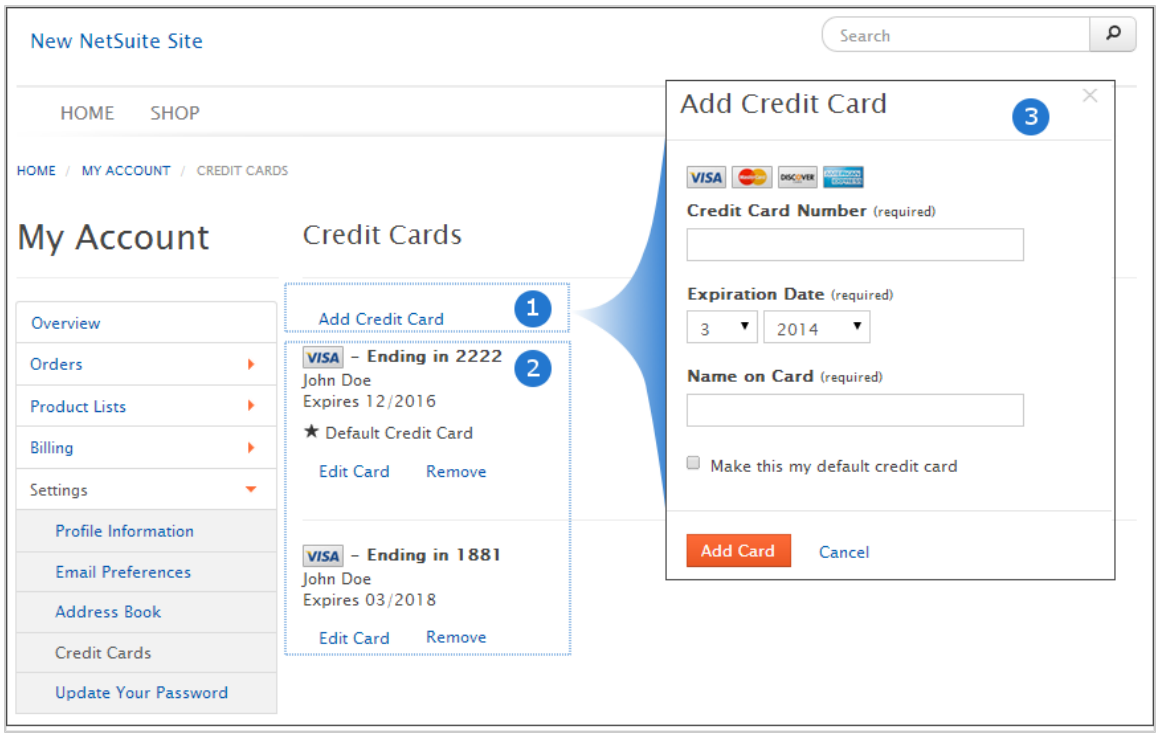



| Page Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. Add New Address | Clicking this link displays the address form in a modal dialog. | address/address_book.txt | address_book_tmpl |

| Page Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 2. Address view | Each individual address is displayed using **address_macro.txt**. Clicking Edit opens a modal dialog the customer can use to update the address. Clicking Remove presents a confirmation message and allows a customer to delete the address.<br>The default billing and shipping addresses are shown in the top row. | address/macros/ address_macro.txt | macro |
| 3. New/Update Address Modal | If no address has been entered for the customer, they are presented with the address form. This form is also displayed in a modal dialog when the user clicks Add Address or Update Address. This modal uses **modal_macro.txt**. | address/macros /address_form_ macro.txt | macro |
| 4. Country dropdown select | Customers can select their country in this field. | globals/macros/ countries_macro.txt | macro |
| 5. State dropdown select | If the country selected is United States, customers choose from a list of states. If the country is not United States, they can type the name of the state. | globals/macros/ states_macro.txt | macro |
| 6. This is a business address box | Clicking the question mark icon presents the following message: "Indicating that this is a business address will help us determine the best delivery method for your items." | address/address_ form_macro.txt | macro |
| 7. Default shipping or billing address boxes | Default addresses are shown in the top row of the address book. | address/address_ form_macro.txt | macro |

## Credit Cards Page

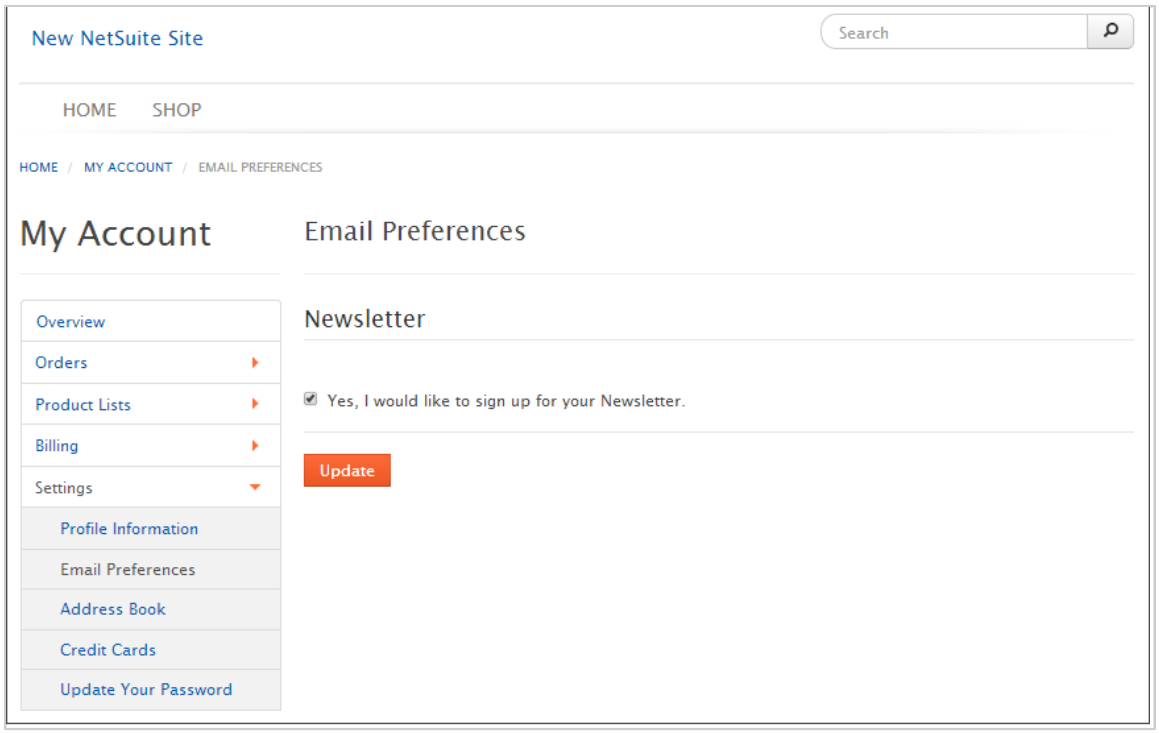
The Credit Cards page lists the credit cards associated with the customer. Customers can add, edit, and remove credit cards. The main body template for this page is **creditcards.txt.**




| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. Add Credit Card | Clicking this link opens the credit card form in a modal dialog. | creditcard/ creditcard.txt | creditcard_ tmpl |
| 2. Credit Card view | Each individual address is displayed using **address_macro.txt**. Clicking Edit opens a modal dialog the customer can use to update the address. Clicking Remove presents a confirmation message and allows a customer to delete the address.<br>The default billing and shipping addresses are shown in the top row. | creditcard/macros/ /creditcard_macro. txt | macro |
| 3. New/Update Credit Card Modal | If no address has been entered for the customer, they are presented with the address form. This form is also displayed in a modal dialog when the user clicks Add Address or Update Address. This modal uses **modal_macro.txt**. | creditcard/ creditcard_form_ macro.txt | macro |

# Email Preferences Page

The Email Preferences page allows customers to subscribe to your marketing campaigns and set other email preferences. The main body template for this page is **email_preferences.txt**. When a customer updates their preferences, a message is displayed confirming their update (or notifying them if an error occurred.) This message is displayed using **message_macro.txt.**


New NetSuite Site
Search
HOME
SHOP
HOME / MY ACCOUNT / EMAIL PREFERENCES
My Account
Email Preferences
Overview
Orders
Product Lists
Billing
Settings
Profile Information
Email Preferences
Address Book
Credit Cards
Update Your Password
Newsletter
Yes, I would like to sign up for your Newsletter.
Update

# Reference Checkout

**Important:** When upgrading to a new bundle version, refer to the Denali Release of SuiteCommerce Advanced for information on enhancements and fixes in that bundle.

Reference Checkout is NetSuite's reference implementation of the web store checkout process, which includes customized login, cart, and checkout pages, along with associated source files and assets. The reference implementation is delivered as a bundle. The implementation leverages the same modern, scalable, and extendable HTML and JavaScript architecture as the Reference Shopping and Reference My Account bundles available for SuiteCommerce Advanced and Site Builder sites. A web designer or NetSuite partner can use Reference Checkout as a base from which to create a unique and innovative eCommerce experience.

