



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

Introduction	1
Manage Themes and Extensions	3
Themes and Extensions Checklist	3
Install Theme and Extension SuiteApps	4
Activate Themes and Extensions	4
Developer Tools	7
Developer Tools Checklist	7
Set Up Your Development Environment	8
Install Node.js	8
Install Gulp.js	8
Download Theme and Extension Developer Tools	9
Development Directories Reference	11
Theme Development Files and Folders	11
Extension Development Files and Folders	14
Gulp Command Reference	17
Develop Themes	21
Theme Development Checklist	21
Download Active Theme Files	22
Test and Deploy a Theme	24
Test a Theme on a Local Server	25
Deploy a Theme to NetSuite	26
Develop Your Theme	28
Customize Pre-Existing Theme Files	28
Add a New File to a Theme	29
Override Active Extension Files	32
Best Practices for Creating Themes	34
General Best Practices	34
HTML Templates	34
Sass	36
Assets	37
Theme Manifest	41
Develop Extensions	46
Extension Development Checklist	46
Create an Extension	47
Create a Baseline Extension	48
Create Additional Modules for an Extension	49
Create Custom Content Types for an Extension	49
Download the Active Theme for Extension Development	51
Test and Deploy an Extension	51
Test an Extension on a Local Server	52
Deploy an Extension to NetSuite	53
Extension Manifest	54
API Guide	60
Extensibility API How-To Topics	60
Localize Text in an Extension	60
Access a Component	62
SuiteCommerce Front-end JavaScript APIs	65
BaseComponent	65
CMSComponent	67
CancelableEvents	70
CartComponent	72
ComponentContainer	84
ExtensionEntryPoint	85

EnvironmentComponent	85
ProductDetailsComponent	90
ProductListPageComponent	95
VisualComponent	103
CustomContentTypeBaseView	109
Utils	112
View	116
Deferred	118
SuiteCommerce Back-end JavaScript APIs	121
BackendBaseComponent	121
BackendCancelableEvents	124
BackendCartComponent	126
BackendComponentContainer	141
SCErrors	142
SCEventWrapper	144
SCModel	144
ServiceController	145
Deferred	148
Class	151
Published Documentation Reference	152

Introduction

This PDF is currently in a DRAFT state and is intended for internal use and beta testing only. A complete publication will be available at a future release.

Welcome to SuiteCommerce Standard (SCS). SuiteCommerce Standard lets website administrators and developers create dynamic and engaging ecommerce web stores that interface seamlessly with their NetSuite account. This document provides information on how to develop themes and extensions for a SuiteCommerce Standard site.

You can activate different themes and extensions for one or more domains associated with your NetSuite site. You can also develop your own custom themes and extensions to create a unique stylistic look or additional functionality for any domains associated with your site. SuiteCommerce Standard also supports Site Management Tools, a feature that, if provisioned for your site, lets you manage content on your web site using an intuitive user interface.

Before You Begin

- This document does not provide information required to setup your NetSuite account or install necessary SuiteApps (bundles).
- In some cases, NetSuite does not provide SCS-specific user interface elements. When using NetSuite, follow paths to SCA in these cases. For example, to edit your SCS Web Site record in NetSuite, go to Setup > SuiteCommerce Advanced > Set Up Web Site.
- SuiteCommerce Standard lets you apply themes and extensions to extend the appearance and functionality of your site by domain.

Themes

A **Theme** is a special type of extension that only affects a domain's design, look, and feel. Themes contain any number of HTML templates, Sass files, and other assets bundled into a single SuiteApp. You activate a theme to change the way a web browser presents your site content by extending the HTML and Sass for a specific domain.

You can only activate one theme per domain at a time.

Extensions

An **Extension** introduces added functionality to your website through any number of JavaScript, SuiteScript, configuration JSON, and other files bundled into a single SuiteApp. Extensions can also include HTML and Sass. When you activate an extension for a domain, you extend the JavaScript and SuiteScript to introduce a new feature or function for your site.

You can activate any number of extensions for a domain or choose not to implement any at all.

Master Checklist

Follow these steps when creating themes and extensions for your domain. This checklist identifies the major tasks involved. Each of these topics provides a more detailed list of tasks to help guide you on your way.

Task	Step	Description	Topic
<input type="checkbox"/>	Install and Activate Themes and Extensions	This section explains how to install theme and extension SuiteApps and how to activate themes and extensions for a domain.	Manage Themes and Extensions
<input type="checkbox"/>	Install All Required Developer Tools	This section explains how to install all developer tools necessary to develop themes and extensions.	Developer Tools
<input type="checkbox"/>	Develop Your Theme	This section includes everything you need to know to build your own theme. This includes best practices for developing Sass, HTML, and assets (images, and fonts).	Develop Themes
<input type="checkbox"/>	Develop Your Extension	This section includes everything you need to know to build your own extension.	Develop Extensions

Resources

This guide includes the following sections as reference materials:

- [API Guide](#) – This section provides conceptual information on APIs.
- [Published Documentation Reference](#) – This section provides links to documentation that is published in the NetSuite Help Center. This includes detailed information on building CCTs, understanding the Backbone and MVC framework, how to build JSON configuration files, and how to use Site Management Tools.

Manage Themes and Extensions

After you set up your SuiteCommerce online store, you can extend its standard capabilities using **themes** and **extensions**.

Before You Begin



Important: After activating a theme or extension for the first time, you can only customize your SuiteCommerce Standard site as described in this document. If you have already made any customizations to SuiteCommerce Standard source code prior to activating extensions or themes, any previous customizations will be lost when you initiate your first activation.

Be aware of the following important information:

- You can activate published themes and extensions as bundled SuiteApps or as custom themes and extensions that you create and deploy using the development tools.
- Published themes and extensions are **unmanaged** SuiteApps. With unmanaged SuiteApps, updates are not automatically pushed to installed bundles. You must explicitly update the SuiteApp to receive the latest changes.
- Installing a published theme or extension does not compile run-time files or apply changes to your domains. Installation simply adds the theme or extension source files into your NetSuite File Cabinet. You later activate the theme or extension using the Manage Extensions wizard.
- To access the Manage Extensions wizard, you must first install the SuiteCommerce Extension Management SuiteApp into your NetSuite account.
- You must activate at least a theme when using the Manage Extensions wizard.
- You can activate only one theme per domain, but you can activate the same theme across multiple domains if desired.
- You can activate any number of extensions per domain. Activating extensions is optional.
- You can only process one activation at a time per SSP Application.

Themes and Extensions Checklist

Follow these steps to install and activate themes and extensions for a domain.

Task	Step	Description	Topic
<input type="checkbox"/>	Install Theme and Extension SuiteApps	Themes are available as SuiteApps, which bundle all of the files necessary to preview and implement a theme. You can install any number of themes into your NetSuite account. This step is required before you can activate a theme or extension.	Install Theme and Extension SuiteApps
<input type="checkbox"/>	Activate a Theme	After installing the desired theme into your account, you can activate it for any domain associated with your site. You must have a theme activated and can have only one theme active for a domain at any one time. This step is required.	Activate Themes and Extensions

Task	Step	Description	Topic
<input type="checkbox"/>	Activate an Extension	You activate extensions at the same time you activate a theme for your domain. You can activate one or more extensions for your domain, or you can choose not to activate any at all.	Activate Themes and Extensions
<input type="checkbox"/>	Download Extension Developer Tools	Like themes, you need a set of tools to create your own extensions. These tools are specific for creating extensions. Like the theme developer tools, these are available as a zipped file in the NetSuite File Cabinet. You download these tools and extract them to create a top-level development directory, within which you build your extensions.	Extension Developer Tools

Install Theme and Extension SuiteApps

Before you can activate a published theme or extension, you must install them as bundled SuiteApps into your NetSuite account.



Important: At a minimum, you must install at least one theme SuiteApp. This provides you with a theme to activate using the Manage Extensions wizard later on.

To install a theme or extension SuiteApp:

1. In NetSuite, go to Customization > SuiteBundler > Search & Install Bundles.
2. In the **Keywords** field, enter the Bundle ID or name of the theme or extension you want to install and click **Search**.
3. Verify that the correct theme or extension SuiteApp is returned in the search and select it.
4. Review the SuiteApp details and then click **Install**.

When the SuiteApp installation is complete, you are ready to activate the associated extension or theme. See [Activate Themes and Extensions](#).

To uninstall a theme or extension SuiteApp:

1. Use the Manage Extensions wizard to deactivate the theme or extension related to the SuiteApp you want to uninstall. To deactivate a theme, you must activate a different theme in its place. Repeat this step for all domains currently activating the theme or extension.



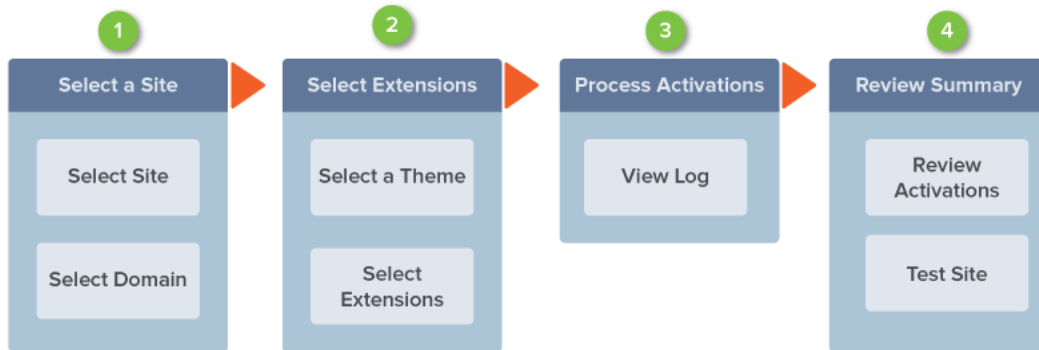
Important: Attempting to delete a SuiteApp associated with an active theme or extension will result in an error.

2. Uninstall the SuiteApp. See the help topic [Uninstalling a Bundle](#) for details.

Activate Themes and Extensions

You can activate themes and extensions for one or more domains associated with your site. Repeat these steps for each domain you want to manage:

- **Step 1: Select a Site** – Select the website and domain to which you want to apply themes and extensions.
- **Step 2: Select Extensions** – Select at least a theme plus any desired extensions for the indicated domain.
- **Step 3: Activate Extensions** – Activate any selected themes and extensions for the indicated domain. This step compiles your domain's runtime files.
- **Step 4: Review Summary** – Review a summary of your activations.



Note: You must have the published theme or extension SuiteApp installed in your account or have deployed a custom theme or extension for it to appear in the Manage Extension wizard.

Step 1: Select a Site

1. In NetSuite, go to Setup > SuiteCommerce Advanced > Extension Management.
2. In the Manage Extensions wizard, choose the site that you want to manage from the **Select Web Site** list.
3. Choose the domain you want to manage from the **Select a Domain** list.
4. Click one of the following:
 - **Cancel** to cancel the activation process and reset the form.
 - **Next>** to select a theme and any optional extensions for the domain.

Step 2: Select Extensions

1. Select the theme you want to activate for your domain from the **Select Theme** list.
2. On the **Extensions Available** subtab, check the **Active** box for any extensions you want to activate.
To deactivate an extension, clear the associated **Active** box.
3. Click one of the following:
 - **Cancel** – to cancel the activation process and to return to Step 1.
 - **< Back** – to go back to Step 1.
 - **Next >** – to activate the theme and any selected extensions.

Step 3: Activate Extensions

The activation process begins immediately with no action required on this page. This step displays the progress of your activations and logs the specific tasks for your reference.

1. Click the Activation Log arrow once to expand the log (optional). Click a second time to collapse. This log displays the processes involved in the activation.
2. When the Activation is 100% complete, click one of the following:
 - **Cancel** – to cancel the activation process and to return to Step 1.
 - **< Back** – to go back to Step 1.
 - **Next >** – to review your activation summary.
3. If this is the initial activation of a theme or extension to a domain, clear the domain's cache by triggering a cache invalidation request. See the help topic [Cache Invalidation](#) for details.

Step 4: Review Summary

1. Review the summary of your activations.
2. Click one of the following:
 - **Cancel** – to go back to Step 1.
 - **< Back** – to go back to Step 1.
 - **Finish** – to close the wizard.
3. Navigate to your domain and confirm that the changes took effect.
If your changes did not take effect, check the following and run the activation process again (if necessary):
 - Clear the domain's cache.
 - Confirm that you are not experiencing any syntax errors in your code.
 - Confirm that no extensions share files of the same name.
 - Check your connection.

Developer Tools

SuiteCommerce Advanced and SuiteCommerce Standard both require different sets of developer tools that let you:

- Deploy custom themes or custom extensions to your NetSuite account. You can also deploy directly to your production or sandbox accounts in NetSuite.
- Retrieve pre-existing theme source code as a basis for building a custom theme.
- Store code locally within a version control system.
- Create and edit code locally in your preferred text editor or IDE.
- Compile the application locally for testing.

This section explains how to build your local environment for theme and extension development. This section also explains the different files, folders, and commands associated with building themes and extensions.

Developer Tools Checklist

Follow these steps to install and use the developer tools required.

Task	Step	Description	Topic
<input type="checkbox"/>	Install Node.js	The first step to setting up your developer environment is install Node.js. This platform is required to run all Gulp.js commands used to build themes and extensions.	Install Node.js
<input type="checkbox"/>	Install Gulp.js	Your next step involves installing the latest version of Gulp.js, an open-source tool that automates many web-based application tasks.	Install Gulp.js
<input type="checkbox"/>	Download Theme Developer Tools	If you are creating your own themes, you must download the Theme Developer Tools. These tools are specific for developing themes and are available as one zipped file in the NetSuite File Cabinet. You download these tools and extract them to create a top-level theme development directory, within which you build your themes.	Theme Developer Tools
<input type="checkbox"/>	Download Extension Developer Tools	If you are creating your own extensions, you must download the Extension Developer Tools. These tools are specific for developing extensions and are available as one zipped file in the NetSuite File Cabinet. You download these tools and extract them to create a top-level extension development directory, within which you build your extensions.	Extension Developer Tools

Developer Tool Resources

This guide includes the following sections as reference materials on the theme and extension developer tools:

- [Development Directories Reference](#) – This section explains the various files and folders located in your theme and extension development directories.
- [Gulp Command Reference](#) – This section explains the Gulp commands used when developing themes and extensions. Each set of tools bears its own set of Gulp commands.

Set Up Your Development Environment

Before you can create custom themes or extensions or make any customizations to any source code, you must first create your local developer environment by following these steps in sequence:

1. [Install Node.js](#)
2. [Install Gulp.js](#)
3. [Download Theme and Extension Developer Tools](#)

Install Node.js

Node.js is the platform required for all Gulp.js tasks described in this section. Node.js is available at:

<https://nodejs.org/en/download/>

Install the version of Node.js that is supported by your implementation. Use the most current minor release for the version of Node.js you are using. The following versions of Node.js are supported:

SuiteCommerce Standard	Supported Node.js Versions
SuiteCommerce Standard	8.9.1 LTS (Long Term Service)

When installing Node.js, ensure that you install all Node.js components, including the Node Package Manager (NPM). The Node Package Manager is used to install Gulp.js and other files required by the developer tools and SuiteCommerce build process.

After running the installer, you should see Node.js in the list of available programs on your system. The `npm` command should also be added to the path on your system.

Note: Ensure that you use the correct installer for your platform. You may have to restart your machine for the Node.js commands to be recognized in the path variable on your system.

To verify that Node.js is installed correctly:

1. Depending on your platform open a command line or terminal window.
2. Enter the following command:

```
node -v
```

If everything is installed correctly, this command outputs the currently installed version of Node.js. There should be no errors or warnings.

Install Gulp.js

Gulp.js is an open-source tool that automates many of the tasks of creating web-based applications. SuiteCommerce sites use Gulp.js to:

- Compile JavaScript files into a single application file that is more efficiently loaded into a browser.

- Compile high-level Sass instructions into CSS.

To Install Gulp.js:

1. Depending on your platform, open a command line or terminal window.
2. Enter the following command:

```
npm install --global gulp
```

Note: Install Gulp.js using the `--global` flag as shown above. On Linux and MacOS, you must run this command using `sudo`. This is required to ensure that Gulp.js is installed correctly.

3. Verify that Gulp.js was installed correctly by entering the following command:

```
gulp -v
```

If installed correctly, this command outputs the currently installed version of Gulp.js. Ensure that this version is 3.9.1 or higher. There should be no errors or warnings.

Permissions

To use Gulp.js to deploy source files to NetSuite, you must use a **System Administrator** role with the following permissions set to **Full**:

- Documents and Files
- Website (External) publisher
- Web Services

Download Theme and Extension Developer Tools

Custom themes and extensions require separate developer tools. These tools are provided for download within your NetSuite account provisioned for SuiteCommerce Standard or SuiteCommerce Advanced.

Depending on your development needs, you might need only one or both of the following tools:

- [Theme Developer Tools](#) – Required to create and edit themes.
- [Extension Developer Tools](#)– Required to create and edit extensions.

Note: The theme and extension developer tools are available as separate downloads from your NetSuite file cabinet. When you extract these tools, you end up two top-level directories for development. One for themes. Another for extensions.

Note: If you experience errors when running these developer tools, see *Troubleshooting the Developer Tools* for assistance.

Theme Developer Tools

Before you can create a custom theme, you must download the theme developer tools and create a top-level (root) development directory. This is where you maintain all theme-related customizations and run Gulp.js commands to fetch files from the server, test your changes locally, deploy custom themes to NetSuite, or publish a bundled theme to the marketplace.

To install theme developer tools:

1. Login to your NetSuite account.
2. In NetSuite, go to Documents > Files > File Cabinet.
3. Navigate to SuiteBundles/Bundle 193237/.
4. Download the .zip file you find there:

```
ThemeDevelopmentTools-18.1.zip
```

5. Extract the .zip file to a location in your local environment. This becomes your root development directory for custom themes.

The .zip file extracts into a directory named **ThemeDevelopmentTools-18.1** by default, but you can rename this directory to suit your needs.