The reference implementation is supported for multiple websites, external catalog sites, HTML sites hosted in NetSuite, Site Builder sites, and SuiteCommerce Advanced websites that use the shopping cart included in the Reference ShopFlow bundle. Reference Checkout is also designed using modern Responsive Web Design to dynamically adjust for various device layouts.

For detailed instructions on installing the Reference Checkout bundle, see Managing Reference Implementation Installations.

**Note:** Multiple versions of the Reference Checkout bundle can be installed and deployed to different sites. To differentiate between bundles, unique version information is included in the Root URL and SSP application name (as of Reference Checkout 2.02.0). For example, for the SSP application named Reference Checkout 2.02.0 the Root URL is https://checkout-2–02–0.netsuite.com.

## Reference Checkout Architecture

Reference Checkout includes HTML and CSS templates, JavaScript libraries (jQuery.js, Bootstrap.js, Underscore.js, and Backbone.js), and SuiteScript service files. You can use the supplied code to build an out-of-the box cart and checkout solution with little or no customization, or you can modify and extend the code as needed for a fully customized experience.

For detailed information on the new architecture, refer to the following help topics in the NetSuite Help Center:

- SuiteCommerce Advanced Overview
- Reference Implementations Overview
- Technology and Framework
- Customizing Implementations

# Reference Checkout Flow

The Reference Checkout Flow is by default broken down into three major step groupings:

- Shipping
- Payment
- Review & Place Order

Each step grouping may consist of multiple steps within the grouping. The step groupings and steps are customizable using the Configuration Wizard as described in Checkout Steps Configuration Wizard.

There are a number of settings within your NetSuite account that can affect the flow and availability of certain components by default within the Reference Checkout pages. If you are not seeing the checkout experience you expect, check the following settings:

- **Customer Registration:** The checkout flow includes the ability for a Guest Customer to shop and checkout items only when the Customer Registration site setting is set to optional. This setting can be defined on the Web Site Set Up page on the Shopping tab.

  When Customer Registration is optional a Checkout as Guest button is available. With this option, a guest shopper enters all required information in a series of steps (shipping information, payment methods etc), and at the end of the checkout process is given the option to save their profile and become a registered shopper. When Customer Registration is required, the shoppers information is collected and stored first before finalizing the transaction.
- **Payment Options:** By default there are up to three payment option Tabs displayed including Credit/Debit, PayPal, and Invoice. A tabbed appearance of the Payment Methods page is used if at least two of the payment options are enabled in your site. Otherwise, a non-tabbed appearance is used. PayPal is displayed when the **Allow non-credit card payment methods during Checkout** option is enabled on the Shopping tab of the Web Site Setup page and a PayPal account for the site has been set on the Setup tab of the Web Site Setup page. Customers must have terms set on the customer record and register in your site to be able to pay by invoice.

  > **Note:** The **Bill Me Later** feature within PayPal is not currently supported in Reference Checkout.
- **Security Code:** The security code box is only displayed when the **Use Card Security Code for Credit Card Transactions** option is enabled at Setup > Accounting > Accounting Preferences > Items/Transactions > Payment Processing.
- **Terms and Conditions:** The Terms and Conditions link on the Review page only displays when this is required by the site configuration as defined on the Web Site Set Up page on the Shopping tab. The template used for the Terms and Conditions is as defined here as well.
- **Gift Certificates:** The Gift Certificates module only displays when the Gift Certificates feature is enabled and set up in your account.
- **Promotions:** The Promotions module only displays when the Promotions feature is enabled in your account.

# Checkout Steps Configuration Wizard

Reference Checkout includes a default three-step checkout process that can be completely customized. Each step includes a set of *step groupings* that reference specific modules.

With the checkout configuration wizard you can:

- Add or remove steps from step groupings as needed
- Customize each step to include or exclude modules as needed
- Add configuration parameters to each module
- Extend the implementation to add your own *custom* cart and checkout modules

> **Note:** The default flow shipped with Reference Checkout is fully supported by NetSuite and we have provided the means to customize your checkout flows. However, as with any customizations, when the default flow has been customized NetSuite can not guarantee the outcome.

For example, the following code snippet in the configuration.js file defines the *Shipping* step group. The Shipping step group consists of two steps, *Shipping Address* and *Choose delivery method*. The Shipping

Address step contains a single module and the Choose delivery method step contains two modules. You can redefine the Shipping step group by modifying this code.

```
      checkoutSteps: [
         {
            name: _('Shipping').translate()
         ,   steps: [
               {
                  name: _('Enter Shipping Address').translate()
               ,   url: 'shipping/address'
               ,   modules: [
                     'OrderWizard.Module.Address.Shipping'
                  ]
               }
            ,   {
                  name: _('Choose delivery method').translate()
               ,   url: 'shipping/method'
               ,   hideBackButton: true
               ,   modules: [
                     ['OrderWizard.Module.Address.Shipping', {edit_url: '/shipping/address'}]
                  ,   'OrderWizard.Module.Shipmethod'
                  ]
               }
            ]
         }
      ,   {
            name: _('Payment').translate()
         ,   steps: [
               {
                  name: _('Choose Payment Method').translate()
               ,   url: 'billing'
                  // for each Step, a message that will appear at the end of the step can be co
nfigured.
               ,   bottomMessage: _('You will have an opportunity to review your order on the n
ext step.').translate()
               ,   modules: [
                     'OrderWizard.Module.PaymentMethod.GiftCertificates'
                  ,   'OrderWizard.Module.PaymentMethod.Selector'
                  //   configure the address module to show a "same as XXX address" checkbox
                  ,   ['OrderWizard.Module.Address.Billing', {enable_same_as: true}]
                  ,   'OrderWizard.Module.RegisterEmail'
                  ]
               }
            ]
         }
      ]
```

For each step there are a number of parameters that can be set. The following table outlines the available parameters with a short description.

| Parameter | Description | Value |
|---|---|---|
| name | The literal name for this step. | "Step" |
| url | This step's URL. MUST be unique for each step. | "step-url" |

| Parameter | Description | Value |
|---|---|---|
| continueButtonLabel | The label of the 'Continue' button for this step. For example, for a One Page Checkout configuration it could be 'Place Order'. | "Continue" |
| hideBackButton | If set to true the "Back" button is hidden for this step. | true |
| headerMacro | Defines the main site header displayed for the step. By default the simplified macro is used, but the normal 'header' (or a custom one) can be used. | header |
| footerMacro | Defines the site footer displayed for the step. By default the simplified footer is used. | footer |
| bottomMessage | Defines a message to display at the bottom of the step under the "continue" and "back" buttons. | 'Some message at the bottom' |
| modules | Modules may accept configuration parameters that are be applied to the module for this step. | array of modules |

# Modifying the Reference Checkout Configuration

This section provides some examples of the types of configuration modifications that can be done using the Configuration Wizard.

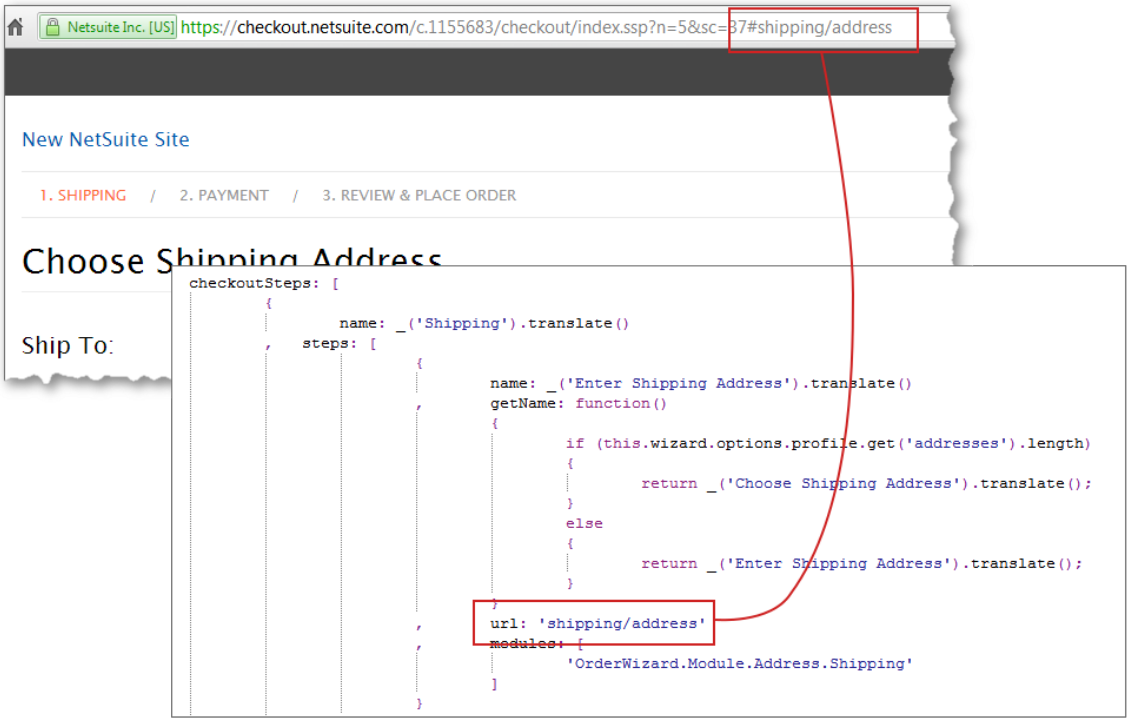
- Setting Custom URLs
- Reordering Modules
- Adding Modules
- Setting Options for Modules
- Reference Checkout

## Setting Custom URLs

Each step has a unique URL. This is important for maintaining statelessness and for effective use of reporting so that you can perform analytics based off of the URLs.