Important: Do not move, delete, or rename any files or folders within the top-level development directory.

6. Open a command line or terminal window.
7. Access your root development directory created previously.
8. Enter the following command to install additional Node.js packages into this directory:

```
npm install
```



Note: This command installs the dependencies required to manage your custom themes. These files are stored in the node_modules subdirectory of the root development directory. This command may take several minutes to complete.

Extension Developer Tools

Before you can create a custom extension, you must download the extension developer tools and create a top-level (root) development directory. This is where you maintain all of your custom extensions and run Gulp.js commands to fetch files from the server, test your changes locally, deploy custom extensions to NetSuite, or publish a bundled extension to the marketplace.

To install extension developer tools:

1. Login to your NetSuite account.
2. In NetSuite, go to Documents > Files > File Cabinet.
3. Navigate to SuiteBundles/Bundle 193237/.
4. Download the .zip file you find there:

```
ExtensionDevelopmentTools-18.1.zip
```

5. Extract the .zip file to a location in your local environment. This becomes your root development directory for your custom extensions.

The .zip file extracts into a directory named **ExtensionDevelopmentTools-18.1** by default, but you can rename this directory to suit your needs.



Important: Do not move, delete, or rename any files or folders within the top-level development directory.

6. Open a command line or terminal window.

7. Access your root development directory created previously.
8. Enter the following command to install additional Node.js packages into this directory:

```
npm install
```

Note: This command installs the dependencies required to manage your custom extensions. These files are stored in the `node_modules` subdirectory of the root development directory. This command may take several minutes to complete.

Development Directories Reference

Theme Development Files and Folders

This section describes the various files and folders included in your top-level theme development directory. You create this when you extract the zip file containing the theme developer tools.

Note: Some of the files and folders listed herein do not appear until you run one or more Gulp.js commands. The following sections point this out where applicable. However, for a full list of Gulp.js commands, see [Gulp Command Reference](#).

The Top-Level Theme Development Directory

The top-level theme development directory contains the following files/folders. Some of the files/folders listed below do not appear until after you run your customizations to a local server or deploy them to NetSuite. You name this top-level directory when you extract the theme development tools.

File/Folder	Description
DeployDistribution/	Created when you run the <code>gulp theme:deploy</code> command, this directory contains all of the files associated with the compiled application. After compilation, Gulp.js deploys the contents of this directory to your NetSuite file cabinet. Do not manually edit the files in this directory.
gulp/	This directory contains all of the files required by Gulp.js. Do not manually edit the files in this directory.
LocalDistribution/	Created when you run the <code>gulp theme:local</code> command, this directory contains all of the files associated with the compiled application used by the local server. When you run <code>gulp theme:local</code> , Gulp.js deploys the contents of this directory to the local Node.js server. Do not manually edit the files in this directory.
node_modules	Created when you run the <code>npm install</code> command, this directory stores the dependencies and other files required by the development tools. Do not manually edit this file.
ns_npm_repository	Do not manually edit this file.
Workspace/	Created when you initially run the <code>gulp theme:fetch</code> command, this directory maintains all downloaded and customized theme and extension HTML, Sass, and asset files. This is the directory where you maintain all of your theme and extension customizations.

	See The Themes Workspace Directory for detailed information on the contents of this directory.
.jshintrc	Do not manually edit this file.
distro.json	Do not manually edit this file.
gulpfile.js	This file contains all the JavaScript code necessary to run Gulp.js. Do not manually edit this file.
javascript-libs.js	Do not manually edit this file.
package.json	This file maintains dependencies required to operate the theme development tools. Do not manually edit this file.
version.txt	This file maintains versioning information for SuiteCommerce Standard. Do not manually edit this file.

The Themes Workspace Directory

The Workspace directory is the location within your top-level theme development directory where you create themes. This directory contains a subdirectory for the downloaded theme and an Extras directory to maintain all extension-related source files. When you run the `gulp theme:fetch` command, the development tools delete all Workspace directory subfolders and download the active theme and extension source files, building a new Workspace environment each time.

```
<TopLevelDevelopmentDirectory>/
  Workspace/
    Extras/
      <THEME_DIRECTORY>/
```

The Theme Directory

When you access your theme development directory and run the `gulp theme:fetch` command, the development tools create a subdirectory to store the source files for the active theme. The name of this directory matches the name of the theme that is active when you run the `gulp theme:fetch` command. Perform all theme customizations and template overrides here.

The theme directory contains the following files or folders:

File/Folder	Description
assets/	This directory maintains any images or fonts associated with the theme. These assets include fonts, logos, icons, and other images related to your site that are not managed by NetSuite. This is also the location where you save any new assets you introduce as part of your theme customizations or extension overrides.
Modules/	This directory contains individual modules as subdirectories associated with the theme. Each module defines a specific area of functionality (feature or utility) for your site and contains Template and Sass files in respective subdirectories. You customize these Sass and HTML files directly.
Overrides/	This directory contains any HTML and Sass associated with all extensions that were active when you ran the <code>gulp theme:fetch</code> command. This directory is initially empty, but its structure matches that of the files in the Extras directory. If you are customizing HTML and Sass files associated with an extension, you place copies of those files here to maintain your overrides.

	See Override Active Extension Files for more details.
manifest.json	This file maintains all extensible resources for the theme and declares any overrides for extension customizations. The development tools automatically edit this file to include any necessary overrides when you run the <code>gulp theme:deploy</code> command. For more information on this file, see <i>Manifest Files</i> .

Example

Your domain has an active theme, **Theme1**. You run the `gulp theme:fetch` command and specify your domain. The development tools clear any existing Workspace directory contents and download all theme-related HTML, Sass, and asset files into the Theme1/ directory. This is the location where you create or edit your theme. If the theme you download includes any previously deployed overrides, the development tools place them in the Overrides directory.

In this example, your Workspace directory structure should look similar to following:

```
<TopLevelDevelopmentDirectory>/
  Workspace/
    Theme1/
      assets/
        img/
        fonts/
      Modules/
        AddressModule@1.0.0/
          Sass/
          Templates/
      Overrides/
      manifest.json
```

The Extras Directory (Theme Development)

When you access your theme development directory, you The `../Workspace/Extras/Extensions` directory contains subdirectories for each extension active when downloaded using the `gulp theme:fetch` command. Each extension subdirectory contains the following files or folders:



Important: At this time, you can only create your own themes. Therefore, do not manually edit or remove files found in the Extras/Extensions directory. You can only customize extension-related HTML and Sass files using the Override method. See [Override Active Extension Files](#).

File/Folder	Description
assets/	This directory maintains any images or fonts associated with the associated extension. These assets include fonts, logos, icons, and other images related to your site that are not managed by NetSuite.
Modules/	This directory contains a module folder that maintains all HTML templates and Sass associated with the extension. These files are provided here for your reference only.
manifest.json	This file lists all extension-related JavaScript, SSP libraries, configuration, HTML templates, Sass, and assets related to the active extensions downloaded when you ran the <code>gulp theme:fetch</code> command. For more information on this file, see Theme Manifest . Do not manually edit this file.

If your domain does not have any active extensions when you run the `gulp theme:fetch` command, or an active extension does not contain any Sass, HTML, or assets, the development tools do not create a folder for that extension.

Note: The Extras subdirectory also contains an `application_manifest.json` file. This file confirms that you have a valid SSP Application version that supports the Themes and Extensions. Do not move, delete, or edit this file.

Extension Development Files and Folders

This section describes the various files and folders included in your top-level extensions development directory. You create this when you extract the zip file containing the extension developer tools.

Note: Some of the files and folders listed herein do not appear until you run one or more Gulp.js commands. The following sections point this out where applicable. However, for a full list of Gulp.js commands, see [Gulp Command Reference](#).

The Top-Level Extension Development Directory

Your top-level extensions development directory contains the following files/folders. Some of the files/folders listed below do not appear until after you run your customizations to a local server or deploy them to NetSuite. You name this top-level directory when you extract the theme development tools.

File/Folder	Description
DeployDistribution/	Created the first time you run the <code>gulp extension:deploy</code> command, this directory contains all of the files associated with the compiled application. After compilation, Gulp.js deploys the contents of this directory to your NetSuite file cabinet. Do not manually edit the files in this directory.
gulp/	Created when you extract the extension developer tools, this directory contains all of the files required by Gulp.js. Do not manually edit the files in this directory.
LocalDistribution/	Created the first time you run the <code>gulp extension:local</code> command, this directory contains all of the files associated with the compiled application used by the local server. When you run <code>gulp extension:local</code> , Gulp.js deploys the contents of this directory to the local Node.js server. Do not manually edit the files in this directory.
node_modules	Created when you run the <code>npm install</code> command, this directory stores the dependencies and other files required by the development tools. Do not manually edit this file.
ns_npm_repository	Created when you install the extension developer tools, this folder contains important files for the NPM package manager. Do not manually edit this file.
Workspace/	Created the first time you run the <code>gulp extension:fetch</code> or <code>gulp extension:create</code> command, this directory maintains all of your extension files under development. This directory also includes an Extras/ folder to maintain theme files for local testing. See The Extensions Workspace Directory for detailed information on the contents of this directory.
.jshintrc	Do not manually edit this file.

distro.json	Do not manually edit this file.
gulpfile.js	This file contains all the JavaScript code necessary to run Gulp.js. Do not manually edit this file.
javascript-libs.js	Do not manually edit this file.
package.json	This file maintains dependencies required to operate the theme development tools. Do not manually edit this file.
version.txt	This file maintains versioning information for SuiteCommerce Standard. Do not manually edit this file.

The Extensions Workspace Directory

The Workspace directory resides in your top-level extensions development directory. This directory contains a subdirectory for each extension you are creating plus an Extras directory to store the active theme source files.

```
<Top-LevelDevelopmentDirectory>/
  Workspace/
    Extras/
      <EXTENSION_DIRECTORY>/
```

The Extension Directory

When you run the `gulp extension:create` command, the developer tools create the Workspace directory (unless it already exists) and places a basic extension with source files for you to begin creating your own extension.

Each extension directory contains the following files or folders:

File/Folder	Description
assets/	This directory maintains any images or fonts associated with the extension. These assets include fonts, logos, icons, and other images related to your site that are not managed by NetSuite.
Modules/	This directory contains individual modules as subdirectories associated with the extension. Each module defines a specific area of functionality (feature or utility) for your site and contains the JavaScript, SuiteScript, Configuration, Templates, and Sass for your extension.
manifest.json	This file maintains all extensible resources for the extension. The development tools automatically edit this file to include any necessary overrides when you run the <code>gulp extension:deploy</code> command. For more information on this file, see Extension Manifest .

Example

You run the `gulp extension:create` command from your top-level extension development directory. During this task, you name your extension **MyCoolExtension** and choose to create all options for extension files (templates, sass, configuration, etc.). You also choose not to create a CCT.

After this command is complete, your Workspace directory contains the following:

```
<TopLevelDevelopmentDirectory>/
```

```

Workspace/
  MyCoolExtension/
    assets/
      fonts/
      img/
      services/
        MyCoolModule.Service.ss
    Modules/
      MyCoolModule/
        Configuration/
        JavaScript/
        Sass/
        SuiteScript/
        Templates/
      manifest.json

```

The MyCoolModule directory contains an example file for each files type. You use these files as an example to build your new extension. For example, the JavaScript folder contains JavaScript collection, model, router, view, and entry point files.

The Extras Directory

When you run the `gulp extension:fetch` command, the developer tools create the Workspace directory (unless it already exists) and downloads the source files for the active theme. These files are provided for reference during local testing only. Do not move, delete, add, or edit the files in the Extras subdirectory.

The Extras subdirectory contains the following files or folders:

File/Folder	Description
assets/	This directory maintains any images or fonts associated with the active theme. These assets include fonts, logos, icons, and other images related to your site that are not managed by NetSuite.
Modules/	This directory contains a module folder that maintains all HTML templates and Sass associated with the theme. These files are provided here for your reference only.
manifest.json	This file lists all extension-related HTML templates, Sass, and assets related to the active theme downloaded when you ran the <code>gulp extension:fetch</code> command. Do not manually edit this file.

Note: The Extras subdirectory also contains an `application_manifest.json` file. This file confirms that you have a valid SSP Application version that supports the Themes and Extensions. Do not move, delete, or edit this file.

Example

You run the `gulp extension:fetch` command.

Your domain has an active theme, **ActiveTheme1**, and an active extension, **MyCoolExtension**. You run the `gulp theme:fetch` command and specify your domain. The extension developer tools download all theme-related HTML, Sass, and asset files into the `Exrtras/ActiveTheme1/` directory.

In this example, your Workspace directory structure should look similar to following:

```

<TopLevelDevelopmentDirectory>/
  Workspace/
    Extras/
      ActiveTheme1/
        assets/
        Modules/
        Overrides/
        manifest.json
      MyCoolExtension/

```

Gulp Command Reference

The following table lists Gulp.js commands required when using the theme and extension developer tools with SuiteCommerce Standard and SuiteCommerce Advanced.




Theme Developer Gulp Commands

Command	Description
gulp theme:fetch	This command downloads the active theme and extension files (Sass, HTML, and other assets) for the specified domain. You must have already activated a theme using the Manage Extensions wizard in NetSuite for this command to run. This command places these files in the top-level directory's Theme directory. If this is the first time running this command, the development tools create subdirectories for these files.
gulp theme:fetch --m <arg>	This command downloads the active theme and extension files (Sass, HTML, and other assets) for the specified sandbox or testing account, where <arg> is the name of the account. For example: gulp theme:fetch --m sandbox.
gulp theme:fetch --to	This command fetches a theme and extensions from NetSuite as usual, but it prompts for login credentials, ignoring the .nsdeploy file.
gulp theme:local	This command compiles all Sass and HTML template files for a theme into a functional application. This command includes any theme overrides to associated extensions. After compilation, this command starts a local server. This server watches for changes to the SuiteCommerce Standard Sass and HTML template files. After the server starts, any changes you make to your theme files or extension overrides are automatically recompiled and visible in the browser.
gulp theme:deploy	This command compiles Sass, HTML, and asset files into a DeployDistribution folder. If you are deploying changes to a published theme, the development tools prompt you for a Vendor Name, Theme Name, Theme Version, Description, and Application when you initially run this command. This command forces you to name a new theme. If you are deploying customizations to a custom theme, this command does not prompt you for this information and gives you the option to create a new theme or update the existing theme. The development tools then create a folder for the theme in the NetSuite file cabinet and deploy the customized theme's code.

Command	Description
	In addition to compiling the application, this command creates the <code>.nsdeploy</code> file, if it does not already exist.
<code>gulp theme:deploy --advanced</code>	This command deploys as an update of the current custom theme, but resets the prompts regarding the Vendor Name, Theme Name, Theme Version, Description, and Application. This command also rewrites this information in the theme's <code>manifest.json</code> file. This command only takes effect on custom themes.
<code>gulp theme:deploy --create</code>	This command creates a new theme instead of updating the existing theme.
<code>gulp theme:deploy --m <arg></code>	This command deploys the active theme and extension files (Sass, HTML, and other assets) to the specified sandbox or testing account, where <code><arg></code> is the name of the account. For example: <code>gulp theme:deploy --m sandbox.</code>
<code>gulp theme:deploy --skip-compilation</code>	This command deploys the current contents of the <code>DeployDistribution</code> folder without compiling the application.
<code>gulp them:deploy --source templates</code>	This command only compiles and deploys the HTML template files.
<code>gulp theme:deploy --source sass</code>	This command only compiles and deploys Sass files.
<code>gulp theme:deploy --source assets</code>	This command only compiles and deploys theme asset files.
<code>gulp extension:deploy --source <multiple></code>	Separate multiple source deployments by commas to deploy more than one type of source code. For example, <code>gulp extension:deploy --source templates,sass</code>
<code>gulp theme:deploy --to</code>	This command deploys to NetSuite as usual, but resets the login credentials, rewriting the <code>.nsdeploy</code> file.
<code>gulp theme:local styleguide</code>	This command compiles your Sass, parses all KSS blocks declared in the Sass files, and creates a style guide accessible in your localhost (<code>localhost:3000/</code>). See the help topic Style Guide for more information.
<code>gulp theme:update-manifest</code>	This command updates the theme's <code>manifest.json</code> file without requiring a deployment.
<code>gulp validate</code>	This command validates the theme's <code>manifest.json</code> file and confirms that the file does not list any files that do not exist in the theme folder.
<code>gulp clear</code>	This command removes the <code>DeployDistribution</code> and <code>LocalDistribution</code> directories and the <code>.nsdeploy</code> file.
<code>gulp</code>	This command displays a list of all gulp commands.

Extension Developer Gulp Commands

Command	Description
<code>gulp extension:create</code>	This command creates an example extension (with one example module) to use as a baseline for development. This command creates all necessary folders to store and maintain your

Command	Description
	customizations within your top-level extension development directory.
gulp extension:create-module	This command creates an additional module within your extension. This command prompts you for information about the module you want to create and the extension within which you want to create it.
gulp extension:fetch	<p>This command downloads the active theme and compiles all theme resources (Sass, HTML, and other assets). This command places theme files in the top-level extension development directory's Extras folder. If this is the first time running this command, the development tools create subdirectories for these files.</p> <div>  Note: Any downloaded theme files are only provided for testing your extensions locally. You do not customize any theme files in your top-level extension development directory. </div> <p>If you also choose to continue development on a previously deployed, custom extension, this command also downloads these files, placing them in your Workspace/<EXTENSION_FOLDER>. The extensions must be active using the Manage Extensions wizard in NetSuite.</p> <div>  Note: This command does not download published extensions. You cannot customize published content. </div>
gulp extension:fetch <arg>	<p>This command downloads the files of a specific extension, where <arg> is the name of the extension. This can be helpful if you want to use a previously deployed extension as a baseline for a new one. You can fetch multiple extensions, separated by a comma. For example: <code>gulp extension:fetch --fetch Badges, CartExtension</code>.</p> <div>  Note: Extensions must be active to download code or to view them on a local server. </div>
gulp extension:local	This command compiles your custom extension files into a functional application and places them in a LocalDistribution folder. After compilation, this command starts a local server. This server watches for changes to any extension files. After the server starts, any changes you make to your extension files are automatically recompiled and visible in the browser.
gulp extension:deploy	<p>This command compiles your custom extension files into a functional application and places them into a DeployDistribution folder.</p> <p>If you have more than one extension in your top-level development directory, this command prompts you to declare which extension to deploy.</p> <p>If you have never deployed the extension, this command prompts you for information about the extension.</p>

Command	Description
	The development tools then create a folder for the extension in the NetSuite file cabinet and deploy the customized extension code. In addition to compiling the application, this command creates the <code>.nsdeploy</code> file, if it does not already exist.
<code>gulp extension:deploy --advanced</code>	This command deploys an update of the extension, but resets the prompts regarding the Vendor Name, Extension Name, Version, Description, Application, etc. This command also rewrites this information in the extension manifest.json file.
<code>gulp extension:deploy --m <arg></code>	This command deploys your extension files to the specified sandbox or testing account, where <code><arg></code> is the name of the account. For example: <code>gulp extension:deploy --m sandbox</code> .
<code>gulp extension:deploy --skip-compilation</code>	This command deploys the current contents of the <code>DeployDistribution</code> folder without compiling the application.
<code>gulp extension:deploy --source configuration</code>	This command only compiles and deploys configuration JSON files.
<code>gulp extension:deploy --source javascript</code>	This command only compiles and deploys JavaScript files.
<code>gulp extension:deploy --source ssp-libraries</code>	This command only compiles and deploys ssp-libraries.
<code>gulp extension:deploy --source services</code>	This command only compiles and deploys services.
<code>gulp extension:deploy --source sass</code>	This command only compiles and deploys Sass files.
<code>gulp extension:deploy --source templates</code>	This command only compiles and deploys HTML files.
<code>gulp extension:deploy --source assets</code>	This command only compiles and deploys assets.
<code>gulp extension:deploy --source <multiple></code>	Separate multiple source deployments by commas to deploy more than one type of source code. For example, <code>gulp extension:deploy --source templates,sass</code>
<code>gulp extension:update-manifest</code>	This command updates the extension's manifest.json file without requiring a deployment.
<code>gulp validate</code>	This command validates the theme's manifest.json file and confirms that the file does not list any files that do not exist in the theme folder.
<code>gulp clear</code>	This command removes the <code>DeployDistribution</code> and <code>LocalDistribution</code> directories and the <code>.nsdeploy</code> file.
<code>gulp</code>	This command displays a list of all gulp commands.

Develop Themes

Theme Development Checklist

Follow these steps to develop, test, and deploy your own theme.

Task	Step	Description	Topic
<input type="checkbox"/>	Activate the Theme You Want to Customize	To create a theme, the best practice is to use an existing theme as a baseline for development. This can be the SuiteCommerce Base Theme or any published or custom theme available in your account. You must activate the theme you intend to use as a baseline before you can download files and begin development. This process uses the Extensions Management wizard to activate themes and extensions.	Activate Themes and Extensions
<input type="checkbox"/>	Activate Extensions You Want to Override	When you develop a theme, your changes to the theme's Sass and HTML do not affect any active extensions. However, you can override any extension's HTML and Sass files to accommodate your theme and deploy both to a specific domain. Overriding extension files does not introduce changes to published content. Instead, you deploy compiled CSS and HTML files for any active extensions used with your theme.	Activate Themes and Extensions
<input type="checkbox"/>	Download the Active Theme and Extensions	After you have activated a theme and any extensions for a domain, you now download all editable files to your local development directory and begin development. Theme files are provided to edit directly. Extension files are provided as a reference to override if necessary.	Download Active Theme Files
<input type="checkbox"/>	Customize Pre-existing Theme Files	You are now ready to start development. These instructions describe how to edit existing theme Sass and HTML files.	Customize Pre-Existing Theme Files
<input type="checkbox"/>	Create New Theme Files	When you customize a theme, you can also create new HTML, Sass, or asset files to meet your needs. This section explains how to do this.	Add a New File to a Theme
<input type="checkbox"/>	Override Extension Files	If your new theme creates a situation where an active extension no longer matches the style your theme provides, you can override the Sass and HTML of the active extensions to match your theme. These procedures explain how to introduce HTML and Sass changes to a deployed extension using the override method.	Override Active Extension Files

Task	Step	Description	Topic
<input type="checkbox"/>	Test Your Theme Locally	Follow these instructions to test your theme, plus any extension overrides, in a local environment.	Test a Theme on a Local Server
<input type="checkbox"/>	Deploy Your Theme to NetSuite	Follow these instructions to deploy your theme files and extension overrides to your NetSuite Account.	Deploy a Theme to NetSuite
<input type="checkbox"/>	Activate Your Theme and any Necessary Extensions	After you deploy your theme files and extension overrides, you must now activate the deployed theme and extensions before your changes take effect on your domain.	Activate Themes and Extensions

Theme Development Resources

Read the following sections for reference materials for developing themes:

- [Best Practices for Creating Themes](#) – This section provides helpful practices to consider when developing Sass and HTML for your themes.
- [Theme Manifest](#) – This section explains the theme's manifest.json file.

Download Active Theme Files

To create a theme, you first download files for the currently active theme to use as the baseline for your own custom theme. You can download files of any published theme or any previously deployed custom theme.

When you run the `gulp theme:fetch` command, the theme developer tools download all editable theme-related files for the active theme. If you have any active extensions when you run the `gulp theme:fetch` command, the Sass, HTML, and assets of those extension download to your local environment as well. These are provided should you need to make changes to the extensions to match your new theme.

To download the active theme and extension files:

Note: You must at least have a theme activated for a domain before continuing with this procedure. If you specify a domain with any active extensions, you will also download any editable extension files.

1. Open a command line or terminal.
2. Access the top-level theme development directory you created when you downloaded the developer tools.
3. Enter the following command:

```
gulp theme:fetch
```

4. When prompted, enter the following information:
 - Email – Enter the email address associated with your NetSuite account.
 - Password – Enter your account password.

- NetSuite account – Select the NetSuite account where SuiteCommerce Standard is installed.
- Website – Select your website with the domain you are customizing.
- Domain – Select your domain with the active themes and extensions you want to download.

What Happens When You Download an Active Theme?

When you run the `gulp theme:fetch` command, the theme development tools:

- Create the Workspace directory in your top-level development directory. If the Workspace directory already exists, the theme development tools clear its contents before downloading.
- Download all theme-related HTML and Sass files for the theme by module to `../Workspace/<THEME_DIRECTORY>/Modules`. The theme directory is intuitively named to match the name of the active theme.
- Download all theme-related assets to `../Workspace/<THEME_DIRECTORY>/assets`.
- Validate that all files declared in the theme's `manifest.json` have been downloaded correctly. If any problems occurred, the development tools list all missing files and direct you to execute the `gulp theme:fetch` again.

If your domain includes any active extensions when you run the `gulp theme:fetch` command, the theme developer tools:

- Download all extension-related HTML and Sass files by module to `../Workspace/Extras/Extensions/<EXTENSION_DIRECTORY>/Modules`. The extension directories are intuitively named to match the name of each active extension.
- Download all extension-related assets by extensions to `../Workspace/Extras/Extensions/<EXTENSION_DIRECTORY>/assets`.
- If you are downloading a custom theme that includes previously deployed overrides, the theme development tools download these into `../Workspace/<THEME_DIRECTORY>/Overrides`.

Example

Your domain has an active theme, **PublishedTheme1**, and an active extension, **PublishedExtension1**. You run the `gulp theme:fetch` command and specify your domain.

In this example, your Workspace directory structure should look similar to following:

```
<TopLevelDevelopmentDirectory>/
  Workspace/
    Extras/
      Extensions/
        PublishedExtension1/
          assets/
          Modules/
          manifest.json
          application_manifest.json
        PublishedTheme1/
          assets/
          Modules/
          Overrides/
          manifest.json
```

When the gulp processes are complete:

- All **theme-related** HTML and Sass files are available for direct customization here (sorted by module):

```
TopLevelDirectory/Workspace/PublishedTheme1/Modules/
```

- All **theme-related** assets are available here:

```
TopLevelDirectory/Workspace/PublishedTheme1/assets/
```

- Any **extension-related** HTML and Sass source files are available for override here (sorted by module):

```
TopLevelDirectory/Workspace/Extras/Extensions/PublishedExtension1/Modules/
```

- All **extension-related** assets are available here:

```
TopLevelDirectory/Workspace/Extras/Extensions/PublishedExtension1/assets/
```

- Any **pre-existing** extension overrides are available here:

```
TopLevelDirectory/Workspace/PublishedTheme1/Overrides/
```

Test and Deploy a Theme

After customizing your theme and extension files, the next step is to test and deploy. Use the theme development tools to either test your customizations on a local server or deploy them to your NetSuite account.

This section explains how to:

- [Test a Theme on a Local Server](#)
- [Deploy a Theme to NetSuite](#)

Before You Begin

Be aware of the following important information:

- The `gulp theme:deploy` command checks to see if the theme you have customized is a published theme or a previously deployed, custom theme.
 - When you deploy customizations to a **published** theme, the theme development tools require that you create a new, custom theme. You cannot overwrite published content.
 - When you deploy customizations to a pre-existing **custom** theme, the theme development tools give you the option to create a new theme or update the existing theme with a new revision.
- When you create a new custom theme, the theme development tools rename the local directory where you created your customizations. This name matches the theme name you specify when you deploy your code.

Example: When you download the files for the **SCS Example** theme, the theme development tools create the `SCS_Example_Theme` directory. You make your customizations and deploy to NetSuite. The theme development tools prompt you to create a new theme, which you name **MyTheme1**. When the deployment is complete, your local theme directory is renamed to **MyTheme1**.

- All theme-related changes and extension-related overrides deploy together as part of a custom theme and are independent of any published content or other deployed themes.
- During activation, if you decide not activate an extension for which you created an override, that override does not take effect at runtime and has no impact on your domain.
- If you deploy a custom theme with extension overrides and later activate new extensions for the same domain. Your deployed customizations will not apply to the new extensions. You must update your theme to include customizations for the new extensions.
- After deploying a theme or extension to NetSuite, you must activate the theme using the Manage Extensions wizard to apply your changes to a domain.

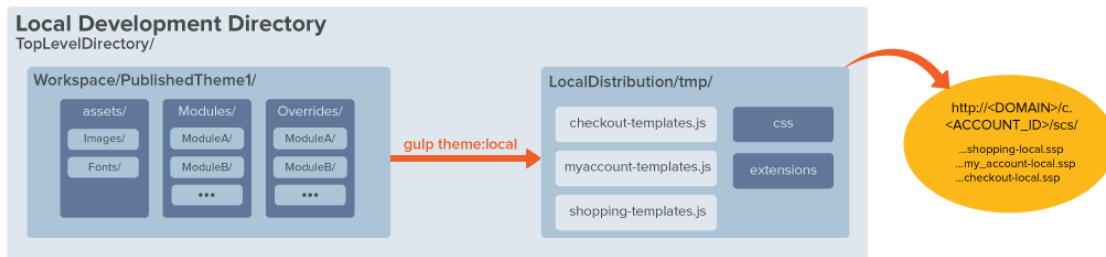
Test a Theme on a Local Server

You can test your theme customizations on a local server before deploying to NetSuite. The local server is installed as part of the theme development tools installation.

To set up your files for a local server, access the top-level directory in a command terminal and run the following command:

```
gulp theme:local
```

This command compiles all source files, including your theme and any extensions customizations, into combined files within a LocalDistribution/tmp directory. This directory contains the Sass and template files used by the local version of the application, so you can test your site before deploying anything to NetSuite.



To run your Theme and Extension customizations on a local server:

1. In your local developer environment, open a command line or terminal and access the top-level development directory.
2. Run the following command:

```
gulp theme:local
```

3. Navigate to the local version of the application using one of the following URLs:

- `http://<DOMAIN_NAME>/c.<ACCOUNT_ID>/scs/shopping-local.ssp`
- `http://<DOMAIN_NAME>/c.<ACCOUNT_ID>/scs/my_account-local.ssp`
- `http://<DOMAIN_NAME>/c.<ACCOUNT_ID>/scs/checkout-local.ssp`

In the above URL patterns, you must replace the following variables with values for your specific environment:

- **DOMAIN_NAME** – replace this value with the domain name configured for your NetSuite website record.

- ACCOUNT_ID – replace this value with your NetSuite account ID.

The server starts automatically. With the local server running, you can make changes to your local files and see the results immediately. Gulp.js automatically recompiles the application when you save any changes to theme-related HTML, Sass, or assets. Simply refresh your browser to see the changes.



Important: If you add a new file or make changes to any overrides after launching the local server, you must run the `gulp theme:local` command again to include the new changes. Gulp.js does not automatically compile new files or process overrides.

Deploy a Theme to NetSuite

The term **deploy** refers to what happens when you use Gulp.js to upload source files and any custom changes to a hosted site. To do this, you use the following command:

```
gulp theme:deploy
```

This command validates your customizations, copies them into a local DeployDistribution directory, and updates the manifest files with any necessary overrides. Gulp.js then uploads these files to your NetSuite account, making them available for activation.

For more Gulp.js commands used with themes and extensions, see [Gulp Command Reference](#).

To deploy your customizations to NetSuite:

1. In your local developer environment, open a command line or terminal and access the top-level development directory.
2. Run the following command:

```
gulp theme:deploy
```

3. When prompted, enter the following information:
 - Email – Enter the email address associated with your NetSuite account.
 - Password – Enter your account password.
 - Vendor – Enter the name of the vendor building this theme.
 - Theme Name – Enter a name for your custom theme.
 - Theme Version – Enter a version for your theme. For example: 1.0.0.
 - Supported Products – Select the product or products (SCA, SCIS, and SCS) to which you want to deploy. An asterisk (*) identifies a selected product.
 - Use the spacebar key to select or deselect a product.
 - Use the UP and DOWN arrow keys to scroll through the product list.
 - Toggle the A key to select or deselect all products.



Note: Theme deployment parameters (Theme Name, Vendor, Theme Version, etc.) are stored in the theme's manifest.json file. Login credentials (email, password, etc.) are stored in the .nsdeploy file. See [Gulp Command Reference](#) for more information on how to re-deploy your custom theme and reset these parameters.

4. In NetSuite, go to Setup > SuiteCommerce Advanced > Extensions Management.

5. Activate your new theme for the domain or domains of your choice. See [Activate Themes and Extensions](#) for instructions.



Important: Your customizations do not apply to your site until you activate the theme for a specific domain using the Manage Extensions wizard in NetSuite. The deploy process includes a notation reminding you of the domain and theme name to set during this process.

What Happens When You Deploy a Custom Theme?

The deployment process is specific to uploading custom themes and extension overrides (if applicable) to a location in your NetSuite File Cabinet. This process does not compile any files or associate any files with a site or domain. To do that, you must activate a theme and extensions.

When you download a theme or extension's source files using the `gulp theme:fetch` command, you receive all HTML, Sass, overrides, and assets (images, fonts, etc.) of the active theme. This command places these files in your `Workspace/<THEME_DIRECTORY>` folder. This is where you build your custom themes and extension overrides.

During the deployment process:

- The development tools copy all of your custom theme development files (modules, assets, and overrides) into the `DeployDistribution` folder in your top-level development directory. If this directory does not exist, the development tools create it.
- The theme development tools validate your customizations.
- The theme development tools update the `manifest.json` file to include any overrides.
- The theme development tools upload these files to your NetSuite file cabinet as bundled files (maintaining the same organizational structure).
- NetSuite creates an Extension record for the custom theme. Like a published theme or extension, these files have not compiled into usable files, but they are available for activation.



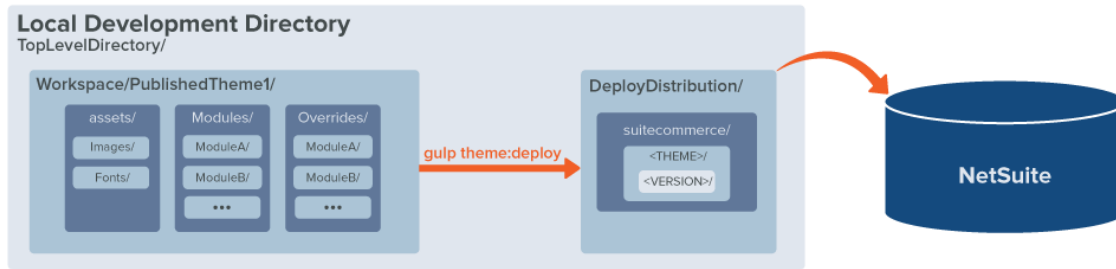
Note: Your extension overrides always deploy as part of a custom theme. If you activate a custom theme but fail to activate any extension for which the theme includes overrides, the application will ignore the overrides and your site will function normally.

Example

You download the source files for the active theme (**PublishedTheme1**) and the active extension (**PublishedExtension1**). You follow best practices and procedures to customize your theme files and override your extension files. After testing on a local server, you decide to deploy your customizations.

You run the `gulp theme:deploy` command, and the theme development tools update the manifest file with any overrides, validate your customizations, and copy the contents of your theme directory to your `DeployDistribution` directory. The development tools then upload your development files to a location in your NetSuite File Cabinet. These files are not compiled into any usable runtime files, but they are now available for activation on a site and domain.

When you run the `gulp theme:deploy` command, the development tools prompt you for a new theme name, which you call **MyCustomTheme1**. Finally, the development tools rename your `Workspace/PublishedTheme1` directory to `Workspace/MyCustomTheme1` and update the manifest and configuration file with the new theme name. This prepares the local environment to function with the new name.



Develop Your Theme

After downloading theme and extension source files to your local developer environment, you are ready to create your own custom themes.