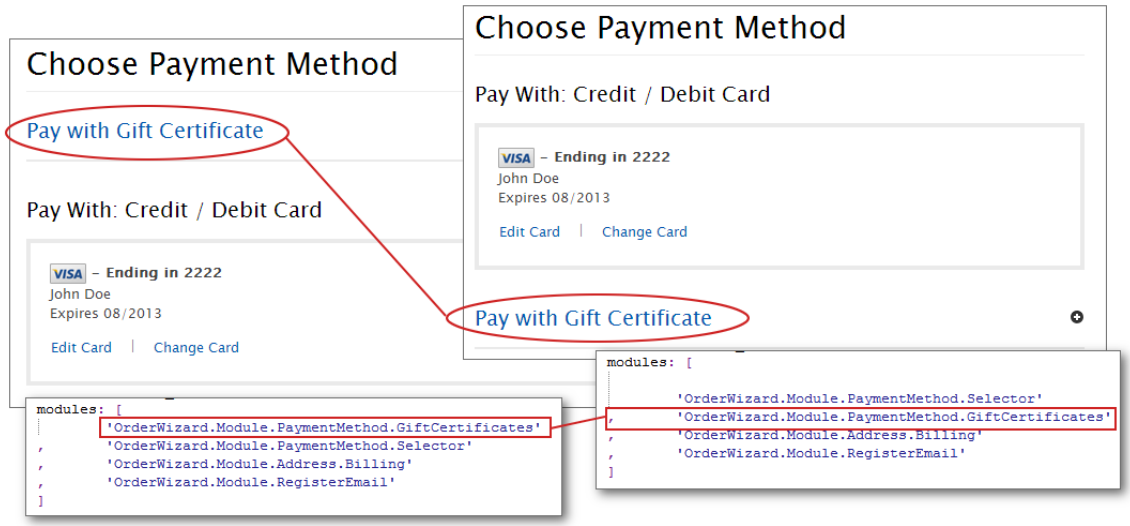
**Important:** Each URL used MUST be unique!

The URL is defined by setting the url parameter for the step.




## Reordering Modules

You can reorder modules within a step group. To reorder, simple reposition the module within the code.




## Adding Modules

By default, several modules are included in the Configuration.js file. These are standard modules that most websites would require. However, included in the implementation are additional modules that can be added to the Configuration.js file. These modules can be found in the **OrderWizard** folder at

Reference Checkout > js > src > app > modules > OrderWizard and will function out of the box. Add the desired module where needed. You can also design and add your own custom modules.

Available modules include:

- OrderWizard.Module.Address.Billing
- OrderWizard.Module.Address.Shipping
- OrderWizard.Module.Confirmation
- OrderWizard.Module.PaymentMethod.Creditcard
- OrderWizard.Module.PaymentMethod.GiftCertificates
- OrderWizard.Module.PaymentMethod.Invoice
- OrderWizard.Module.PaymentMethod.PayPal
- OrderWizard.Module.PaymentMethod.Selector
- OrderWizard.Module.RegisterEmail
- OrderWizard.Module.RegisterGuest
- OrderWizard.Module.Shipmethod
- OrderWizard.Module.ShowPayments
- OrderWizard.Module.ShowShipments
- OrderWizard.Module.TermsAndConditions

You can also add modules to steps in multiple step groupings. For example, by default a guest customer is prompted to register in the final confirmation step. By placing the OrderWizard.Module.RegisterGuest module in multiple steps you can offer the registration at many points during the checkout process.

Each module is designed to use components and naming conventions consistent between the modules. When creating custom modules, we suggest that you follow similar conventions. For example, the RegisterEmail module contains the following components that are common among most other modules:

- **Module definition:** named from the most generic component (OrderWizard) to the most specific component (RegisterEmail)

```
define('OrderWizard.Module.RegisterEmail', ['Wizard.Module'], function (WizardModule)
```

- **Associated Template:** this defines the HTML layout for the rendered output. Templates are found in the templates folder at templates/order_wizard.

```
template: 'order_wizard_registeremail_module'
```

- **Render function:** the wizard object renders each step sequentially as it is defined in the code. The _render(); function is responsible for displaying the template in the layout.

```
,   render: function()
   {
      this.profile = this.wizard.options.profile;

      // if the user is logged we dont render the module
      if (this.profile.get('isGuest') !== 'T')
      {
         this.trigger('ready', true);
      }
      else
```

```
         {
            this._render();
            if (this.profile.get('email') && this.wizard.isPaypalComplete())
            {
               this.trigger('ready', true);
            }
         }
      }
```

- **Submit function:** defines the logic performed when the Continue button is clicked.

```
submit: function()
      {

         var email = this.$('input[name=email]').val()
         ,   newsletter = this.$('input[name=sign-up-newsletter]').is(':checked')
         ,   self = this;

         // if the user is not guest or not change the current values we just resolve the promi
se
         if (this.profile.get('isGuest') !== 'T' || (email === this.profile.get('email') && thi
s.profile.get('emailsubscribe') === (newsletter ? 'T' : 'F') ))
         {
            return this.isValid();
         }

         this.profile.set('email', email);
         this.profile.set('confirm_email', email);

         return this.isValid().then(function() {
            self.profile.set('emailsubscribe', newsletter);
            return self.profile.save();
         });

      }
```

- **isValid function:** defines error checking for entries.

```
isValid: function()
      {
         var promise = jQuery.Deferred();

         if (this.wizard.options.profile.get('isGuest') !== 'T')
         {
            return promise.resolve();
         }

         if (!Backbone.Validation.patterns.email.test(this.wizard.options.profile.get('email'))
)
         {
            return promise.reject(_('Email is required').translate());
         }

         return promise.resolve();
      }
```

## Setting Options for Modules

Some modules are configured to accept options for extra parameters in addition to the parameters available for the step as a whole.

For example, in the shipping step grouping, the OrderWizard.Module.Address.Shipping module is configured by default to include an edit_url parameter directing to the shipping address URL:

```
['OrderWizard.Module.Address.Shipping', {edit_url: '/shipping/address'}]
```

When setting multiple parameters for a module, separate each parameter with a comma:

```
['OrderWizard.Module.ShowShipments', {edit_url: '/shipping/address', show_edit_button: true}]
```

# Setting Up a One-Page Checkout Experience

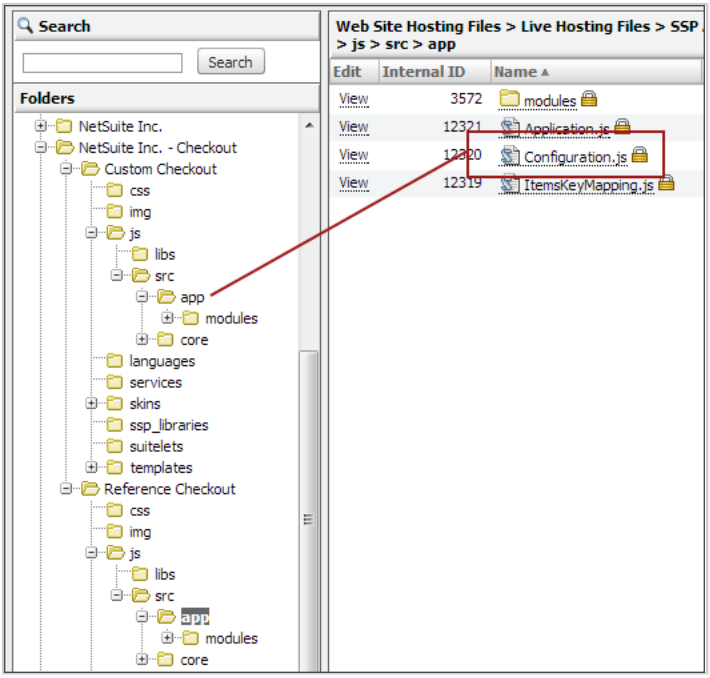
In addition to the default three-step checkout flow, Reference Checkout includes a pre-configured one-page checkout flow. The one page checkout flow consists of two pages for the end customer, the checkout page complete with Shipping, Addressing, Payment processing, Gift Certificate and Promotions processing, followed by an Order Review page. Enabling One-Page Checkout is a matter of editing the Configuration.js file to comment out the default flow and enable the One-Page Checkout flow.