This section explains how to:

- [Customize Pre-Existing Theme Files](#)
- [Add a New File to a Theme](#)
- [Override Active Extension Files](#)

Before You Begin

Be aware of the following important information:

- The examples presented in this section describe customizing Sass files. However, you can customize HTML files as well to suit your needs. You can also introduce new images and fonts as assets.
- You can customize theme-related files directly within your Workspace/<THEME_DIRECTORY> folder.
- You must use the Override method to customize extension-related HTML and Sass files.
- All theme and extension customizations deploy together and remain as part of one custom theme. These customizations take effect when you activate the theme and extensions for a domain.
- When referencing assets (images and fonts) in your theme customizations and extension overrides, use the helpers provided in this section to maintain dynamic paths.



Important: To develop a theme, you must have experience working with HTML, CSS, and Handlebars.js. Before customizing any theme or extension source code, read and understand the [Best Practices for Creating Themes](#).

Customize Pre-Existing Theme Files

When you download a theme's source files, you have access to the HTML, Sass, and assets used by that theme. You can customize the HTML and Sass files directly within your Workspace directory.

To customize a pre-existing theme file:

1. Open the theme directory in your top-level development directory.

For example:

```
Top-LevelDevelopmentDirectory/Workspace/<THEME_DIRECTORY>/
```

2. Locate the subdirectory containing the file or files you want to customize.

For example, you want to customize the top margin of the **Add to Cart** button for this theme. You locate the `_cart-add-to-cart-button.scss` file in the theme's Cart module:

```
.../<THEME_DIRECTORY>/Modules/Cart@1.0.0/Sass/_cart-add-to-cart-button.scss
```

3. Edit the HTML or Sass file within the associated module.

You can edit this file directly. See [Best Practices for Creating Themes](#) for details. In this example, you edit the `.cart-add-to-cart-button-button` class and change the `margin-top` value from `lv3` to `lv5`.

```
//...
// New code with new margin-top value:
.cart-add-to-cart-button-button{
  @extend .button-primary;
  @extend .button-large;
  width: 100%;
  font-weight: $sc-font-weight-semibold;
  margin-top: $sc-margin-lv5;
}

/* Previous Code:
.cart-add-to-cart-button-button{
  @extend .button-primary;
  @extend .button-large;
  width: 100%;
  font-weight: $sc-font-weight-semibold;
  margin-top: $sc-margin-lv3;
}*/
//...
```

4. Save the file.
5. Repeat this procedure in this fashion for any other theme-related HTML or Sass files you intend to customize.

Add a New File to a Theme

When you customize a theme, you can also create new HTML, Sass, or asset files to meet your needs. This includes editing a manifest file to ensure that your new file compiles at activation.

Note: To reference assets within your code, use the HTML and Sass Helpers provided. See [Assets](#) for more information.

To create a new file for a theme:

1. Create your new HTML, Sass, or asset file in the correct location, as required.

File Type	Location Within Workspace/<THEME_DIRECTORY>/
HTML	../Modules/<MODULE>/Templates
Sass	../Modules/<MODULE>/Sass
Assets	../assets

2. Create each file as required. See [Best Practices for Creating Themes](#) for details.



Important: To avoid file name collisions, do not create any new files or folders that share the same name, even if they reside in different locations.

3. Save this file in the correct location within your directory structure.

For example, you want to add a new Sass file titled `_cart-new.scss` to the theme's existing `Cart@1.0.0` module. You save this file in the following location:

```
.../Workspace/<THEME_DIRECTORY>/Cart@1.0.0/Sass/_cart-new.scss
```

4. Open the theme's manifest.json file.

For example:

```
.../Workspace/<THEME_DIRECTORY>/manifest.json
```

5. Include your new file in the appropriate location as required:

- If adding a template, list the file by application. Include the `.tpl` file in the `files` array of the appropriate application object (Shopping, MyAccount, or Checkout). When declaring your new file, include the path to the correct module. The order of your declaration does not matter.

For example:

```
//...
"templates": {
  "application": {
    "shopping": {
      "files": [
        //...
      ]
    }
  }
}
//...
```

- If adding a Sass file, list the `.scss` file in the `files` array of the `sass` object. You set up the Sass entry point in a later step. When declaring your new file, include the path to the correct module. Add this line in a location that makes the most semantic sense within the Sass hierarchy. For example:

```
//...
"sass": {
  "entry_points": {
    "shopping": "Modules/Shopping/shopping.scss",
    "myaccount": "Modules/MyAccount/myaccount.scss",
    "checkout": "Modules/Checkout/checkout.scss"
  }
  "files": [
    //...
  ]
}
//...
```

- If adding an asset, add your asset as part of the `files` array of the `img`, `font-awesome`, or `font` object, as appropriate. When declaring your new file, include the path to the correct folder (`img/`, `font-awesome/`, or `font/`). The order of your declaration does not matter. For example:

```
//...
"assets": {
  "img": {
    "files": [
      //...
    ]
  }
  "font-awesome": {
    "files": [
      //...
    ]
  }
}
//...
```

This ensures that the compiler includes your customizations. In this example, you are adding a Sass file. Therefore, you add `_cart-new.scss` as a dependency to the `files` array of the `sass` object.

Your manifest file might look similar to the following example:

```
"sass": {
  "entry_points": {
    "shopping": "Modules/Shopping/shopping.scss",
    "myaccount": "Modules/MyAccount/myaccount.scss",
    "checkout": "Modules/Checkout/checkout.scss"
  }
  "files": [
    //...
    "Modules/Cart@1.0.0/Sass/_cart-summary.scss",
    "Modules/Cart@1.0.0/Sass/_cart.scss",
    "Modules/Cart@1.0.0/Sass/_cart-new.scss",
    //...
  ]
}
```

6. Save the manifest.json file.
7. If your new file is an asset or template, you have no further action required. However, if your new file is a Sass file, follow these additional steps:
 - a. Identify the application or applications impacted by your new Sass file.
 - b. Edit the application entry point file to include the same dependency you introduced in the manifest file.

Application Impacted	Application Entry Point
Shopping	.../Modules/Shopping/shopping.scss
My Account	.../Modules/MyAccount/myaccount.scss
Checkout	.../Modules/Checkout/checkout.scss

In this example, the Cart module impacts the Shopping application. Therefore, you open the Shopping entry point file and include the dependency. Your Shopping.scss file should look similar to the following example:

```
//...
@import "../Cart@1.0.0/Sass/cart-summary";
```

```
@import "../Cart@1.0.0/Sass/cart";
@import "../Cart@1.0.0/Sass/cart-new";
//...
```

Just as you did in the manifest.json file, place this file in such a manner that makes semantic sense within the Sass hierarchy you are customizing.

- c. Save the application Sass file.

Override Active Extension Files

This section explains how to use the **Override** method to customize HTML and Sass files related to extensions.

When you download an extension, the theme development tools download all extension-related HTML and Sass files and place them in the appropriate folder within the ../Workspace/Extras/extension directory. These files are provided as a reference only. Do not edit these files directly. Instead, use the Override method described below.

A critical factor regarding themes and extensions is the difference in how they are customized. At this time, you cannot create your own custom extensions. However, you can customize the HTML and Sass of any active extensions for a domain and deploy them with your theme customizations.

Example

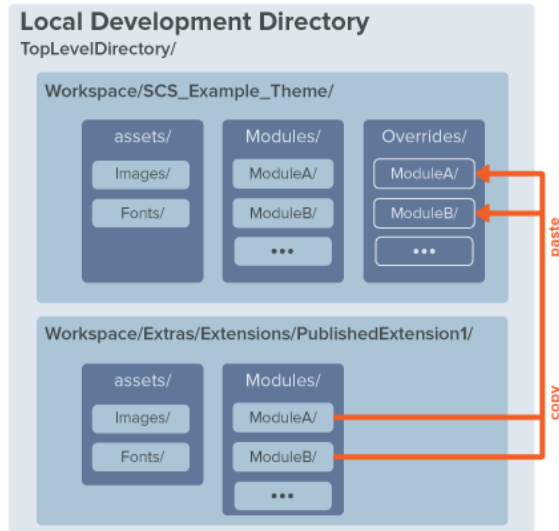
You want to activate a custom theme for your domain. You also want to activate an extension for the same domain. However, your custom theme includes a new color, but the extension's Sass does not include the new definition. You can customize the extension's Sass files to include this color definition using the Override method.

All of your theme customizations and extension overrides are maintained as part of your custom theme. Therefore, when you later activate your custom theme and the extension for your domain, the added functionality (introduced with the extension) includes your new color definition

The Override Method

To customize extension-related HTML and Sass files, you must use the Override method. This involves placing copies of source HTML and Sass from your Extras directory into your theme directory and making your customizations there. When you deploy your changes, the theme development tools detect the new files and initiate an override within the manifest. You cannot create new HTML and Sass files for extensions.

Note: When you downloaded an extension's HTML and Sass files, the theme development tools placed them in your Extras/Extensions directory, sorted by extension. At the same time, the development tools created an identical directory structure in your theme's Overrides directory.



Important: You cannot override asset files. To introduce new assets for your template customizations or extension overrides, add them as you would for any new files within the template directory. See [Add a New File to a Theme](#).

To reference any assets, use the HTML and Sass Helpers as a best practice. See [Assets](#) for details.

To customize an extension using the Override method:

1. Locate the source file you want to override.

For example, you want to override the `_Pub-Extend-Error.scss` file of the **Published Extension 1** extension. Therefore, the file you want to override is in the following location:

```
.../Workspace/Extras/Extensions/PublishedExtension1/Modules/PubExtendModule@1.0.0/Sass/_Pub-Extend-Error.scss
```

2. Copy the source file to your operating system's clip board.
3. Paste a copy of the file in the corresponding location within the theme's Overrides directory.

For example:

```
.../Workspace/<THEME_DIRECTORY>/Overrides/Modules/PubExtendModule@1.0.0/Sass/_Pub-Extend-Error.scss
```

Important: Do not rename these files. These files must share the same name for the Override method to function correctly.

4. Open your new file in your Overrides directory.

5. Follow best practices to customize your new file. See [Best Practices for Creating Themes](#) for important details on customizing these files.
6. Repeat this procedure for all extension-related files you intend to customize.

Best Practices for Creating Themes

Before creating a custom theme or overriding extension-related files, read and understand these best practices. The information in this section provides important steps you must take to ensure that your custom themes and extension overrides deploy without error.

Different files and resources require different recommended best practices. Read the following sections before making any customizations:

However, the following list explains some overarching best practices to follow:

- [General Best Practices](#)
- [HTML Templates](#)
- [Sass](#)
- [Assets](#)

General Best Practices

The following list provides some general knowledge and practices to remember when customizing themes and extensions for SuiteCommerce Standard:

- Do not move, delete, or edit any files located in your Workspace/Extras directory.
- Whenever possible, use the existing folder structure as created when you downloaded themes and extensions to your Workspace directory. If you must add new subdirectories, make sure the paths to any files are included in the manifest and any required entry points.
- Customize all theme-related files directly within the theme's module or assets folders.
- Customize extension-related HTML and Sass files using the Override method. See [Override Active Extension Files](#) for details.
- Place any new assets (images or fonts) in the appropriate location within the theme's assets directory.
- Use helpers when referencing any assets within your HTML or Sass customizations and overrides. See [Assets](#) for details.
- If adding a new file for your theme, declare it within the theme's manifest.json file.
- If adding a new Sass file for your theme, declare it in the appropriate Sass entry point file. See [Sass](#) for details.
- Follow the template context when editing HTML template files or creating overrides. See [HTML Templates](#) for details.
- To avoid file name collisions, do not create any new files or folders that share the same name, even if they reside in different locations. The exception to this practice is when copying and pasting extension file overrides.
- When naming files, folders, themes, and extensions, avoid using spaces or special characters.

HTML Templates

SuiteCommerce Standard templates use the Handlebars.js templating engine to keep the raw HTML separate from the deeper development code. Templates define the HTML for specific areas of the

user interface, such as a button or Home banner. The template engine combines all of these files into a single, finished web page. To achieve this, templates use an easy-to-understand syntax, which is a combination of HTML and Handlebars.js expressions (helpers). These helpers tell the templating engine to include the value of variables (properties and objects) when rendering your site or executing functions.

These variables are specific to each template, depending on what the template is designed to render. This information is called the template's **context**.

Template Context

Each template relies on its context to know what variables are available (what data the template can render). You cannot customize a template to render information it cannot provide. Therefore, you must operate within this context when making any customizations to pre-existing templates or introducing your own.

Each SuiteCommerce Standard template includes a list of context variables within the file itself. This lists the context objects, types, or any properties as part of an array. This information is nested within comment blocks at the end of each file.

The following example from `case_list.tpl` depicts this context notation.

```
...
{!----
Use the following context variables when customizing this template:

    pageHeader (String)
    hasCases (Number)
    isLoading (Boolean)
    showPagination (Boolean)
    showCurrentPage (Boolean)
    showBackToAccount (Boolean)

----}}
```

The following code snippet depicts the `hasCases` number and `isLoading` boolean variables as used in the `case_list.tpl` template code. This code either renders a support case list or renders a **Loading** or **No Cases Found** string, depending on the query results.

```
...
{{#if hasCases}}
    <table class="case-list-recordviews-table">
        <thead class="case-list-content-table">

            ...

        </thead>
        <tbody data-view="Case.List.Items"></tbody>
    </table>
{{else}}
    {{#if isLoading}}
        <p class="case-list-empty">{{translate 'Loading...'}}</p>
    {{else}}
        <p class="case-list-empty">{{translate 'No cases were found'}}</p>
    {{/if}}
}}
```



```
{{/if}}
...
```

To find the template context in a template file:

1. In an editor, open the source .tpl file for the template you intend to customize.
2. At the end of the file, look for the following string, located within comment tags:

```
{{!--
Use the following context variables when customizing this template:
...}}
```

3. Note the context variables available to the template and customize accordingly.

To find the context variables using the browser developer tools:

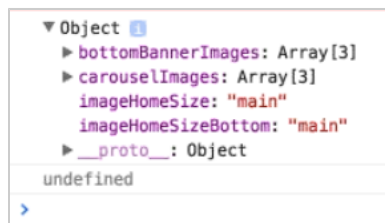
At this time, some template files do not include this context as described above. In these cases, you can use the `{{log this}}` helper as described below. This procedure assumes that you have already begun customizing a template. Do not edit source files directly.

Note: The examples in this section uses Google's Chrome browser developer tools.

1. Open the template file you are customizing in an editor.
2. Include the following helper as the first line in the template for which you want to reveal the context.

```
{{log this}}
```

3. Deploy your customization to your local server. See [Test a Theme on a Local Server](#).
4. Navigate to the page rendering your custom template.
5. Using your browser's developer tools, inspect and refresh the page.
6. The developer tools **Console** tab lists the variables available to the template as the output of the `{{log this}}` helper:



Sass

The SCS design architecture lets you customize the design elements of your web store experience using Sass, or Syntactically Awesome StyleSheets. Sass is a scripting language that is transformed into CSS when you use the theme development tools to compile and deploy your customizations to the application. The SuiteCommerce Designer's Guide provides more information on customizable Sass styles. See the help topic [Design Architecture](#) for details.

Note: All SuiteCommerce Sass files are named as partials (with a preceding underscore), such as `_BaseSassStyles.scss`. These files are located in the Sass directories of any modules requiring Sass variable definitions.

Sass Entry Points

Every theme relies on an **entry point** file to load Sass into the correct application (Shopping, My Account, or Checkout). Each entry point declares the Sass files that are part of the application. For example, if a Sass file affects the shopping application only, it needs to load through the `shopping.scss` entry point. If it affects all three applications, it needs to load through all three entry points.

For an example of how to edit an entry point, see [Add a New File to a Theme](#).

Important: All pre-existing and new Sass files must be declared in the theme's `manifest.json` file. See [Theme Manifest](#) for more information.

Each theme includes the following Sass entry points:

Application Impacted	Application Sass File
Shopping	<code>.../Modules/Shopping/shopping.scss</code>
My Account	<code>.../Modules/MyAccount/myaccount.scss</code>
Checkout	<code>.../Modules/Checkout/checkout.scss</code>

Assets

An asset is an image or font that is not managed by NetSuite but still part of a theme or extension. An example of an asset is a carousel image, an icon, or a new font. This can be either a pre-existing asset or one that you introduce as part of a custom theme.

When you run the `gulp theme:fetch` command:

- Assets for the active theme download to your `../Workspace/<THEME_DIRECTORY>/assets` folder. This is the location where you manage all of your assets (new and pre-existing).
- Assets for any active extensions download to your `../Workspace/Extras/Extensions/<EXTENSION_DIRECTORY>/assets` folder. Do not move, delete, edit, or add files in this location.

When you activate your theme, all assets are placed in specific locations in your NetSuite File Cabinet based on parameters you specify when you deployed your theme (vendor name, theme name, and theme version). Later, when you activate a newer version of the same theme, your assets are now located in a different location in your File Cabinet. Your code must adapt to the change in the path. If you use absolute paths, the links to these assets will break.

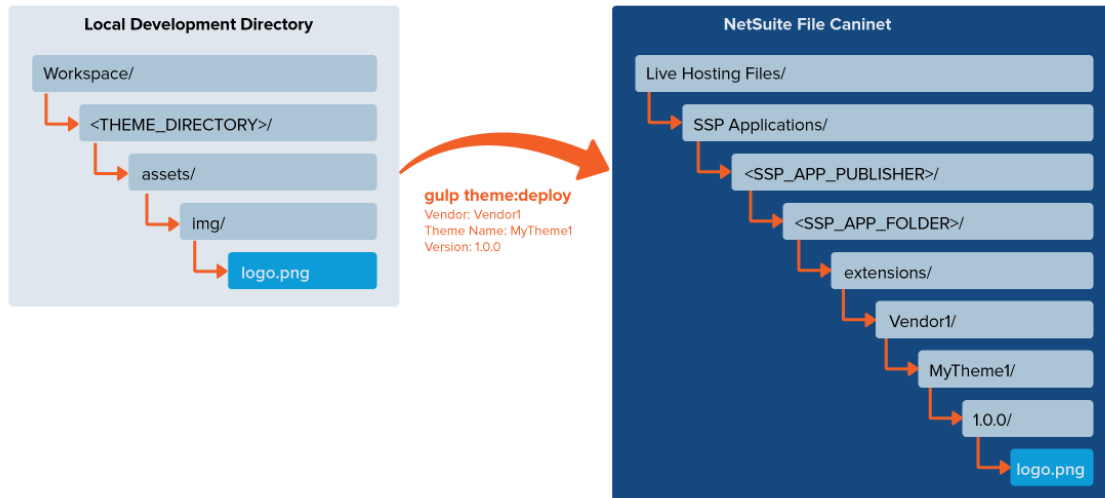
Important: Do not use absolute paths when referring to assets in your code customizations and overrides. NetSuite manages asset files in different locations in NetSuite based on the theme's vendor, extension name, version, etc. As a best practice, always use the HTML and Sass helpers when referencing assets to ensure that you maintain dynamic paths without unnecessary code changes.