### To enable One-Page Checkout:

1. Edit the Configuration.js file at Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Checkout > Reference Checkout > js > src > app.

   This file can NOT be edited directly from the Reference Checkout SSP application. Copy the file to the same location in the Custom Checkout SSP application folder and edit the file there. For details see Using the Custom Folder.

> **Note:** The easiest way to get your One Page Checkout implemented is to edit a copy of the file from the Custom Checkout SSP application. Always copy files from within the NetSuite file cabinet. Important metadata is lost if the files are copied from within your SuiteScript IDE.




2. Comment out the default three-step checkout and enable the One-Page Checkout configuration.

   The sample code below shows that there are two sections: a sample configuration followed by the default section. Ensure these two sections are commented out but remove the /* */ comments from the One-Page Checkout section.

```
   ,   startCheckoutWizard: true


/*
checkoutSteps: [
   {
      name: 'Step Group'
   ,   steps: [
         {
         ...
         SAMPLE CHECKOUT STEPS
         ...
      ]
   }
]
*/
/*
   ,   checkoutSteps: [
         {
         ...
```

```
         DEFAULT CHECKOUT STEPS
         ...
      ]
*/

      ,   checkoutSteps: [
         {
            name: _('Shipping').translate()
         ,   steps: [
               {
                  name: _('Enter Shipping Address').translate()
               ,   getName: function()
                  {
                     if (this.wizard.options.profile.get('addresses').length)
                     {
                        return _('Choose Shipping Address').translate();
                     }
                     else
                     {
                        return _('Enter Shipping Address').translate();
                     }
                  }
               ,   url: 'shipping/address'
               ,   continueButtonLabel: _('Place Order').translate()
               ,   modules: [
                     'OrderWizard.Module.Address.Shipping'
                  ,   'OrderWizard.Module.Shipmethod'
                  ,   'OrderWizard.Module.PaymentMethod.GiftCertificates'
                  ,   'OrderWizard.Module.PaymentMethod.Selector'
                  ,   'OrderWizard.Module.Address.Billing'
                  ,   'OrderWizard.Module.RegisterEmail'
                  ,   'OrderWizard.Module.TermsAndConditions'

                  ]
               ,   save: function()
                  {
                     return this.wizard.model.submit();
                  }
               }
            ]
         }
      ,   {
            name: _('Thanks').translate()
         ,   steps: [
               {
                  url: 'thanks'
               ,   headerMacro: 'header'
               ,   hideSummaryItems: true
               ,   modules: [
                     'OrderWizard.Module.RegisterGuest'
                  ,   'OrderWizard.Module.Confirmation'
                  ,   'OrderWizard.Module.ShowPayments'
                  ,   'OrderWizard.Module.ShowShipments'
                  ]
               ,   present: function ()
```

```
                {
                    this.wizard.application.trackTransaction(this.wizard.model);
                }
            }
        ]
    }
]
```

3. Reorder, Add, Remove and customize modules as needed for your Web Site implementation as you would when using the default three-step checkout flow.

   See Modifying the Reference Checkout Configuration

# Multiple Ship To

When Multiple Ship To (MST) to is enabled and configured for your web store, users can send groups of items to separate addresses. With MST shoppers can:

- Define different shipping methods for each address items are being shipped to.

  **Note:** Although different shipping methods can be defined for each address, user can not define different shipping methods for the same address within the same order.

- Specify a quantity of items for checkout but then split that quantity as needed to send to multiple different addresses.
- Work within the default flow or within a One Page Checkout flow.

**Important:** In this Limited Release version of MST there are several known limitations. See Known Limitations for further details.

For more details, see the following topics:

- User Experience
- Setup
- Feature Architecture
- Known Limitations

## User Experience

When using MST, users typically proceed through the following steps:

1. A user adds items with specified quantities to the Cart as usual and proceeds to checkout.
2. If the user has more than one shippable item, they can choose to ship to multiple addresses.
3. The user selects an address and creates a shipment *package* for that address by checking which items in the cart should be shipped to that address. At this time a user can split a quantity of items by defining the quantity to be sent to the address selected. Remaining quantities are available when an additional address is selected.

   **Note:** If no shipping addresses exists, the user is prompted to enter one. If a single address is available the MST flow can still be used. Two addresses are not required.

4. The user creates additional shipment packages until all items have been allotted to a shipment package.

5. For each shipment package, the user defines a delivery method.
6. The user selects a payment method and completes the order.

Refer to the *SuiteCommerce Advanced — Multi Ship To User Experience Design Document* for a complete overview of the user experience.

# Setup


- Prerequisites
- Feature Architecture
- Configuration Options
- Known Limitations


## Prerequisites

In order to use MST, the following prerequisites must be met:

- The **Multiple Shipping Routes** feature must be enabled in your account. For detailed information on enabling this feature and using Multiple Shipping Routes with NetSuite, see the help topic Multiple Shipping Routes.
- The `isMultiShippingEnabled` object must be added to the configuration file and set to `true`. See Configuration Options.
- Configure your site setup in NetSuite to use the Standard Sales Order type.

  Go to the Web Site Setup record for your site at Setup > SuiteCommerce Advanced > Set Up Web Site. On the Setup tab, in the Preferences section, select the **Standard Sales Order** option for the Scripting Template fields.

## Configuration Options

To enable or disable the MST edit the `isMultiShippingEnabled` object in the ssp_libraries/backend.configuration.js file. By default this is set to `true`.

```
,   isMultiShippingEnabled: true;
```

**Note:** By default this object is not present. If the `isMultiShippingEnabled` object is not present it is as if it is set to `false` and MST will not be enabled.

Non-shippable items are not processed by MST logic. Non-shippable item types are defined in the `non_shippable_types` object in the ssp_libraries/backend.configuration.js file. You can modify the array of non-shippable items defined within this object as needed.

```
non_shippable_types: ['GiftCert', 'Service','DwnLdItem','Markup','OthCharge','Discount']
```

# Feature Architecture

This section describes the implementation of MST with descriptions of related files and unique characteristics of MST modules.

> **Note:** When using MST, groups of items that share an address and a delivery method are identified by a unique address id.

## Modules

The primary modules used to define the behavior of MST include the following at js/src/app/modules/OrderWizard:

- **OrderWizard.Module.MultiShipTo.EnableLink.js**: enables the MST link when MST criteria are met. Checks to see if the MST feature is enabled and verifies that items in the cart are deliverable items (versus services for example). The link is defined in the order_wizard_msr_enablelink_module.txt template.

  Important methods include:

  - `isActive`: checks that the Multiple Shipping Routes feature is enabled within your NetSuite account and that items in the Cart are deliverable.
  - `toggleMultiShiipTo`: switches between MST or Single Ship To.
- **OrderWizard.Module.MultiShipTo.Select.Addresses.Shipping.js**: checks that there are enough valid addresses for MST.

  Important methods include:

  - `setAddress`: this is disabled since in MST multiple addresses are used rather than a single address.
- **OrderWizard.Module.MultiShipTo.Set.Addresses.Packages.js**: defines the group of items that will be sent to the same shipping address. This is done by setting the shipping address for each line item that is intended for that address. There is no representation of the package itself.

  > **Note:** Non-shippable items are handled by the `LiveOrder.Model` in the Order module.

  Important methods include:

  - `applyCurrentAddress`: this method takes the currently selected address and applies it to all checked items. The state is also saved to the backend to ensure that the package creation is not lost on page reload.
  - `initialize`: re-renders the view when a line is updated with a new quantity or changed address for example.
  - `validateLineQuantity`: this method validates the quantity per shipping address in order to allow for splitting a quantity of items into different packages. The logic for splitting packages is handled in the NetSuite back end. This method only defines the split quantity.
  - `updateLineQuantity`: resets the `splitquantity` for a line to the newly entered quantity.
  - `getFirstAddressIdWithoutShipments`: this method is called on save of the `applyCurrentAddress` method causing the next available address id that has not been assigned a line to be rendered.
- **OrderWizard.Module.MultiShipTo.Shipmethod.js** : defines the shipping method per package. The rendered shipping method options can be either a combo box or radio buttons depending on context. Both options are predefined in the `order_wizard_msr_shipmethod_module` template.

  Important methods include:

  - `getPackages`: renders all items grouped by shipping address.
  - `selectDeliveryMethod`: sets the delivery method for each address id and then for each item corresponding to that address id.

> **Note:** In this part of the checkout flow, packages are identified by the address id since there will be only one set of items per address id.