SuiteCommerce Standard provides a few HTML and Sass Helpers to maintain dynamic paths to your assets, regardless of the vendor, theme, version, etc.

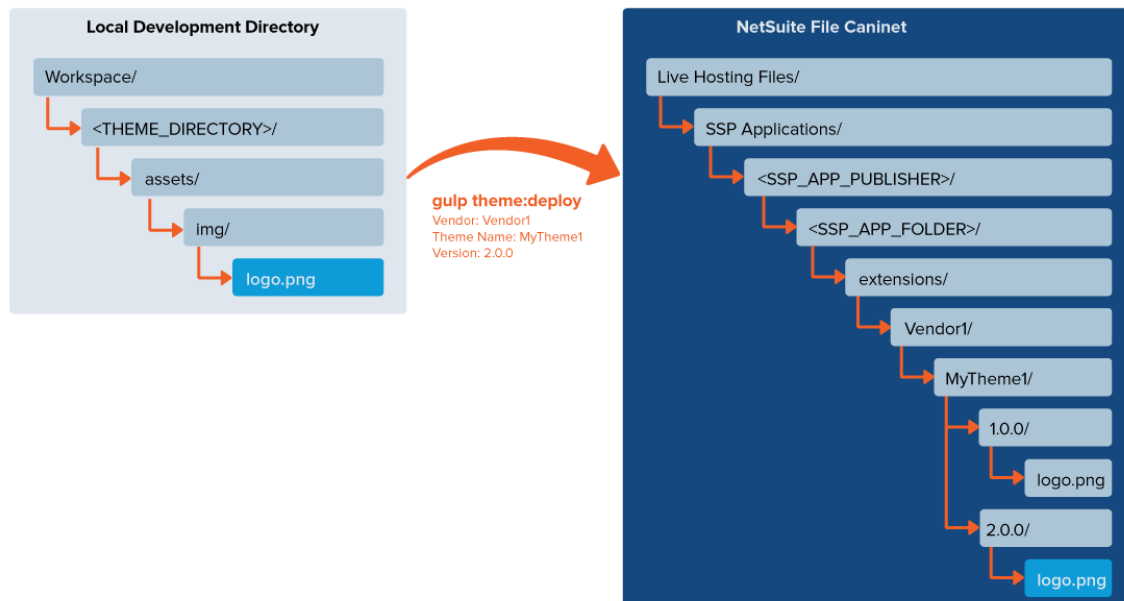
Example:

Using the theme development tools, you deploy a custom theme (**MyTheme1**, Version: **1.0.0**). You then activate that theme for a specific site and domain. The compiler places all assets in your NetSuite File Cabinet. The exact location is specific to the SSP Application, vendor, extension, and version that you specified when you deployed your custom theme.

This location might be similar to the following:



Later, you decide to update your custom theme, giving it a new version, **2.0.0**. When you activate your latest version, the compiler places all assets into a new location:



HTML Helpers

The following four Handlebars.js helpers provide a means to maintain dynamic paths to your assets when customizing HTML templates.

getExtensionAssetsPath(default_value)

Use this HTML template helper:

- To access pre-existing assets included with an active extension
- In your extension overrides (templates)

Note: The `default_value` argument is the relative path to the asset.

Example Syntax in a Template File:

The following is an example of how to use this helper in an HTML template:

```

```

Example Result:

This helper returns the active extension's asset path (where `<FULL_PATH>` is the SSP Application base path):

```
<FULL_PATH>/extensions/<VENDOR>/<EXTENSION>/<VERSION>/img/logo.png
```

getExtensionAssetsPathWithDefault(config_value, default_value)

Use this HTML template helper:

- To access pre-existing assets included with an active extension
- In your extension overrides (templates)
- When you expect that the asset has a configured path in NetSuite, but want to provide a default path if the configuration does not exist.

Note: The first argument (`config_value`) is the path as configured in NetSuite. In cases where the asset is not configured, you provide a fallback path (`default_value`) to be retrieved from the extension's assets.

Example Syntax in a Template File:

The following is an example of how to use this helper in an HTML template:

```

```

Example Result:

This helper returns the active extension's asset path. If the first argument is defined, this helper returns the path as configured in NetSuite:

```
<MY_PATH>/img/other_logo.png
```

If the first argument is undefined, this helper uses the second argument to return the correct path of the active extension:

```
<FULL_PATH>/extensions/<VENDOR>/<EXTENSION>/<VERSION>/img/logo.png
```

getThemeAssetsPath(default_value)

Use this HTML template helper:

- To access new and pre-existing assets included in your theme directory
- In your extension overrides (templates)
- In your theme customizations (templates)

Using this helper, `default_value` is the relative path to the asset.

Example Syntax in Template File:

```

```

Example Result:

This helper returns the active theme's asset path (where `<FULL_PATH>` is the SSP Application base path):

```
<FULL_PATH>/extensions/<VENDOR>/<THEME_NAME>/<VERSION>/img/logo.png
```

getThemeAssetsPathWithDefault(config_value, default_value)

Use this HTML template helper:

- To access new and pre-existing assets included in your theme directory
- In your extension overrides (templates)
- In your theme customizations (templates)
- When you expect that the asset has a configured path in NetSuite, but want to provide a default path if the configuration does not exist.

Using this helper, `config_value` is the path as configured in NetSuite. In cases where the asset is not configured, you provide a fallback path (`default_value`) to be retrieved from the extension's assets.

Example Syntax in Template File:

```

```

Example Result:

This helper returns the active theme's asset path. If the first argument is defined, this helper returns the path as configured in NetSuite:

```
<MY_PATH>/img/other_logo.png
```

If the first argument is undefined, this helper uses the second argument to return the correct path of the active theme:

```
<FULL_PATH>/extensions/<VENDOR>/<THEME_NAME>/<VERSION>/img/logo.png
```

Sass Helpers

The following two helpers provide a means to maintain dynamic paths to your assets when customizing Sass files.

getExtensionAssetsPath(\$asset)

Use this Sass helper:

- To access pre-existing assets included with an active extension
- In your extension overrides (Sass)

Example Syntax in a Sass File:

```
body {
  background-image: url(getExtensionAssetsPath('img/background-image.png'));
}
```

Example Result:

This helper returns the active extensions's asset path.

```
<FULL_PATH>/extensions/<VENDOR>/<EXTENSION>/<VERSION>/img/logo.png
```

getThemeAssetsPath(\$asset)

Use this Sass helper:

- To access new and pre-existing assets included in your theme directory
- In your extension overrides (Sass)
- In your theme customizations (Sass)

Example Syntax in a Sass File:

```
body {
  background-image: url(getThemeAssetsPath('font-awesome'));
}
```

Example Result:

This helper returns the active theme's asset path.

```
<FULL_PATH>/extensions/<VENDOR>/<THEME_NAME>/<VERSION>/font-awesome
```

Theme Manifest

Your theme's Workspace directory includes various manifest.json files. The manifests are JSON files that include all the information required to compile resources for an active theme or extension. Of these files, only one requires editing when you customize a theme. This is the manifest file included in your theme's development directory:

```
../Workspace/<THEME_DIRECTORY>/manifest.json
```

This topic explains this file.

Workspace/<THEME_DIRECTORY>/manifest.json

This file lists all HTML templates, Sass, and assets related to the active theme that you downloaded when you ran the `gulp theme:fetch` command. You only need to edit this file to introduce any new HTML, Sass, or asset files you create as part of your theme customizations.

This file also manages any extension overrides. However, the theme development tools add the necessary overrides to this manifest when you deploy your customizations.

See [Add a New File to a Theme](#) for instructions on how to edit this file when you add any new HTML, Sass, or assets.

The rest of this topic explains the different areas of the Manifest.json file.

Theme Metadata

The first entries in the manifest file include metadata about the theme or extension itself. These fields are automatically populated when you initially run the `gulp theme:deploy` command.

```
{
  "name": "StandardTheme",
  "vendor": "SuiteCommerce",
  "type": "theme",
  "target": "SCS",
  "version": "1.2.0",
  "description": "Standard theme for SCS",
  //...
```

- Name (string) – uniquely identifies the name of the theme. This field is required.
- Vendor (string) – identifies the vendor as a string. This field is required.
- Type (string) – indicates if the type as a **theme**. This field is required.
- Target (comma-separated string) – indicates the SuiteCommerce Applications supported by the theme, such as SCS. This field is required.
- Version (string) – indicates the version of the theme, such as 1.0.0. This field is required.
- Description (string) – provides a description of the theme as it appears in NetSuite. This field is optional.

Overrides

The `override` object is only included if you introduce extension overrides. When you use the Override method, the Gulp.js commands detect any HTML or Sass overrides and include them in this file automatically.

For example, if you override the `_error.sass` file of the **Extension1** extension and run the `gulp theme:deploy` command, the theme development tools add the following override your theme's manifest.json file as part of the deployment process:

```
//...
"override": [
  {
    "src": "Overrides/Extension1/Modules/MyExamplePDPEExtension1/Sass/_error.scss",
    "dst": "Extension1/Modules/MyExamplePDPEExtension1/Sass/_error.scss"
  },
  //...

```

Templates

The `templates` object lists all HTML template files included in the theme by application. The `application` object includes one object per application (`shopping`, `myaccount`, and `checkout`). Each application lists each file in the `files` array.

You manually add any new template files to this array that you introduced as part of your theme customizations.

```
//...
"templates": {
  "application": {
    "shopping": {
      "files": [
        "Modules/AddToCartModifier/Templates/add_to_cart_modifier.tpl"
        //...
      ]
    }
    "myaccount": {
      "files": [
        //...
      ]
    }
    "checkout": {
      "files": [
        //...
      ]
    }
  }
}
//...

```

Sass

The `sass` object declares the paths to each application entry point and all Sass files to be loaded when you deploy. You manually add any new Sass files to the `files` array that you introduced as part of your theme customizations.



Note: When listing a new Sass file, declare each file in an order that makes the most semantic sense within the Sass hierarchy.

```
//...
"sass": {

```



```

"entry_points":{
  "shopping": "Modules/Shopping/shopping.scss",
  "myaccount": "Modules/MyAccount/myaccount.scss",
  "checkout": "Modules/Checkout/checkout.scss"
}
"files":[
  "Modules/Shopping/shopping.scss",
  "Modules/MyAccount/myaccount.scss",
  "Modules/Checkout/checkout.scss",
  "Modules/twitter-bootstrap-sass@3.3.1/assets/stylesheets/bootstrap/_alerts.scss",
  //...
]
}
//...

```

Assets

The `assets` object defines paths to the images and fonts located in the theme directory's `assets` folder. This defines an array for each image and font used. These paths are relative to the theme's `assets` folder path. This is where you add any new asset files introduced as part of your theme customizations.

```

//...
"assets": {
  "img": {
    "files": [
      "img/favicon.ico",
      "img/image-not-available.png",
      "img/add-to-cart-logo.png",
      //...
    ]
  },
  "font-awesome": {
    "files": [
      "font-awesome/FontAwesome.otf",
      "font-awesome/custom/fontawesome-webfont.eot",
      //...
    ]
  },
  "fonts": {
    "files": [
      "fonts/DancingScript-Bold.ttf",
      "fonts/DancingScript-Regular.ttf",
      //...
    ]
  }
}
//...

```

Record Paths


The final part of the manifest file lists the path to the theme, the Extension record and Activation IDs as stored in NetSuite.

```
//...  
"path": "SuiteBundles/Bundle 193239/SuiteCommerce Base Theme",  
"extension_id": "4535",  
"activation_id": "59"
```

Develop Extensions

Extension Development Checklist

Follow these steps to create, develop, test, and deploy your own theme.

Task	Step	Description	Topic
<input type="checkbox"/>	Create Baseline Extension Files	The next step is to create a baseline set of files from which you can create an extension. The extension developer tools help you out by walking you through a series of questions. The developer tools then create an example extension within the correct directory structure as a starting point for development.	Create an Extension
<input type="checkbox"/>	Download the Active Theme	Before you can test any extension on a local server or deploy it to NetSuite, your local environment requires a theme to render your site in a browser during testing. These theme files are also required by the developer tools to process your extension when deploying to an account. Follow these steps to download the active theme for a domain to use for local testing.	Download the Active Theme for Extension Development
<input type="checkbox"/>	Develop Your Extension		
<input type="checkbox"/>	Enable Your CCT for Use with SMT	If you are building your extension as a CCT, you must complete the process by creating your custom records in NetSuite and developing your CCT module.	To Build a CCT:
<input type="checkbox"/>	Deploy Your SuiteScript and Configuration Files to NetSuite	<p>If your extension includes SuiteScript or configuration (JSON) files, you must deploy your files to your account and activate the extension before testing locally. SuiteScript includes services, which do not exist in your account's backend until you deploy them. Likewise, changes to configuration JSON files do not apply to a domain until deployed.</p> <div>  Note: Although deploying your code is a required task later in the process, doing so at this time is required if you intend to test your extension locally and your extension includes SuiteScript or configuration files. </div>	Deploy an Extension to NetSuite Activate Themes and Extensions
<input type="checkbox"/>	Test Your Extension Locally	Follow these instructions to run your extension on a local server. This involves	Test an Extension on a Local Server

Task	Step	Description	Topic
		combining your extension development files into compiled, runtime files for a specific domain. If your extension includes SuiteScript or Configuration files, you must deploy your files before testing them locally.	
<input type="checkbox"/>	Deploy Your Extension to NetSuite	Follow these instructions to deploy your extension files to your NetSuite Account.	Deploy an Extension to NetSuite
<input type="checkbox"/>	Activate Your Theme and any Necessary Extensions	After you deploy your extension to an account, you must now activate it for a domain before your changes take effect. This is required if you are for both creating a new extension or updating a pre-existing one.	Activate Themes and Extensions

Extension Development Resources

Read the following sections for reference materials for developing extensions:

- [Extension Manifest](#) – This section explains the extension's manifest.json file.

Create an Extension

The following section explains how to use the extension developer tools to build a baseline extension. This is a critical first step when building a new extension.

This section explains how to:

- [Create a Baseline Extension](#)
- [Create Additional Modules for an Extension](#)
- [Create Custom Content Types for an Extension](#)

What Happens When You Create a Baseline Extension?

When you run the `gulp extension:create` command, the extension developer tools:

- Create the Workspace directory in your top-level development directory, unless it already exists.
- Create a subdirectory to house all of your extension's code. This is where you develop your extension.
- Create a manifest.json file. This file includes all the information required to compile resources for your extension.

Example

You run the `gulp extension:create` command and build a baseline extension, **MyCoolExtension** with a module called **MyCoolModule**. You set this extension to include all available files.

In this example, your Workspace directory structure should look similar to following:

```
<TopLevelDevelopmentDirectory>/
  Workspace/
    MyCoolExtension/
      assets/
        fonts/
        img/
        services/
      Modules/
        MyCoolModule/
          Configuration/
          JavaScript/
          Sass/
          SuiteScript/
          Templates/
        manifest.json
```

Create a Baseline Extension

A **Baseline Extension** is a set of files, installed locally in your extension development directory. You use these files as a basis to build any extension for your SuiteCommerce site.

When you run the `gulp extension:create` command, the developer tools create this baseline extension in a Workspace directory within your top-level extension development directory. The baseline extension starts out with one module that you design, based on your needs. It can include any combination of the following file types within one new module:

- Templates
- Sass
- Configuration
- JavaScript
- SuiteScript

To create a baseline extension:

1. Open a command line or terminal.
2. Access the top-level extension development directory you created when you installed the extension developer tools.
3. Enter the following command:

```
gulp extension:create
```

4. When prompted, enter the following information:

Pressing **enter** results in the default value.

- Extension Name – Provide a name for your extension.
- Vendor Name – Enter the partner or business vendor name.
- Version – Add a version number for your extension. Best practice is to use a semantic versioning (SemVer) system.
- Description – Provide some text to describe your extension.

- Applications – Select one or more applications to use your extension (Shopping, Checkout, My Account). An asterisk (*) identifies a selected application.
 - Use the spacebar key to select or deselect an application.
 - Use the UP and DOWN arrow keys to scroll through the application list.
 - Toggle the A key to select or deselect all applications.
- File Types to Include – Select one or more file types your extension requires (Templates, Sass, Configuration, JavaScript, SuiteScript). An asterisk (*) identifies a selected application.
- Initial Module Name – Name the module to be created as part of the baseline extension. Do not use spaces or special characters.



Important: If your extension includes SuiteScript, you must deploy your extension to a NetSuite account before you can test your extension locally. The extension's JavaScript files require a service on the NetSuite backend. This service does not exist until you deploy your code. See [Deploy an Extension to NetSuite](#) for details.

Create Additional Modules for an Extension

The `gulp extension:create` command builds an extension with one baseline module. However, if your extension requires more than one module, the developer tools let you create more using the `–module` attribute.

To add an additional module to your baseline extension:

1. Create a baseline extension, if you have not done so already. See [Create a Baseline Extension](#) for details.
2. Open a command line or terminal.
3. Access the top-level extension development directory you created when you installed the extension developer tools.
4. Enter the following command:

```
gulp extension:create-module
```

5. When prompted, enter the following information:
 - Module Name – Name the module to be created as part of the baseline extension. You can add more later on.
 - File Types to Include – Select one or more file types your extension requires (Templates, Sass, Configuration, JavaScript, SuiteScript). An asterisk (*) identifies a selected application.
 - Extension – Select the name of the extension within which you want to add this module.
6. Repeat steps 4–5 to add any additional modules as needed.