  - `retrieveInitialDeliveryMethods`: retrieves the list of available delivery methods from the NetSuite backend. This method is called in the `initialize` method each time the order is changed since delivery methods may vary depending on order details such as quantity or shipping address.
- **OrderWizard.Module.MultiShipTo.NonShippableItems.js**: this module is only active during a MST checkout. The `getLinesNotShippable` method returns all lines that are non-shippable items.
- **OrderWizard.Module.MultiShipTo.Set.Addresses.Packages.OPC.js** : the logic in this module is similar to the logic contained within the standard checkout flow defined in the `OrderWizard.Module.MultiShipTo.Set.Addresses.Packages` module although it is simplified since multiple views do not need to be rendered for multiple steps. However, the current view must be re-rendered each time a change is made. This is handled in the `render` method which is called by the `initialize` method each time there is a change..

## Proxy Modules

MST leverages the use of Proxies. With Proxies, you can refer to multiple modules from within one module and the module initialized will depend on context. You can also emulate different steps from within a single module. This improves performance by delaying the initialization of a module containing a large object until it has been determined that the object is required.

For example, an address proxy module, `OrderWizard.Module.Proxy.Addresses`, is used to switch between the single ship to module and the multiple ship to module as required.

> **Note:** All of the proxy modules inherit from a base proxy module, `OrderWizard.Module.Proxy`. In general, you should not have to modify the base module.

For each proxy, the list of related modules and the available states are defined. The `isReady` function is used to allow a module to be skipped over. When `isReady` returns `true` that module is not included in the checkout process at that point. The `updateEvents` property is used to specify an array of events that when triggered causes the proxy to check which modules are active via the `isActive` function and render the view accordingly.

To emulate different steps in a flow, each state is defined within the proxy. The URL defined within the state is appended to the step URL as it is defined within the configuration file. For example, in the Addresses proxy the select addresses and packages steps are emulated:

```
this.states = [
   {
      urlOptions: 'state=selectAddress'
   }
,   {
      urlOptions: 'state=packages'
   }
];
```

> **Note:** In the case of One Page Checkout proxies, no states are defined since separate steps do not need to be emulated.

# Known Limitations

Known limitations include the following:

- MST is only available when running web stores from NetSuite accounts on Version 2015, Release 1 or later.
- Promo codes are unsupported in the MST flow. When a user attempts to enter a Promo code in MST, an error is returned. If a user has already entered a promo code while defining a single ship to order and then switches to MST, an error is returned and the promo code is automatically removed.
- MST is limited to defining one shipping method per shipping address. Users can not create multiple shipments to the same address. For example, a user can not order one set of items shipping Home using standard shipping methods and another set of items shipping Home using One Day shipping method.
- There is a known performance impact on the Proceed to Checkout flow when a shipping method with real time rates is used. This is exacerbated for orders with MST. NetSuite recommends that you run performance tests before going live when using real time rates.
- Skip Login is not supported when MST is enabled. See checkout_skip_login.
- MST is not compatible with Scriptable Cart.

# Reference Checkout Template Reference

This section describes the templates used to render each page of the Reference Checkout experience. Templates are described as they are shipped with the bundle. The use and placement of the templates is completely customizable.

> **Important:** Web development experience is required to modify any template file installed by the Reference Checkout implementation.

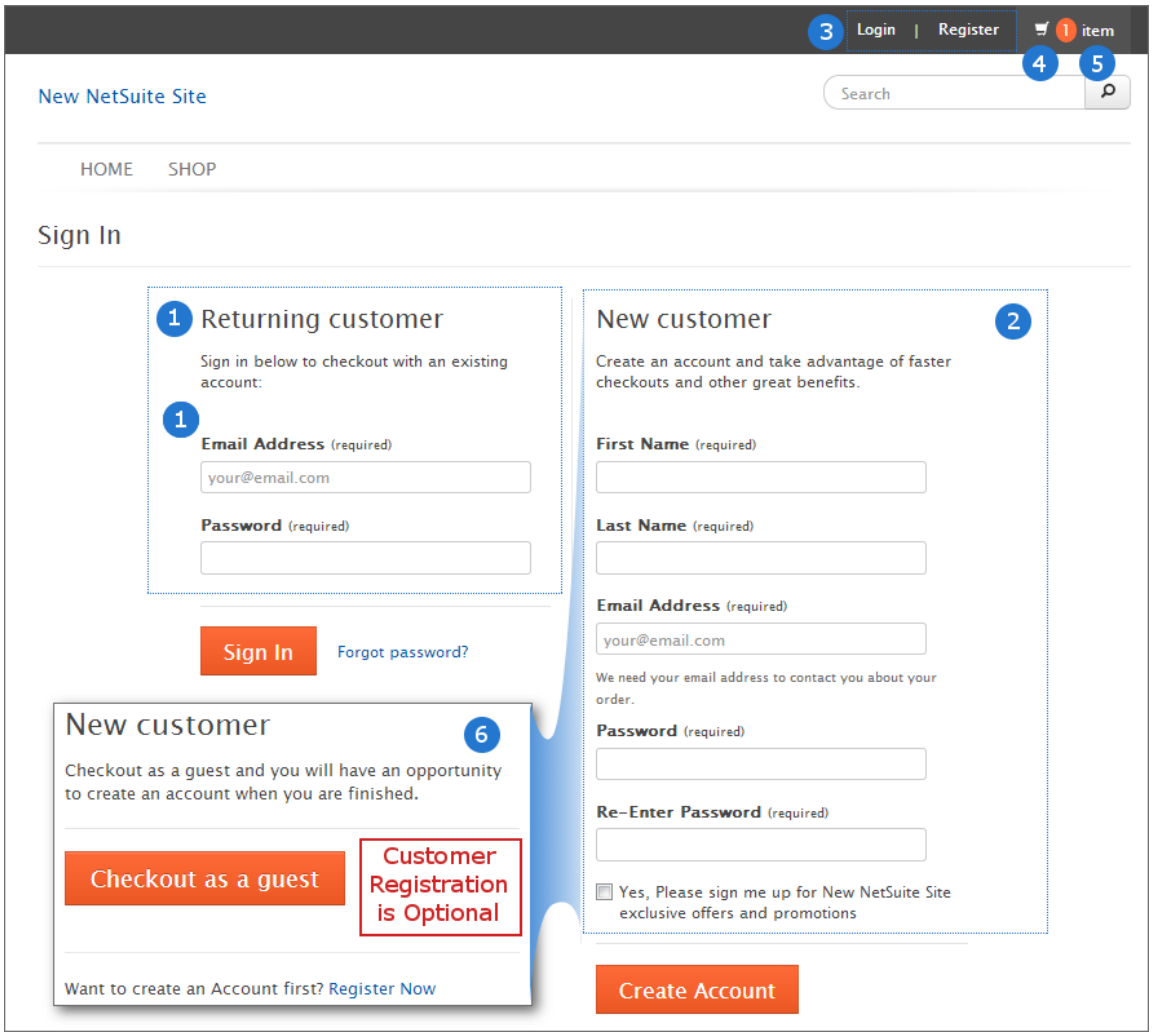
To access template files, go to Web Site Hosting Files > Live Hosting Files > SSP Applications > NetSuite Inc. - Checkout > Reference Checkout > templates. Note that all the template files in the Reference Checkout application folder are locked. To modify any of these files, copy them to the Custom Checkout folder. For details, see Customizing Implementations. The layout.txt template is the container for all of the Checkout pages and is used to generate all of the content between the header and footer.

Each of the following sections describes the templates used for each step grouping in the Reference Checkout experience.

- Sign In
- Shipping Page
- Payment Page
- Review & Place Order

## Sign In

The default display of the Sign In page varies depending on the setting for Customer Registration defined on the Web Site Setup page. When Customer Registration is required, the New Customer form is displayed in the first step. When Customer Registration is set to optional, the Checkout as Guest option is displayed.




| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. Returning Customer | Allows returning customers to enter login information or request a password reset. | login_register/login.txt | login_tmpl |
| 2. New Customer | This template is also displayed when the register now link is selected below the *Checkout as Guest* button. | login_register/register.txt | register_tmpl |
| 3. Login | Link to login page. | header\macros\header_profile_macro.txt | headerProfile |
| 4. Cart | Link to View Cart. | header\macros\mini_cart_macro.txt | miniCart |
| 5. Item Count | Displays number of items currently in the cart. | \header\macros\mini_cart_summary_macro.txt | miniCartSummary |
| 6. Checkout As Guest | This region is displayed instead of the full New Customer form only when site settings define Customer Registration as optional. Customers can select the Register Now link to expand the full registration form. | login_register/checkout_as_guest.txt | checkout_as_guest_tmpl |