Create Custom Content Types for an Extension

When you create your baseline extension files, the developer tools provide a method to build your extension as a CCT. The developer tools provide on-screen instructions to assist you with developing your CCT. This requires:

- Creating a baseline extension

- Adding one or more baseline CCTs
- Customizing your CCT and preparing it for use with Site Management Tools

Note: To implement a CCT using Site Management Tools, you must configure the **CMS Adapter Version** to 3 in the SuiteCommerce Configuration record. This property is located in the **Integrations** tab and **Site Management Tools** subtab.

To add a CCT to your baseline extension:

1. Create a baseline extension, if you have not done so already. See [Create a Baseline Extension](#) for details.
2. Open a command line or terminal.
3. Access the top-level extension development directory you created when you installed the extension developer tools.
4. Enter the following command:

```
gulp extension:create-cct
```

5. When prompted, enter the following information:
 - **CCT Name** – Name the CCT to be added to your baseline extension.
 - **File Types to Include** – Select one or more file types your extension requires (Templates, Sass, Configuration, JavaScript, SuiteScript). An asterisk (*) identifies a selected application.
 - **Extension** – Select the name of the extension within which you want to add this module.
6. Repeat steps 4–5 to add any additional CCTs as needed.

To Build a CCT:

After you have successfully created your baseline CCT, you must set it up for use with Site Management Tools and customize your code to meet your needs.

1. Open the extension's manifest.json file. This can be found in Workspace/<EXTENSION_DIRECTORY>/.
This file lists the following as part of the manifest metadata. You need this information when setting up your records in NetSuite.
 - **icon** – Equals a default icon that is visible in SMT Admin. As a default, the developer tools provide one to help you get started.
 - **settings_record** – Equals the **ID** of the custom record you associate with this CCT.
 - **registercct_id** – Equals the **Name** field of the CMS Content Type Record for your CCT. This is also the value of the **id** property within the `registerCustomContentType()` method of your CCT module's entry point JavaScript file.
2. Prepare your CCT for use with SMT.
Follow the instructions listed in the help topic [Custom Content Type](#). Build your custom record and your CMS Content Type Record using the parameters explained in Step 1.
3. Develop your CCT module.
Follow the instructions listed in the help topic [Create a CCT Module](#). Ensure that your entry point JavaScript file incorporates your CMS Content Type record name, as described in Step 1.
4. Deploy and activate your extension. Then log into SMT to test confirm that your CCT was added correctly. See the help topic [Site Management Tools](#) for details on using SMT.


- See [Activate Themes and Extensions](#) for instructions on deploying your extension.
- See [Test and Deploy an Extension](#) for instructions on activating your extension.

Download the Active Theme for Extension Development

Before you can deploy an extension or test it on a local server, you must first download files for the currently active theme by running the following command:

```
gulp extension:fetch
```

This step is required before you run your extension on a local server or deploy to a NetSuite account. Before you test on a local server or deploy to NetSuite, the developer tools must ensure that your code compiles correctly. To do this, the developer tools require Sass and HTML files. When you download the active theme, you provide these files.

 **Important:** You must have a theme active for the selected domain for this command to run.

Download the active theme files:

1. In your local developer environment, open a command line or terminal and access the top-level extension development directory.
2. Run the following command:

```
gulp extension:fetch
```

3. When prompted, enter the following information:
 - Email – Enter the email address associated with your NetSuite account.
 - Password – Enter your account password.
 - NetSuite account/role – Select the NetSuite account where SuiteCommerce Standard is installed.
 - Website – Select your website with the domain you want to access.
 - Domain – Select your domain with the active theme you want to download.

The `gulp extension:fetch` command creates a theme subdirectory in your Workspace/Extras directory. These files are for use by the developer tools only. Do not add, edit, delete, or move any files in this location.

Test and Deploy an Extension

After creating your extension files, the next step is to test on a local server and, assuming the files meet your approval, deploy them to a live site. Use the extension developer tools to test on a local server or deploy them to your NetSuite account.

This section explains how to:

- Test an Extension on a Local Server

■ Deploy an Extension to NetSuite

Test an Extension on a Local Server

In addition to deploying your themes and extensions to NetSuite, you can also test them by running SuiteCommerce Standard or SuiteCommerce Advanced on a local server. The local server is installed as part of the Node.js installation and uses the Express web framework. The server starts automatically when you run the `gulp theme:local` or `gulp extension:local` command using the developer tools.

When the server starts, Gulp.js initializes watch tasks that listen for changes to files in the JavaScript, Templates, or Sass directories. When you save changes to a file, gulp automatically recompiles the source files and updates the LocalDistribution directory. Gulp also outputs a message to the console when an update occurs.



Important: Typically, you test your code locally before deploying to a live site in a NetSuite account. However, if you are developing an extension that includes SuiteScript or configuration (JSON) files, you must deploy your files to your account and activate the extension for these changes to be accessible by the local server. SuiteScript includes services, which do not exist in your account's backend until you deploy them. Likewise, changes to configuration JSON files do not apply to a domain until deployed. See [Activate Themes and Extensions](#).



Note: If you modify any manifest files, you must restart your local server to see changes.

To set up your extension for a local server, access the top-level directory in a command terminal and run the `gulp extension:local` command. This compiles all source files into combined files within a LocalDistribution/tmp directory for use by the local version of the application.

To test your extension on a local server:

1. Open a command line or terminal and access the top-level development directory. This is the same directory created when you extracted the Extension Developer Tools.
2. Run the following command:

```
gulp extension:local
```

If this is the first time you are running `gulp extension:local` in this directory, this command creates a subdirectory called LocalDistribution. It then compiles the source files and outputs them to this directory.

3. Navigate to the local version of the application using one of the following URLs:
 - **Shopping:** `http://<DOMAIN_NAME>/c.<ACCOUNT_ID>/<SSP_APPLICATION>/shopping-local.ssp`
 - **My Account:** `http://<DOMAIN_NAME>/c.<ACCOUNT_ID>/SSP_APPLICATION/my_account-local.ssp`
 - **Checkout:** `http://<DOMAIN_NAME>/c.<ACCOUNT_ID>/SSP_APPLICATION/checkout-local.ssp`

In the above URL patterns, you must replace the following variables with values for your specific environment:

- **DOMAIN_NAME** — replace this value with the domain name configured for your NetSuite website record.
- **ACCOUNT_ID** — replace this value with your NetSuite account ID.
- **SSP_APPLICATION** — replace this value with the URL root that you are accessing.

For SuiteCommerce Standard implementations, this variable should read `scs`. For example:

```
http://www.mysite.com/c.123456/scs/shopping-local.ssp
```

For SuiteCommerce Advanced, this variable equals the URL root of the SCA implementation. For example:

```
http://www.mysite.com/c.123456/sca-dev-aconcogua/shopping-local.ssp
```

The URLs you use should be similar to the following examples:

Note: When accessing the secure checkout domain using HTTPS on the local server, you must use a different URL. See *Using Secure HTTP (HTTPS) with the Local Server* for more information.

The server starts automatically. With the local server running, you can make changes to your local files and see the results immediately. Gulp.js automatically recompiles the application when you save any changes to any JavaScript, Sass, and Templates. Simply refresh your browser to see the changes.

Deploy an Extension to NetSuite

The term **deploy** refers to what happens when you use Gulp.js to upload source files and any custom changes to a hosted site. To do this, you use the following command:

```
gulp extension:deploy
```

This command validates your code, copies them into a local DeployDistribution directory, and updates the manifest.json. The developer tools then upload these files to your NetSuite account, making them available for activation.

Note: For more Gulp.js commands used with extensions, see [Extension Developer Gulp Commands](#).

To deploy your extension to NetSuite:

1. In your local developer environment, open a command line or terminal and access the top-level development directory.
2. Run the following command:

```
gulp extension:deploy
```

3. When prompted, enter the following information:
 - Email – Enter the email address associated with your NetSuite account.
 - Password – Enter your account password.
 - NetSuite account – Select the NetSuite account where SuiteCommerce Standard is installed.
 - Website – Select your website with the domain you are customizing.
 - Domain – Select your domain with the active themes and extensions you want to download.
4. In NetSuite, go to Setup > SuiteCommerce Advanced > Extensions Management.
5. Activate your new extension for the domain or domains of your choice. See [Activate Themes and Extensions](#) for instructions.



Important: Any changes to your extension source code do not apply to your site until you activate the extension for a specific domain using the Manage Extensions wizard in NetSuite. The deploy process includes a notation reminding you of the domain and theme name to set during this process.

What Happens When You Deploy an Extension?

The deployment process is specific to uploading custom extensions to a location in your NetSuite File Cabinet. This process does not compile any files or associate any files with a site or domain. To do that, you must activate a theme and include your extensions as necessary.

During the deployment process:

- The development tools copy all of your custom extension development files (modules and assets) into the DeployDistribution folder in your top-level development directory. If this directory does not exist, the development tools create it.
- The extension development tools validate your code.
- The extension development tools upload these files to your NetSuite file cabinet (maintaining the same organizational structure). These are simply your development files. Nothing is compiled or activated at this time.

Extension Manifest

Your extension's Workspace directory includes various manifest.json files. The manifests are JSON files that include all the information required to compile resources for an active theme or extension. Of these files, only one requires editing when you customize a theme. This is the manifest file included in your theme's development directory:

```
../Workspace/<THEME_DIRECTORY>/manifest.json
```

This topic explains this file.

Workspace/<EXTENSION_DIRECTORY>/manifest.json

This file lists all JavaScript, JSON, SuiteScript, HTML templates, Sass, and assets related to your extension. You should not need to edit this file. This file is created when you run the `gulp extension:create` command and updates to declare all extension files when you run either `gulp extension:local` or `gulp extension:deploy`.

The rest of this topic explains the different areas of the Manifest.json file.

Extension Metadata

The first entries in the manifest file include metadata about the extension itself. These fields are automatically populated when you initially run the `gulp extension:create` command.

```
{
  "name": "MyCoolExtension",
  "vendor": "Acme",
  "type": "extension",
  "target": "SCS",
  "version": "1.0.0",
```

```
"description": "My cool extension does magic!",
//...
```

- Name (string) – uniquely identifies the name of the extension. This field is required.
- Vendor (string) – identifies the vendor as a string. This field is required.
- Type (string) – indicates if the type as an **extension**. This field is required.
- Target (comma-separated string) – indicates the SuiteCommerce Applications supported by the theme, such as SCS. This field is required.
- Version (string) – indicates the version of the theme, such as 1.0.0. This field is required.
- Description (string) – provides a description of the extension as it appears in NetSuite. This field is optional.

If your extension is set up as a CCT, your manifest's metadata includes specific information regarding your CCT.

```
//...
  "cct": {
    "icon": "img/cct_acme_mynewextension_icon.svg",
    "settings_record": "customrecord_cct_acme_mynewextension",
    "registercct_id": "cct_acme_mynewextension"
  },
//...
```

In this example:

- `icon` – Equals a default icon that is visible in SMT Admin.
- `settings_record` – Equals the **ID** of the custom record you associate with this CCT.
- `registercct_id` – Equals the **Name** field of the CMS Content Type Record for your CCT. This is also the value of the `id` property within the `registerCustomContentType()` method of your CCT module's entry point JavaScript file.

Assets

The `assets` object defines paths to the images and fonts located in the theme directory's `assets` folder. This defines an array for each image and font used. These paths are relative to the theme's `assets` folder path. Extensions treat services as assets. These are created on NetSuite servers when you activate/deploy your extension.

If the extension developer tools detect file name with the pattern `XYZ.ServiceController.js`, they create the service (`.ss`) file and add it to the manifest. Later, when you activate the extension, the declared `.ss` moves to the backend as an asset.

```
//...
"assets": {
  "img": {
    "files": []
  },
  "fonts": {
    "files": []
  },
  "services": {
    "files": [
      "services/MyCoolModule.Service.ss",

```

```

        "services/AdditionalCoolModule.Service.ss"
    ]
}
},
//...

```

Configuration

The `configuration` object defines paths to the JSON files in your extension directory's Configuration folder.

```

//...
"configuration": {
    "files": [
        "Modules/MyCoolModule/Configuration/MyCoolModule.json",
        "Modules/AdditionalCoolModule/Configuration/AdditionalCoolModule.json"
    ]
},
//...

```

Templates

The `templates` object lists all HTML template files included in the extension by application. The `application` object includes one object per application (`shopping`, `myaccount`, and `checkout`). Each application lists each file in the `files` array.

```

//...
"templates": {
    "application": {
        "shopping": {
            "files": [
                "Modules/MyCoolModule/Templates/acme_mycoolextension_mycoolmodule_list.tpl",
                "Modules/MyCoolModule/Templates/acme_mycoolextension_mycoolmodule_edit.tpl",
                "Modules/AdditionalCoolModule/Templates/acme_mycoolextension_additionalcoolmodu
le_list.tpl",
                "Modules/AdditionalCoolModule/Templates/acme_mycoolextension_additionalcoolmodu
le_edit.tpl"
            ]
        },
        "myaccount": {
            "files": [
                "Modules/MyCoolModule/Templates/acme_mycoolextension_mycoolmodule_list.tpl",
                "Modules/MyCoolModule/Templates/acme_mycoolextension_mycoolmodule_edit.tpl",
                "Modules/AdditionalCoolModule/Templates/acme_mycoolextension_additionalcoolmodu
le_list.tpl",
                "Modules/AdditionalCoolModule/Templates/acme_mycoolextension_additionalcoolmodu
le_edit.tpl"
            ]
        },
        "checkout": {
            "files": [
                "Modules/MyCoolModule/Templates/acme_mycoolextension_mycoolmodule_list.tpl",

```

```

        "Modules/MyCoolModule/Templates/acme_mycoolextension_mycoolmodule_edit.tpl",
        "Modules/AdditionalCoolModule/Templates/acme_mycoolextension_additionalcoolmodu
le_list.tpl",
        "Modules/AdditionalCoolModule/Templates/acme_mycoolextension_additionalcoolmodu
le_edit.tpl"
    ]
  }
}
},
//...

```

Sass

The `sass` object declares the paths to each application entry point and all Sass files to be loaded when you deploy.

Note: If manually listing Sass files, declare each file in an order that makes the most semantic sense within the Sass hierarchy.

```

//...
"sass": {
  "entry_points": {
    "shopping": "Modules/MyCoolModule/Sass/_mycoolextension-mycoolmodule.scss",
    "myaccount": "Modules/MyCoolModule/Sass/_mycoolextension-mycoolmodule.scss",
    "checkout": "Modules/MyCoolModule/Sass/_mycoolextension-mycoolmodule.scss"
  },
  "files": [
    "Modules/MyCoolModule/Sass/_mycoolextension-mycoolmodule.scss",
    "Modules/MyCoolModule/Sass/_mycoolmodule.scss",
    "Modules/AdditionalCoolModule/Sass/_mycoolextension-additionalcoolmodule.scss",
    "Modules/AdditionalCoolModule/Sass/_additionalcoolmodule.scss"
  ]
},
//...

```

JavaScript

The `javascript` object declares the paths to each application entry point and all JavaScript files to be loaded when you deploy.

```

//...
"javascript": {
  "entry_points": {
    "shopping": "Modules/MyCoolModule/JavaScript/Acme.MyCoolExtension.MyCoolModule.js",
    "myaccount": "Modules/MyCoolModule/JavaScript/Acme.MyCoolExtension.MyCoolModule.js",
    "checkout": "Modules/MyCoolModule/JavaScript/Acme.MyCoolExtension.MyCoolModule.js"
  },
  "application": {
    "shopping": {
      "files": [
        "Modules/MyCoolModule/JavaScript/Acme.MyCoolExtension.MyCoolModule.js",
        "Modules/MyCoolModule/JavaScript/MyCoolModule.Router.js",

```

```

        "Modules/MyCoolModule/JavaScript/MyCoolModule.List.View.js",
        "Modules/MyCoolModule/JavaScript/MyCoolModule.Edit.View.js",
        "Modules/MyCoolModule/JavaScript/MyCoolModule.Collection.js",
        "Modules/MyCoolModule/JavaScript/MyCoolModule.Model.js",
        "Modules/AdditionalCoolModule/JavaScript/Acme.MyCoolExtension.AdditionalCoolMod
ule.js",
        "Modules/AdditionalCoolModule/JavaScript/AdditionalCoolModule.Router.js",
        "Modules/AdditionalCoolModule/JavaScript/AdditionalCoolModule.List.View.js",
        "Modules/AdditionalCoolModule/JavaScript/AdditionalCoolModule.Edit.View.js",
        "Modules/AdditionalCoolModule/JavaScript/AdditionalCoolModule.Collection.js",
        "Modules/AdditionalCoolModule/JavaScript/AdditionalCoolModule.Model.js"
    ]
},
"myaccount": {
    "files": [
        "Modules/MyCoolModule/JavaScript/Acme.MyCoolExtension.MyCoolModule.js",
        "Modules/MyCoolModule/JavaScript/MyCoolModule.Router.js",
        "Modules/MyCoolModule/JavaScript/MyCoolModule.List.View.js",
        "Modules/MyCoolModule/JavaScript/MyCoolModule.Edit.View.js",
        "Modules/MyCoolModule/JavaScript/MyCoolModule.Collection.js",
        "Modules/MyCoolModule/JavaScript/MyCoolModule.Model.js",
        "Modules/AdditionalCoolModule/JavaScript/Acme.MyCoolExtension.AdditionalCoolMod
ule.js",
        "Modules/AdditionalCoolModule/JavaScript/AdditionalCoolModule.Router.js",
        "Modules/AdditionalCoolModule/JavaScript/AdditionalCoolModule.List.View.js",
        "Modules/AdditionalCoolModule/JavaScript/AdditionalCoolModule.Edit.View.js",
        "Modules/AdditionalCoolModule/JavaScript/AdditionalCoolModule.Collection.js",
        "Modules/AdditionalCoolModule/JavaScript/AdditionalCoolModule.Model.js"
    ]
},
"checkout": {
    "files": [
        "Modules/MyCoolModule/JavaScript/Acme.MyCoolExtension.MyCoolModule.js",
        "Modules/MyCoolModule/JavaScript/MyCoolModule.Router.js",
        "Modules/MyCoolModule/JavaScript/MyCoolModule.List.View.js",
        "Modules/MyCoolModule/JavaScript/MyCoolModule.Edit.View.js",
        "Modules/MyCoolModule/JavaScript/MyCoolModule.Collection.js",
        "Modules/MyCoolModule/JavaScript/MyCoolModule.Model.js",
        "Modules/AdditionalCoolModule/JavaScript/Acme.MyCoolExtension.AdditionalCoolMod
ule.js",
        "Modules/AdditionalCoolModule/JavaScript/AdditionalCoolModule.Router.js",
        "Modules/AdditionalCoolModule/JavaScript/AdditionalCoolModule.List.View.js",
        "Modules/AdditionalCoolModule/JavaScript/AdditionalCoolModule.Edit.View.js",
        "Modules/AdditionalCoolModule/JavaScript/AdditionalCoolModule.Collection.js",
        "Modules/AdditionalCoolModule/JavaScript/AdditionalCoolModule.Model.js"
    ]
}
}
//...

```

Ssp-libraries

The `ssp-libraries` object declares the paths to all SuiteScript files to be loaded when you deploy.

```
//...
"ssp-libraries": {
  "entry_point": "Modules/MyCoolModule/SuiteScript/Acme.MyCoolExtension.MyCoolModule.js",
  "files": [
    "Modules/MyCoolModule/SuiteScript/Acme.MyCoolExtension.MyCoolModule.js",
    "Modules/MyCoolModule/SuiteScript/MyCoolModule.ServiceController.js",
    "Modules/MyCoolModule/SuiteScript/MyCoolModule.Model.js",
    "Modules/AdditionalCoolModule/SuiteScript/Acme.MyCoolExtension.AdditionalCoolModule.js"
  ],
  "Modules/AdditionalCoolModule/SuiteScript/AdditionalCoolModule.ServiceController.js",
  "Modules/AdditionalCoolModule/SuiteScript/AdditionalCoolModule.Model.js"
}
//...
```


API Guide

Extensibility API How-To Topics

Localize Text in an Extension

This section describes how to configure and manage localized text in an extension. An extension can include dictionary files with localized text, one file per language. When a specific language is set for a website, string literals specified in the extension code are replaced with language-specific strings, which appear on the site.

Use the Extensibility API to replace string literals with the localized version in JavaScript files on the frontend and in SuiteScript files on the backend. Use the Handlebars `translate` custom helper to replace text with the localized version in template files. In addition, the Extensibility API includes a method to add additional translated string literals to the language-specific dictionary files.

The translated string literals are retrieved from name-value pairs defined in the `SC.Translations` object literal within the dictionary files, located in `[need location]`.

For more information about localization in SCS and SCA, see the help topic [Localization](#).

To configure and manage localized text in an extension:

1. Enable your site for localized content and set up the desired languages. For more information, see the help topic [Set Up NetSuite](#).

(include more information about what happens when you do this - i.e. which files are created and where - waiting on info from S.Gurin). Each dictionary file follows the naming convention `<language>_<LOCALE>.js`.

2. To direct the Extensibility API to translate text in a JavaScript or SuiteScript file:
 1. Locate the desired text to translate. For example, you want the string 'Extension' in the following code example translated:

```
return Backbone.View.extend({
  template: mine_extension_module_list_tpl
  , title: 'Extension'
  , events: {
    'click [data-action="remove"]': 'removeModule'
  }
  ...
})
```

2. Use the `translate()` method for the string 'Extension':

```
return Backbone.View.extend({
  template: mine_extension_module_list_tpl
  , title: _('Extension').translate()
  , events: {
    'click [data-action="remove"]': 'removeModule'
  }
  ...
})
```

The `translate()` method has the following syntax:

```
_'<string {0} {1}>'.translate(0, 1)
```

In the above syntax, `<string>` is the string value to translate and 0 and 1 are optional parameters for values within `<string>`. Parameter values are optional but you can include an unlimited number, as required.

Note: You can also call the `translate()` method directly from the `Utils` object, for example, `Utils.translate('Extension')`.

3. To translate text in a template file, use the NetSuite-provided Handlebars `translate` custom helper in the template file.

For example, a template file contains the following code with the variable `promotion_text`:

```
...
<span>
  {{promotion_text}}
</span>
...
```

To translate the variable `promotion_text`, use the `translate` custom helper:

```
...
<span>
  {{translate promotion_text}}
</span>
...
```

The `translate` custom helper uses the `_.translate()` function in `HandlebarsExtra.js`, which is located in the `JavaScript` subdirectory of the `HandlebarsExtras` application module. Similar to the `translate()` method in the `Utils` class, you can use parameters in the `translate` custom helper, for example:

```
{{translate 'There are ${0} apples in the tree' appleCount}}
```

3. The `SC.Translations` array, located within `[need location]`, includes the name-value pairs that contain translated values for the active language. To add additional values to this array in the extension code, use the `EnvironmentComponent.setTranslation(locale, keys)` method in the Extensibility API.

The following example shows the contents of the `SC.Translations` array in the `es_ES.js` dictionary file:


```
SC.Translations={
  "${0} Product": "${0} producto",
  "${0} Products": "${0} productos",
  ...
  "or": "o",
  "Use Current Location": "Usar ubicaciyn actual"
};
```

Use the `setTranslation()` method to add a new translated name-value pair. For example:

```
environmentComponent.setTranslation('es_ES', [{key: 'Faster than light', value: 'M6s r6pido que la luz'}])
```

The following code example shows the contents of the `es_ES.js` dictionary file after the method adds the translation:

```
SC.Translations={
  "$() Product": "$() producto",
  "$() Products": "$() productos",
  ...
  "or": "o",
  "Use Current Location": "Usar ubicaciyn actual",
  "Faster than light": "M6s r6pido que la luz"
}
```

 **Note:** You can also manually edit the dictionary file in `[location]` to add additional translated strings.

Other

Notes to reviewers

- I am assuming the process works basically the same for extensions as it did for the component-based architecture (on the frontend) and I am assuming the links in this topics will remain valid for 18.1. Am checking with dev.
- Organizationally, I am going to include the how-to's in the help in the order you might use them in the development process, so views will come before translation, for example. some might be a judgement call (like CCTs are sort of a completely separate process). The overview for this section will include a table that lists the available topics with a brief description.
- I need to meet with Damian and figure out how to start generating consistent code samples. Keith wants to do them in the style of the way CCTs and patches were done (where you can download the code samples used in the topics)
- need better (real-world) examples - going to bring this up with Damian

Outstanding questions

- Why is the underscore mixing syntax [<http://docs.commerceapplication.com/ncube8/apidocs/frontend/Utils.html#translate>] included? Should we just stick with the new syntax (if you should use it anyway)? Is there a benefit to mentioning both?
- Where are the files like `es_ES.js` (that contains `SC.Translations` array) located for an extension?

Access a Component

If you require your extension to access a specific component, for example, the Product Details Page or Product List Page, you must first access the component. To access a component and create the instance, pass the component identifier to the main container object `ComponentContainer`. You can then verify that the component exists and return the instance to the rest of the extension code, including routers, models, collections, and views.

The following table lists the available components and their associated identifiers:

Component Name	Identifier	Location
CartComponent	Cart	Frontend
CheckoutComponent	Checkout	Frontend
CMSComponent	CMS	Frontend
EnvironmentComponent	Environment	Frontend
ProductDetailsPageComponent	PDP	Frontend
ProductListPageComponent	PLP	Backend

To access the frontend and backend components

1. Access a front-end component in the `mountToApp` function in the entry point for your extension. By default, the entry point file is located in the JavaScript subdirectory in your extension code and uses the following naming convention:

```
<vendor name>.<extension name>.<version>.<module name>.js
```

Access a component with the `ComponentContainer.getComponent(component_name)` method. The following example shows how to access any one of the available components and then validate that it exists before returning it to the rest of your extension code:

```
define(
  'Vendor.ExtensionName.1.0.0.Extension'
, [
  'Vendor.ExtensionName.1.0.0.ExtensionName.Model'
, 'Vendor.ExtensionName.1.0.0.Extension.View'
]
, function (
  ExtensionModel
, ExtensionView
)
{
  'use strict';
  return {
    mountToApp: function mountToApp (container)
    {

      // Get the pdp component
      var pdp = container.getComponent('PDP');

      // Get the cart component
      var cart = container.getComponent('Cart');

      // Get the PLP component
      var plp = container.getComponent('PLP');

      // Get the checkout component
      var checkout = container.getComponent('Checkout');

      // Get the Environment component
      var env = container.getComponent('Environment');
```

```
// Get the CMS Component - Only for registering a component
var cms = container.getComponent('CMS');

if(pdp)
{
    // Use the pdp component here or pass it to the rest of the views and models
    pdp.addChildViews();

    // Pass the component instance to the model options
    new ExtensionModel({pdp: pdp});
}
}
};
});
```

2. Access the `BackEndCartComponent` component anywhere in the SuiteScript code, using the `Application.getComponent(component_name)` method. Currently, only the `BackEndCartComponent` component is available to SuiteScript code.

The following example shows how to use `getComponent()` to access the `BackEndCartComponent` instance:

```
define('Vendor.ExtensionName.1.0.0.Extension.File'
, [
    'Application'
    , 'underscore'
]
, function(
    Application
    , _
)
{
    'use strict';

    //Get the cart component
    var cart_component = Application.getComponent('Cart');
});
```

Accessing a Back-End Component

Note that you must include `Application` as a dependency.

Other

Notes to reviewers

- The API classes/component names and methods needs links, but if we are doing jsdocs for the API reference, then I do not have the links yet
- This topic is based on the following confluence page: <https://confluence.corp.netsuite.com/display/~nalbarengue/Extensibility+API+-+How+to+access+a+Component#ExtensibilityAPI-HowtoaccessaComponent-Componentsavailablecurrently>

This page also includes information (under What a Component Is) that more properly belongs in one of the conceptual, overview topics.

- Also, I/we need to figure out if we want to stick with the front-end/back-end terms, or use something else. We need to somehow distinguish between code in the JavaScript and SuiteScript directories.

Outstanding questions

- // I'm passing the component instance to my model options `new ExtensionModel({pdp: pdp});`
What exactly is this doing? why would you want to do this? this extension has/would have a custom model?
- (from confluence) Make sure that you are in the correct ssp application (how? -isnt that specified in manifest.json?) and that your component exists (what constitutes 'existing?') before following with the rest of your extension's logic.

SuiteCommerce Front-end JavaScript APIs

The SuiteCommerce Front-end JavaScript APIs include the following classes:

- BaseComponent
- CancelableEvents
- CartComponent
- CMSComponent
- ComponentContainer
- ExtensionEntryPoint
- ProductDetailsComponent
- ProductListPageComponent
- Utils
- View
- VisualComponent



Note: This documentation is in progress. For the JsDoc version of this documentation, see [SuiteCommerce Front-end JavaScript APIs](#).

BaseComponent

Description	Base abstract class for front-end components. Use the <code>BaseComponent.extend(componentDefinition)</code> method to build concrete components.
Extends	<code>CancelableEvents</code>
Methods	<ul style="list-style-type: none"> <code>BaseComponent.extend(componentDefinition)</code> <code>BaseComponent.on(event_name, handler)</code> <code>BaseComponent.off(event_name, handler)</code>
Members	<ul style="list-style-type: none"> <code>application</code> <code>componentName</code>

Members

application

Member description	The name which identify this kind of component. This name is used both for registering a new component and getting a component implementation with ComponentContainer .
Type	ComponentContainer

componentName

Member description	The name which identify this kind of component. This name is used both for registering a new component and getting a component implementation with ComponentContainer .
Type	String

Methods

BaseComponent.extend(componentDefinition)

Description	Extends the current component to generate a child component.
Parameters	<ul style="list-style-type: none"> ■ <code>componentDefinition {Object}</code> — Any object with properties/methods that will be used to generate the Component that will be returned.
Returns	BaseComponent

BaseComponent.on(event_name, handler)

Description	Attaches an event handler to a specific event name. Alias for CancelableEvents.cancelableOn(event_name, handler) .
Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to attach to ■ <code>handler {function}</code> —
Returns	void

BaseComponent.off(event_name, handler)

Description	Detaches an event handler from a specific event name. Alias for CancelableEvents.cancelableOff(event_name, handler) .
-------------	---

Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to detach from. Required. ■ <code>handler {function}</code> —
Returns	<code>void</code>

CMSComponent

Description	This component allows to interact with Site Management Tools. Right now it only allow a CCT implementer to register new CCT types when the application starts. Obtain an instance of this component by calling <code>container.getComponent("configuration")</code> .
Extends	BaseComponent
Methods	<ul style="list-style-type: none"> ■ <code>TODO</code>
Members	<ul style="list-style-type: none"> ■ <code>TODO</code>

Members

application

Description	The name which identify this kind of component. This name is used both for registering a new component and getting a component implementation with ComponentContainer .
Type	ComponentContainer

componentName

Description	The name which identify this kind of component. This name is used both for registering a new component and getting a component implementation with ComponentContainer .
Type	<code>String</code>

Methods

CMSComponent.registerCustomContentType(cctView)

Description	Register a new CCT in the current application. CCTs implementers need to call this o when appl.
--------------------	---

Parameters	<ul style="list-style-type: none"> ■ <code>cctView {*}</code> — Custom Content Type View constructor TODO type?.
Returns	<code>void</code>

CMSComponent.extend(componentDefinition)

Description	Extends the current component to generate a child component.
Parameters	<ul style="list-style-type: none"> ■ <code>componentDefinition {Object}</code> — Any object with properties/methods that will be used to generate the component that will be returned.
Returns	<code>BaseComponent</code>

CMSComponent.on(event_name, handler)

Description	Allow to attach an event handler into a particular event name. Alias for <code>CMSComponent.cancelableOn(event_name, handler)</code> .
Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to attach to. ■ <code>handler {function}</code> —
Returns	<code>void</code>

CMSComponent.off(event_name, handler)

Description	Allow to detach an event handler. Alias for <code>CMSComponent.cancelableOff(event_name, handler)</code> .
Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to detach from. Note that this parameter is mandatory. ■ <code>handler {function}</code> —
Returns	<code>void</code>

CMSComponent.cancelableOn(event_name, handler)

Description	Allow to attach an event handler into a particular event name.
Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to attach to. ■ <code>handler {function}</code> — The event handler function that will be invoked when the event <code>event_name</code> is triggered. This function can receive optionally one parameter representing the action parameter. Besides optionally can return a <code>Deferred</code> to details the execution of the trigger's callback. If the returned <code>Deferred</code> is rejected the trigger's callback won't be called.

Returns void

CMSSComponent.cancelableOff(event_name, handler)

Description Allow to detach an event handler..

Parameters

- `event_name {String}` — The name of the event to detach from. Note that this parameter is mandatory.
- `handler {function}` — The event handler function that will be removed from the list of register handlers. Note this parameter is required..

Returns TODO

CMSSComponent.cancelableDisable(event_name)

Description Allow to disable all the handlers of a particular event.

Inherits From [CancelableEvents.cancelableDisable\(event_name\)](#)

Parameters

- `event_name {String}` — The name of the event to disable.

Returns void

CMSSComponent.cancelableEnable(event_name)

Description Allow to enable (restore) all the handlers of a particular event.

Inherits From [CMSSComponent.cancelableEnable\(event_name\)](#)

Parameters

- `event_name {String}` — The name of the event to disable.

Returns void

CMSSComponent.cancelableTrigger(event_name, ...args)

Description Trigger the indicate event with the passed in parameters. In case that any of the event handler reject the execution (the callback won't be called). As the event handler of an event can be asynchronous (that why it use Deferrers) the invocation of the callback is async. This means that this function will return Deferred to represent this asynchronous.

Inherited from [CancelableEvents.cancelableTrigger\(event_name, ...args\)](#)

Parameters

- `event_name {String}` — Event to trigger.
- `args {params}` — All other parameter passed to this function will be broadcaster to all event handlers.

Returns Deferred

CMSSComponent.cancelableTriggerUnsafe(event_name, ...args)

Description	Trigger the indicate event with the passed in parameters WITHOUT sanitize them. In case that any of the event handler reject (returns a rejected Deferred) the execution (the callback won't be called). As the event handler of an event can be asynchronous (that why it use Deferrers) the invocation of the callback is async. This means that this function will return Deferred to represent this asynchronous
Inherited from	CancelableEvents.cancelableTriggerUnsafe(event_name, ...args)
Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — Event to trigger. ■ <code>args {params}</code> — All other parameter passed to this function will be broadcaster to all event handlers.
Returns	Deferred

CancelableEvents

Description	Cancelable events adds canceling functionality to the Observer pattern so Listeners can notify the Subject to cancel the actual action. Particularly, in SuiteCommerce extension APIs, for each action, two different events are triggered, one before the action happen and other after the action happen. In the case of before events, listeners can cancel the execution of the actual action by throwing an Error or by returning a rejected {@link Deferred} instance.
Methods	<ul style="list-style-type: none"> ■ CancelableEvents.cancelableOn(event_name, handler) ■ CancelableEvents.cancelableOff(event_name, handler) ■ CancelableEvents.cancelableEnable(event_name) ■ CancelableEvents.cancelableDisable(event_name) ■ CancelableEvents.cancelableTrigger(event_name, ...args) ■ CancelableEvents.cancelableTriggerUnsafe(event_name, ...args)

How To Cancel Events

In the case of before events, listeners can cancel the execution of the actual action by throwing an Error or by returning a rejected {@link Deferred} instance.

This is specially useful when doing validations, being able to call an external service to validate the input, and canceling if the validation fails.