# Shipping Page

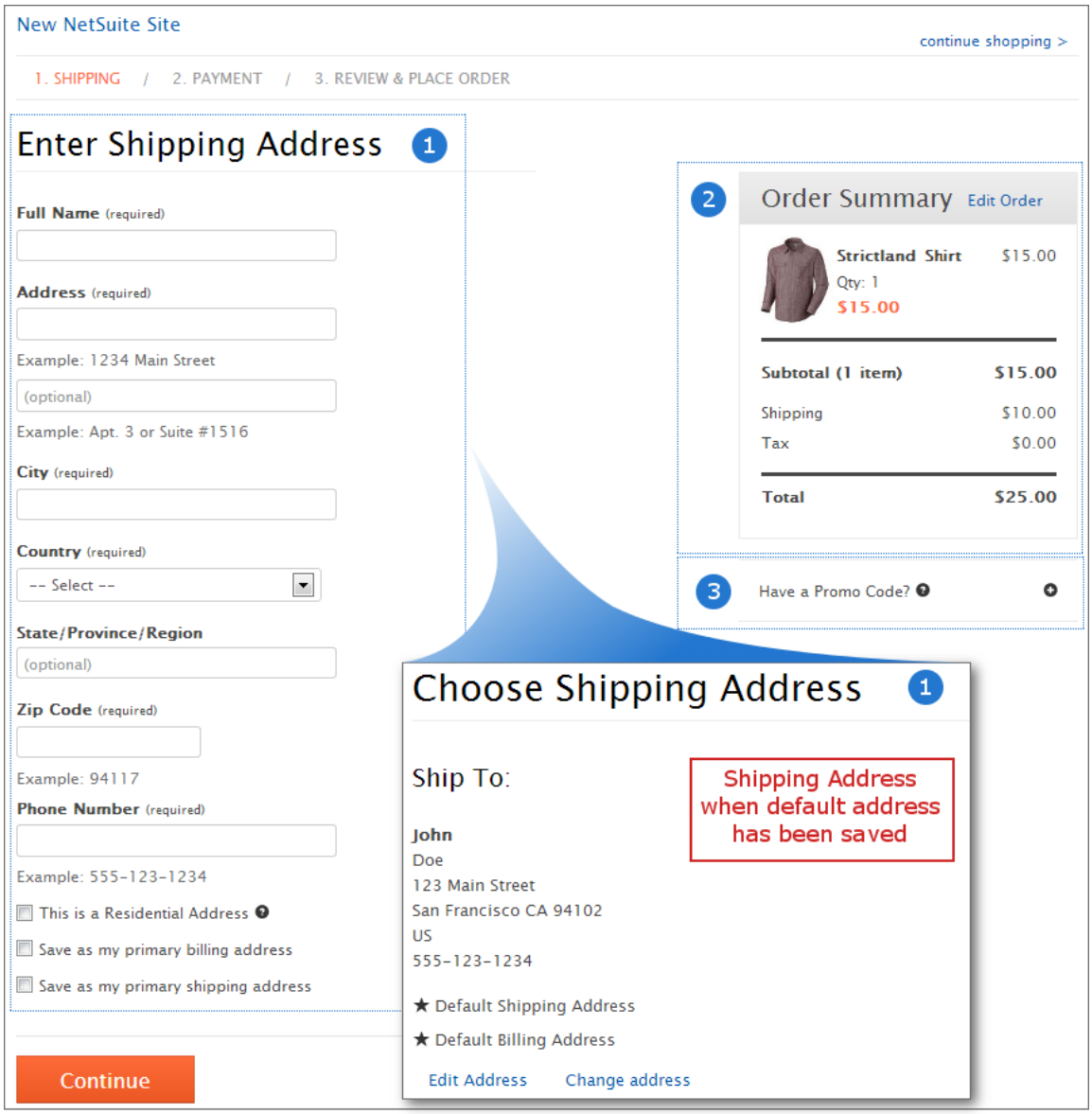
Shipping is the first step group in the process and by default includes the Shipping Address module and Delivery Method module in separate steps.

- Default Modules:
  - OrderWizard.Module.Address.Shipping
  - OrderWizard.Module.Shipmethod
- Default URLs:
  - shipping/address
  - shipping/method

## Shipping Address

Existing customers can select an address from their saved addresses, create a new address or edit an existing address. An address set as the default shipping address is displayed initially. New customers or guest customers must enter their shipping address.

> **Note:** If the Shipping address is overwritten by the Billing address during checkout, examine the **Shipping Information Is Required** setting for the Web Site. When this setting is set to False, the Shipping address defaults to the Billing address. You can find this setting at Setup > SuiteCommerce Advanced/Site Builder > Set Up Web Site > Shopping Tab > Shipping Page > Shipping Information Is Required.



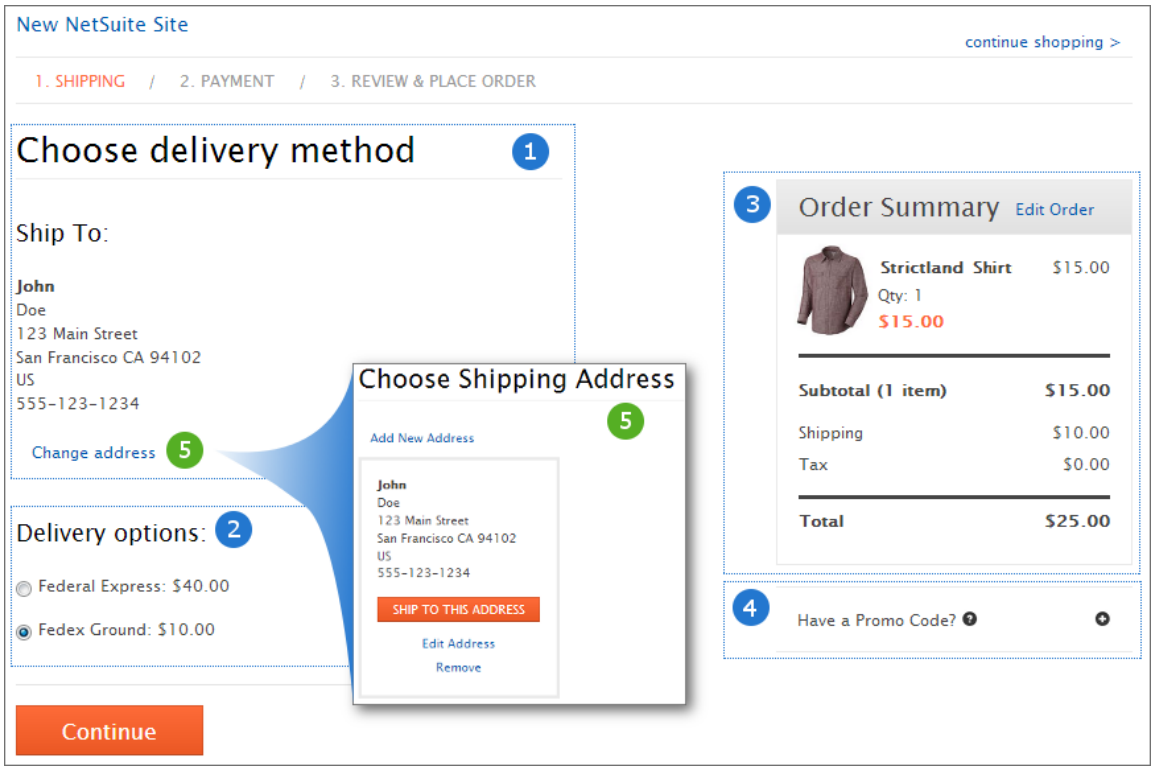

| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. Address | For guest shoppers or customers who do not have a shipping address saved, the default shipping screen is displayed. Otherwise, the default shipping address is displayed with the option to edit or change the selected shipping address. Note: it is NOT the last used shipping address that is displayed by default but instead the address listed as the default shipping address in that customer's profile. | order_wizard/order_wizard_address_module.txt address/macros/address_macro.txt | order_wizard_address_module_tmpl |

| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 2. Order Summary | Displays the summary of the order with tax and shipping information. | shopping_cart\macros\checkout_cart_summary_macro.txt | checkoutCartSummary |
| 3. Promo Code | If customers enter an invalid promotion code, they are presented with an error message. When a promotion code is successfully applied, the estimated total of the order is updated.<br>The promotion code feature in the account must be enable for this to be displayed. | shopping_cart\macros\promocode_form_macro.txt | promocodeForm |

## Delivery Method

The delivery method step allows customers to select the desired delivery method based on the shipping address defined. Customers can also access and change the shipping address from this page.

**Important:** Since the delivery methods available are dependent on the shipping address, the delivery method module must always follow the shipping address module in the Configuration.js file. If the Configuration.js is modified incorrectly such that the delivery method module follows the shipping address module, the site continues to function but errors are thrown in the delivery method module step.



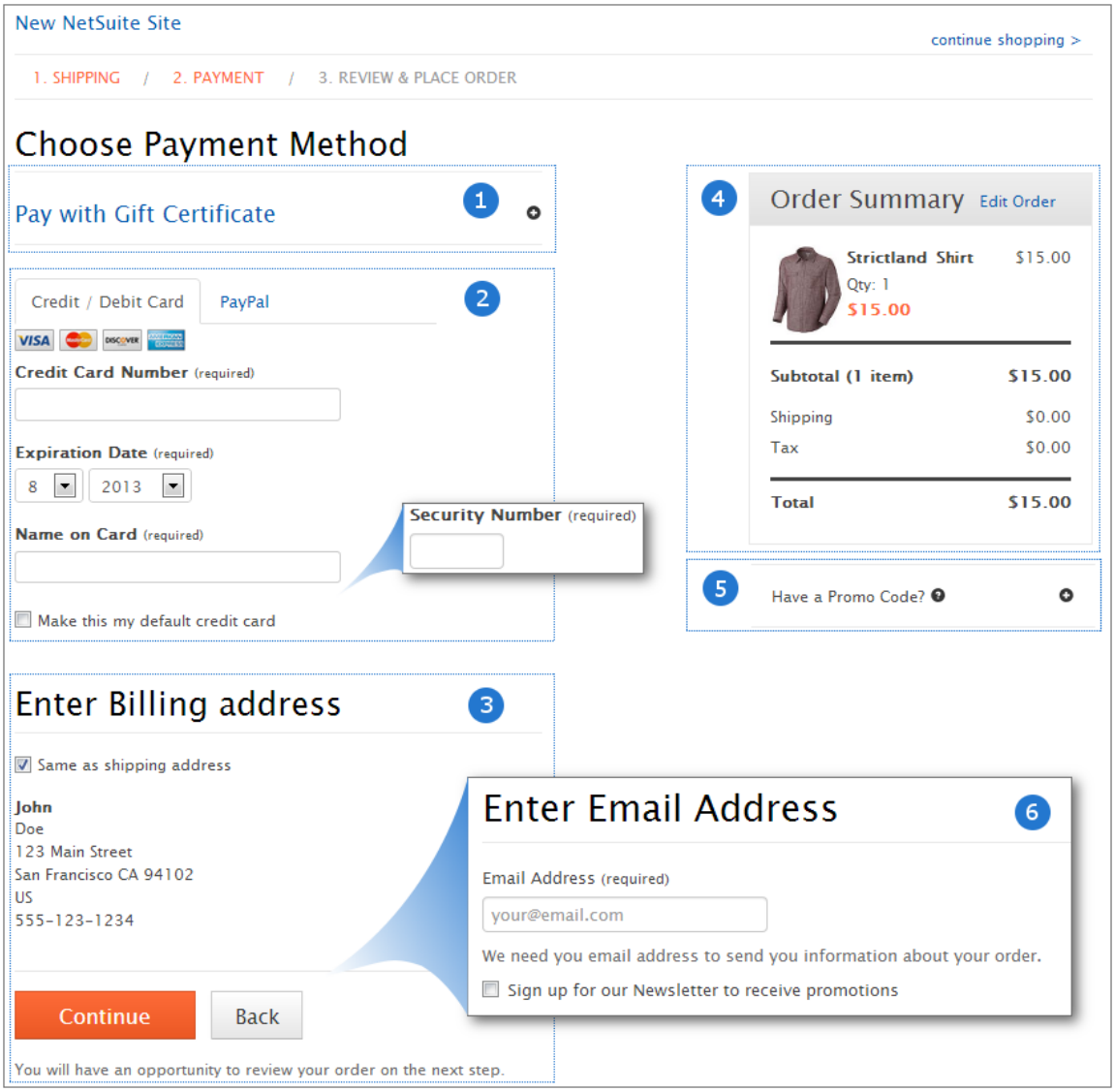

| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. Ship To Address | The address selected in step 1 is displayed. Customers can change the address or add a new address (5). | order_wizard/order_wizard_address_module.txt<br>address/macros/address_macro.txt | order_wizard_address_module_tmpl<br>address_tmpl |

| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 2. Delivery Options | Customers can choose available delivery options based the entered shipping address. Defaults to the *Default Web Site Shipping Method* defined on the Web Site Setup page. | order_wizard/order_wizard_shipmethod_module.txt | order_wizard_shipmethod_module_tmpl |
| 3. Order Summary | Displays the summary of the order with tax and shipping information. | shopping_cart\macros\checkout_cart_summary_macro.txt | checkoutCartSummary |
| 4. Promo Code | If customers enter an invalid promotion code, they are presented with an error message. When a promotion code is successfully applied, the estimated total of the order is updated.<br>The promotion code feature in the account must be enable for this to be displayed. | shopping_cart\macros\promocode_form_macro.txt | promocodeForm |

## Payment Page

The payment page consists of a single step with four modules by default. Each module may or may not render certain output depending on site settings as described in the table below. If multiple credit cards have been defined, a customer can choose which card to use. Otherwise, the customer must enter card information. Registered customers can select a billing address, define a new address, or edit an existing address. New customers must enter billing address information.

- Default Modules:
  - OrderWizard.Module.PaymentMethod.Selector
  - OrderWizard.Module.PaymentMethod.GiftCertificates
  - OrderWizard.Module.Address.Billing
  - OrderWizard.Module.RegisterEmail
- Default URL:
  - billing




| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. Gift Certificate | If your site does NOT support gift certificates, you must comment out or remove the giftcertificates module in the Configuration.js file. | order_wizard/order_wizard_paymentmethod_giftcertificates_module.txt | order_wizard_paymentmethod_giftcertificates_module_tmpl |
| 2. Payment Card | Currently only one credit card can be used per payment.<br>The options available by default depend on your site settings. For example, if both PayPal and Invoicing are set up for your site, then those tabs are also displayed.<br>**Note:** The **Bill Me Later** feature within PayPal is not currently supported in Reference Checkout.<br>When the Account setup requires a card security card code for payment processing the Security Number field also displays (not shown in image). | order_wizard/order_wizard_paymentmethod_selector_module.txt<br>creditcard\macros\creditcard_security_number_form_macro.txt | order_wizard_paymentmethod_selector_module_tmpl |

| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 3. Billing Address | The display varies depending on the customers saved addresses. Customer can select same as shipping address, add a new address, edit the address or remove the address from here. | order_wizard/order_wizard_address_module.txt address/address.txt | order_wizard_address_module_tmpl address_tmpl |
| 4. Order Summary | Displays the summary of the order with tax and shipping information. | shopping_cart\macros\checkout_cart_summary_macro.txt | checkoutCartSummary |
| 5. Promo Code | If customers enter an invalid promotion code, they are presented with an error message. When a promotion code is successfully applied, the estimated total of the order is updated. The promotion code feature in the account must be enable for this to be displayed. | shopping_cart\macros\promocode_form_macro.txt | promocodeForm |
| 6. Email Address | This only displays here when a customer is shopping as a guest. | order_wizard/order_wizard_registeremail_module.txt | order_wizard_registeremail_module_tmpl |

# Review & Place Order

The Review & Place Order page consists of two step groups that allow the user to review the final order details and make changes if necessary, place the order, and receive a confirmation page displaying the order number and details.

Also included in the Review & Place Order step grouping within the Configuration.js file is a **save** function. You can modify this save function to include any custom validation required for your site before an order is submitted.

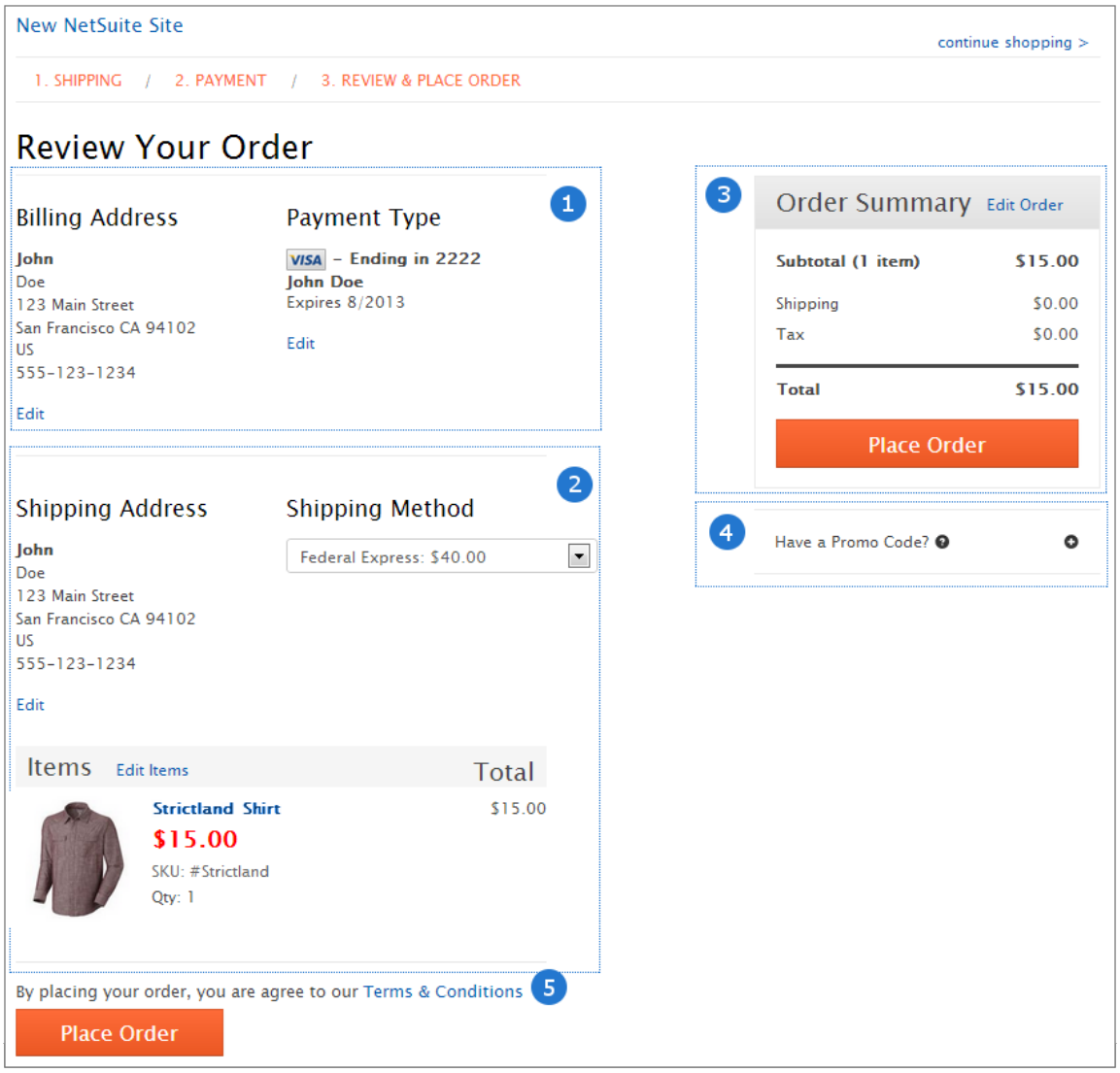
```
save: function()
{
      return this.wizard.model.submit();
}
```

**Note:** Save functions are called from within each module in the sequence but this save function within the Configuration.js file allows you to add final validation or other logic required for your site without having to modify the module code. It is provided for convenience.

- Default Modules:
  - OrderWizard.Module.ShowPayments
  - OrderWizard.Module.ShowShipments
  - OrderWizard.Module.TermsAndConditions
  - OrderWizard.Module.Confirmation
  - OrderWizard.Module.RegisterGuest
  - OrderWizard.Module.ShowPayments
  - OrderWizard.Module.ShowShipments
- Default URLs:
  - review
  - confirmation

## Review Your Order

The Review Your Order page allows customers a final review of the order. Customers can access and edit or change all order components. The Terms and Conditions link is also provided. When the Place Order button is clicked an Email verification is sent to the customer.



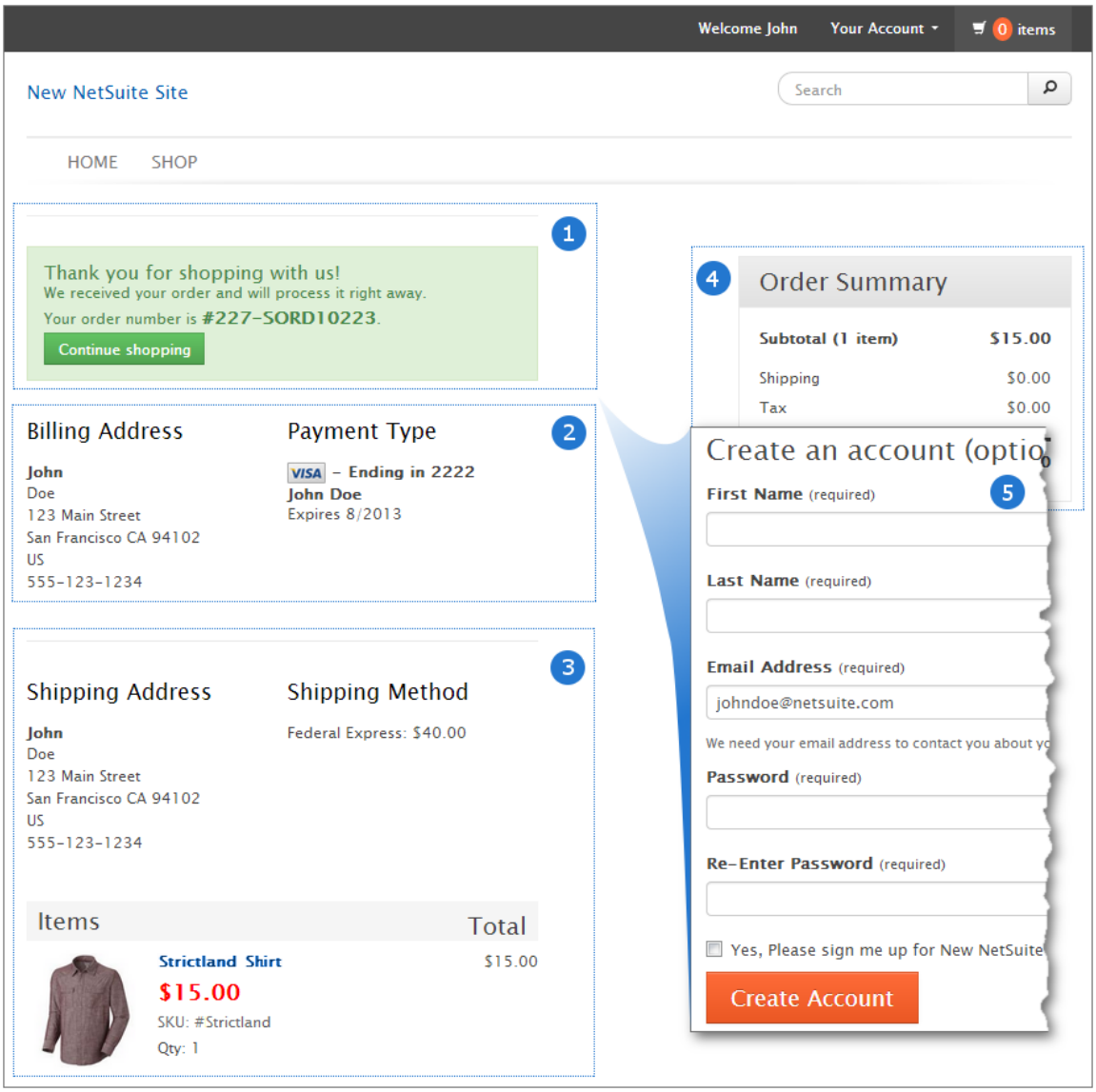

| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. Show Payments | Shows the customer's billing and credit card information in view-only format. Note that the credit card number is obfuscated for security. | order_wizard/order_wizard_showshipments_module.txt | order_wizard_showshipments_module_tmpl |
| 2. Show Shipments | Customers can update their shipping options without leaving the review page. | order_wizard/order_wizard_showshipments_module.txt | order_wizard_showshipments_module_tmpl |
| 3. Order Summary | Displays the summary of the order with tax and shipping information. | shopping_cart\macros\checkout_cart_summary_macro.txt | checkoutCartSummary |

| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 4. Promo Code | If customers enter an invalid promotion code, they are presented with an error message. When a promotion code is successfully applied, the estimated total of the order is updated.<br>The promotion code feature in the account must be enable for this to be displayed. | shopping_cart\macros\promocode_form_macro.txt | promocodeForm |
| 5. Terms and Conditions | This module is rendered only if the site requires agreement to the terms and conditions as defined on the Web Site Setup page.<br>The message rendered when the link is click is also as defined in the Web Site Set Up page. | order_wizard/order_wizard_termsandconditions_module.txt | order_wizard_termsandconditions_module_tmpl |

## Confirmation

The Confirmation page provides a read only display of all order details and the order number.

**Note:** The main site header is used in the confirmation page to allow for easy navigation back to the main website shopping areas.




| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| 1. Confirmation Notice | An order confirmation message is displayed at the top of the page that includes the customer's order number. | order_wizard/ order_wizard_ confirmation_ module.txt | order_wizard_ confirmation_ module_tmpl |
| 2. Show Payments | Shows the customer's billing and credit card information in view-only format. Note that the credit card number is obfuscated for security. | order_wizard/ order_wizard_ showpayments_ module.txt | order_wizard_ showpaymen ts_module_ tmpl |
| 3. Show Shipments | Shows the customer's shipping information in view-only format. | order_wizard/order_ wizard_showshipme nts_module.txt | order_wizard _showshipme nts_module_ tmpl |
| 4. Order Summary | Displays the summary of the order with tax and shipping information. | shopping_cart\ macros\checkout_ | checkoutCartS ummary |

| Element | Description | Template File | Metadata/Tag |
|---|---|---|---|
| | | cart_summary_ macro.txt | |
| 5. Create Account | When shopping as a guest, the option to create an account is displayed during order confirmation.<br>Depending on the site configuration the customer may be redirected to a login page once this form is submitted. | order_wizard/order_ wizard_register_ guest_module.txt | order_wiza rd_register_ guest_ module_tmpl |