Canceling an option selection from a listener by throwing an error:

```
pdp_component.on('beforeOptionSelection', function()
{
    if(someCondition) throw new Error('Canceling select option.');
```

Canceling an option selection from a listener by returning a rejected Promise:

```
pdp_component.on('beforeOptionSelection', function()
{
    if(someCondition) return jQuery.Deferred().reject(new Error('Canceling select option.'));
});
```

Methods

CancelableEvents.cancelableOn(event_name, handler)

Description	Allow to attach an event handler into a particular event name.
Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to attach to. ■ <code>handler {function}</code> — The event handler function that will be invoked when the event event_name is triggered. This function can receive optionally one parameter representing the action parameter. Besides optionally can return a Deferred to details the execution of the trigger's callback. If the returned Deferred is rejected the trigger's callback won't be called.
Returns	void Deferred

CancelableEvents.cancelableOff(event_name, handler)

Description	Allow to detach an event handler.
Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to detach from. Note that this parameter is mandatory. ■ <code>handler {function}</code> — The event handler function that will be removed from the list of register handlers. Note this parameter is required..
Returns	void

CancelableEvents.cancelableDisable(event_name)

Description	Disables all the handlers of a particular event.
Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to disable.
Returns	void

CancelableEvents.cancelableEnable(event_name)

Description	Allow to enable (restore) all the handlers of a particular event.
Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to disable.
Returns	void

CancelableEvents.cancelableTrigger(event_name, ...args)

Description	Trigger the indicate event with the passed in parameters. In case that any of the event handler reject the execution (the callback won't be called).
Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — Event to trigger. ■ <code>args {repeatable}</code> — All other parameter passed to this function will be broadcaster to all event handlers.
Returns	<code>{@link Deferred}</code>

CancelableEvents.cancelableTriggerUnsafe(event_name, ...args)

Description	Trigger the indicate event with the passed in parameters WITHOUT sanitize them. In case that any of the event handler reject (returns a rejected Deferred) the execution (the callback won't be called).
Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — Event to trigger. ■ <code>args {repeatable}</code> — All other parameter passed to this function will be broadcaster to all event handlers.
Returns	<code>{@link Deferred}</code>

CartComponent

Description	Provides methods for manipulating cart's lines, summary, estimates, promotions, submit. It also expose cancelable events that will be triggered before and after lines, estimates, promotions, etc change. An instance of this component can obtained by calling <code>container.getComponent("cart")</code> .
Extends	VisualComponent
Methods	<ul style="list-style-type: none"> ■ TO DO: finish
Members	<ul style="list-style-type: none"> ■ CART_VIEW ■ CART_MINI_VIEW ■ WIZARD_VIEW
Events	<ul style="list-style-type: none"> ■ TO DO: finish

Methods

CartComponent.addChildView(view_id, childViewConstuctor)

Description	Adds a child view to an existing View which is already appended in the DOM with the given <code>data-view</code> HTML attribute. Example:
--------------------	---

```
checkout.addChildView('Wizard.StepNavigation', function () {
```

```
return new CheckoutExtensionView({checkout:checkout});
});
```

Inherits From[VisualComponent.addChildView\(view_id, childViewConstructor\)](#)**Parameters**

- `view_id {string}` — The identifier of the view, of the current component, that will be extended with an extra child view.
- `childViewConstructor {SimpleChildViewConstructor}` — Identifier of the location where the new view will be located inside the specified View (`view_id`).

Returns

void

CartComponent.addChildViews(view_id, child_views)

Description

Adds one or more child views to existing Views which is already appended in the DOM with the given `data-view` HTML attribute. Notice that this is a very flexible method but also very complex. Users rarely will need such flexibility and in general they should use [CartComponent.addChildView\(view_id, childViewConstructor\)](#) which is simpler.

Usage example:

```
checkout.addChildViews(
  checkout.WIZARD_VIEW
  , {
    'Wizard.StepNavigation':
    {
      'CheckoutView':
      {
        childViewIndex: 1
        , childViewConstructor: function ()
        {
          return new CheckoutExtensionView({checkout:checkout});
        }
      }
    }
  }
);
```

Parameters

- `view_id {string}` — The identifier of the view, of the current component, that will be extended with an extra/s child view/s.
- `child_views {object}` — Identifier of the location where the new view will be located inside the specified View (`view_id`).

Returns

void

CartComponent.addLine(data)

Description

Adds a new line into the cart. Returns a Deferred that is resolved into the added line id {@link String}, in the case the operation was done successfully, or rejected with an error message..

Parameters	<ul style="list-style-type: none"> ■ <code>data {Object}</code> —
Returns	<code>{@link Deferred}</code>

CartComponent.addLines(lines)

Description	Adds new lines into the cart. Returns a Deferred that is resolved with the an array of line ids (Array of strings) into the added line id in the case the operation or rejected with an error message.
Parameters	<ul style="list-style-type: none"> ■ <code>lines {Object}</code> — needs description
Returns	<code>{@link Deferred}</code>

CartComponent.addPayment(TODO)

Description	Adds a payment method //TODO: returns? - Deferred resolved with ? //TODO: data parameter ? what's the structure?.
Returns	TODO

CartComponent.addPromotion(data)

Description	Adds a promotion.
Parameters	<ul style="list-style-type: none"> ■ <code>data {Object}</code> — the promocode to add.
Returns	TODO

CartComponent.addToViewContextDefinition(view_id, property_name, type, callback)

Description	Adds an extra property to the UI context of a view id to extend the interaction with its template.
Inherits from	VisualComponent.addToViewContextDefinition(view_id, property_name, type, callback)
Parameters	<ul style="list-style-type: none"> ■ <code>view_id {string}</code> — The identifier of the view, of the current component, that will have its context extended.. ■ <code>property_name {string}</code> — Name of the new property to be added. ■ <code>type {string}</code> — Name of the type of the result of the callback (function that generates the value of the new property). ■ <code>callback {function}</code> — Function in charge of generating the value for the new property..

Returns void | null if everything works as expected. An exception will be thrown otherwise.

CartComponent.addToViewEventsDefinition(view_id, event_selector, callback)

Description Allows to add an extra event handler over a particular view for the given event selector.

Inherits from [VisualComponent.addToViewEventsDefinition\(view_id, event_selector, callback\)](#)

Parameters

- `view_id {string}` — The identifier of the view, of the current component, that will be extended with an extra event handler..
- `event_selector {string}` —
- `callback {function}` — Event handler function called when the specified event occurs.

Returns void | null if everything works as expected. An exception will be thrown otherwise.

CartComponent.clearEstimateShipping(TODO)

Description Clears the shopping estimations in the cart

Parameters @params TODO

Returns @returns TODO

CartComponent.estimateShipping(locationData)

Description Returns the estimated shipping costs.

Parameters ■ `locationData {Object}` — TODO: returns ?.

Returns @returns TODO

CartComponent.getAddresses()

Description Returns the addresses of the order.
Return a jQuery Deferred that is resolved with an array of {@link Address} in the case the operation was done successfully or the promise is rejected with an error message.

Returns {@link Deferred}

CartComponent.getBillAddress()

Description	Returns the billing address of the order. Return a jQuery Deferred that is resolved with an {@link Address} in the case the operation was done successfully or the promise is rejected with an error message.
Returns	{@link Deferred}

CartComponent.getLines()

Description	Returns the lines of the cart as a {@link Deferred} that is resolved in the case the operation was done successfully or the promise is rejected with an error message. The promise resolves with an array of {@link Line}.
Returns	{@link Deferred}

CartComponent.getPaymentMethods()

Description	Returns the payment methods added to the order. TODO: as what object
Returns	{@link Deferred}

CartComponent.getPromotions()

Description	Returns the promotions' codes added to the cart. Returns a {@link Deferred} that is resolved with an array of {@link Promotion} in the case the operation was done successfully or rejected with an error message.
Returns	{@link Deferred}

CartComponent.getShipAddress()

Description	Returns the shipping address of the order. Return a jQuery {@link Deferred} that is resolved with an {@link Address} in the case the operation was done successfully or the promise is rejected with an error message.
Returns	{@link Deferred}

CartComponent.getShipMethod()

Description	Returns the ship methods of the order.
--------------------	--

Return a jQuery {@link Deferred} that is resolved with an {@link ShipMethod} in the case the operation was done successfully or the promise is rejected with an error message.

Returns {@link Deferred}

CartComponent.getShipMethods()

Description Returns the ship methods of the order.
Return a jQuery Deferred that is resolved with an {@link ShipMethod} in the case the operation was done successfully or the promise is rejected with an error message.

Returns {@link Deferred}

CartComponent.getSummary()

Description Returns the summary of the cart as a Deferred that is resolved with a {@link Summary} in the case the operation was done successfully or rejected with an error message.

Returns {@link Deferred}

CartComponent.removeChildView(view_id, placeholder_selector, view_nameopt)

Description Removes a child view for a given view id.

Inherits From [VisualComponent.removeChildView\(view_id, placeholder_selector, view_name\)](#)

Parameters

- `view_id {string}` — The identifier of the view, of the current component, that will be extended with an extra child view.
- `placeholder_selector {string}` — Identifier of the location where the new view will be located inside the specified View (view_id).
- `view_name {string}` — Identifier of an specific view into the placeholder.

Returns void | null if everything works as expected. An exception will be thrown otherwise.

CartComponent.removeLine(internalid)

Description Removes a line from the cart. Returns a Deferred that is resolved in the case the operation was done successfully or rejected with an error message.

Parameters ■ `internalid {String}` — id of the line to remove.

Returns {@link Deferred}

CartComponent.removePromotion(data)

Description Removes a promotion.

Parameters ■ `data {Object}` — TODO: what object

Returns {@link Deferred}

CartComponent.removeShipping(TODO)

Description Removes the shipping method.

Parameters @params TODO

Returns @returns TODO

CartComponent.removeToViewContextDefinition(view_id, property_name)

Description Removes an extra property from the UI context of a view.

Inherits from [VisualComponent.removeToViewContextDefinition\(view_id, property_name\)](#)

Parameters ■ `view_id {string}` — The identifier of the view, of the current component, that will have its context extended.

 ■ `property_name {string}` — Name of the new property to be added.

Throws Error

Returns void | null if everything works as expected. An exception will be thrown otherwise.

CartComponent.removeToViewEventsDefinition(view_id, event_selector)

Description Allows to remove and an extra event handler added previously..

Inherits From [VisualComponent.removeToViewEventsDefinition\(view_id, event_selector\)](#)

Parameters ■ `view_id {string}` — The identifier of the view, of the current component..

 ■ `event_selector {string}`

Throws Error

Returns void

CartComponent.setChildIndex(view_id, placeholder_selector, view_name, index)

Description	Change the position of a Child View inside a container. Returns null if everything works as expected. An exception will be thrown otherwise.
Parameters	<ul style="list-style-type: none"> ■ <code>view_id {string}</code> — The identifier of the view, of the current component, that will have the Child View to change the index. ■ <code>placeholder_selector {string}</code> — Identifier of the location where the view is located inside the specified View (<code>view_id</code>). ■ <code>view_name {string}</code> — Identifier of an specific view into the placeholder. ■ <code>index {number}</code> — The new index to position the Child View.
Inherits From	VisualComponent.setChildIndex(view_id, placeholder_selector, view_name, index)
Throws	Error
Returns	void null

CartComponent.submit()

Description	Submits the order. Returns a {@link Deferred} that is resolved with a {@link ConfirmationSubmit} object in the case the operation was done successfully, or rejected with an error message.
Returns	{@link Deferred}

CartComponent.unvoidLine(TODO)

Description	Un-voids a line. Implemented only for SCIS
Parameters	@params TODO
Returns	@returns TODO

CartComponent.updateCustomer(TODO)

Description	Updates a customer data. Implemented only for SCIS
Parameters	@params TODO

Returns @returns TODO

CartComponent.updateLine(line)

Description Updates a cart's line. Returns a Deferred that is resolved with {@link Line} in the case the operation was done successfully, or rejected with an error message.

Parameters ■ `line {Line}` —

Returns {@link Deferred}

CartComponent.voidLine(TODO)

Description Voids a line. Implemented only for SCIS.

Parameters @params TODO

Returns @returns TODO

Members

CART_VIEW

Description Name of the cart main view. Use this name to reference views in methods like `CartComponent.addChildView(view_id, childViewConstructor)`, `CartComponent.addToViewContextDefinition(view_id, property_name, type, callback)`, etc.

Type String

CART_MINI_VIEW

Description Name of the mini-cart main. Please use this name to reference views in methods like `CartComponent.addChildView(view_id, childViewConstructor)`, `CartComponent.addToViewContextDefinition(view_id, property_name, type, callback)`, etc.

Type String

WIZARD_VIEW

Description TODO

Type String

Events

afterAddLine

Description Triggered after a new line is added in the cart TODO: type.

Parameters ■ TODO {boolean} — TODO.

Type @type TODO

afterAddPayment

Description Triggered after a payment method is added to the order TODO: type.

Parameters ■ TODO {boolean} — TODO.

Type @type TODO

afterAddPromotion

Description Triggered before a promotion is added to the cart

Parameters ■ TODO {boolean} — TODO.

Type @type TODO

afterClearEstimateShipping

Description Triggered after an estimate shipping is cleared in the cart .

Parameters ■ TODO {boolean} — <p>TODO.

Type @type TODO

afterEstimateShipping

Description Triggered after an estimate shipping is done in the cart TODO: type.

Parameters ■ TODO {boolean} — TODO.

Type @type TODO

afterRemoveLine

Description Triggered after a cart's line is removed

Parameters ■ TODO {boolean} — TODO.

Type @type TODO

afterRemovePromotion

Description Triggered after a promocode is removed from the car.

Parameters ■ TODO {boolean} — TODO.

Type @type TODO

afterSubmit

Description Triggered after the order is submitted TODO: type.

Parameters ■ TODO {boolean} — TODO.

Type @type TODO

afterUpdateLine

Description Triggered after a cart's line is updated See TODO: type.

Parameters ■ TODO {boolean} — TODO.

Type @type TODO

beforeAddLine

Description Cancelable event triggered before adding a new car's line. See [CancelableEvents](#).

Parameters ■ TODO {boolean} — TODO.

Type @type TODO

beforeAddPayment

Description Triggered before a payment method is added to the order. See [CancelableEvents](#).

Parameters ■ `TODO {boolean}` — `TODO`.

Type @type `TODO`

beforeAddPromotion

Description Triggered before a promotion is added to the cart. See [CancelableEvents](#).

Parameters ■ `TODO {boolean}` — `TODO`.

Type @type `TODO`

beforeClearEstimateShipping

Description Cancelable event triggered before clearing an estimate shipping in the cart. See [CancelableEvents](#).

Parameters ■ `TODO {boolean}` — `TODO`.

Type @type `TODO`

beforeEstimateShipping

Description Cancelable event triggered before doing an estimate shipping in the cart. See [CancelableEvents](#).

Parameters ■ `TODO {boolean}` — `TODO`.

Type @type `TODO`

beforeRemoveLine

Description Cancelable event triggered before a cart's line is removed. See [CancelableEvents](#).

Parameters ■ `TODO {boolean}` — `TODO`.

Type @type `TODO`

beforeRemovePromotion

Description	Triggered before a promocode is removed from the cart. See CancelableEvents .
Parameters	<ul style="list-style-type: none"> ■ <code>TODO {boolean}</code> — TODO.
Type	@type TODO

beforeSubmit

Description	Triggered before the order is submitted. See CancelableEvents .
Parameters	<ul style="list-style-type: none"> ■ <code>TODO {boolean}</code> — TODO.
Type	@type TODO

beforeUpdateLine

Description	Cancelable event triggered before a cart's line is updated. See CancelableEvents .
Parameters	<ul style="list-style-type: none"> ■ <code>TODO {boolean}</code> — TODO.
Type	@type TODO

ComponentContainer

Description	Manager of components. Extensions can get components implementations and register new component classes. A component is referenced always by its name. Host application provides a container instance to extensions through method ExtensionEntryPoint.mountToApp(componentContainer) .
--------------------	---

Methods

Methods

ComponentContainer.getComponent(component_name)

Description	Returns the requested component based on its name if there is no component with that name registered in this container.
Parameters	<ul style="list-style-type: none"> ■ <code>component_name {String}</code> —TODO
Returns	{@link BaseComponent}

ComponentContainer.registerComponent(component)

Description	Allows to register a new component into the running application it also seals the component, so as to not add new properties or messing up with the available components APIs..
Parameters	<ul style="list-style-type: none"> ■ <code>component</code> {@link BaseComponent} — Component to be registered.
Returns	<code>void</code>

ExtensionEntryPoint

Description	An extension entry point is an object provided by component implementers like extensions or modules. Implement the {@link ExtensionEntryPoint#mountToApp} that is called by the host application that passes a {@link ComponentContainer} so they can obtain existing component instances to extend the application {@link ComponentContainer#getComponent} or register new component types {@link ComponentContainer#registerComponent} or register component {@link BaseComponent} implementations. See {@link ComponentContainer}
Functions	<ul style="list-style-type: none"> ■ ExtensionEntryPoint.mountToApp(componentContainer)

Methods

ExtensionEntryPoint.mountToApp(componentContainer)

Description	When the host application starts, it will call this method for each activated extension, in order of activation, pass the component container as parameter so extensions can get components to work with (see {@link ComponentContainer#getComponent}) or register new components (see {@link ComponentContainer#registerComponent})..
Parameters	<ul style="list-style-type: none"> ■ <code>componentContainer</code> {ComponentContainer} — TODO.
Returns	<code>void</code>

EnvironmentComponent

Description	Allows front-end accessing environment-related information like configuration, site settings, execution context and user session. This information is read-only, meaning that user changes in returned objects won't impact the application. An instance of this component can be obtained by calling <code>container.getComponent("Environment")</code> .
Extends	{@link BaseComponent}
Methods	<ul style="list-style-type: none"> ■ TODO: list
Members	<ul style="list-style-type: none"> ■ TODO: list

Methods

EnvironmentComponent.cancelableDisable(event_name)

Description Allow to disable all the handlers of a particular event.

Parameters ■ `event_name {String}` — The name of the event to disable.

Inherited from {@link CancelableEvents#cancelableDisable}

Returns void

EnvironmentComponent.cancelableEnable(event_name)

Description Allow to enable (restore) all the handlers of a particular event.

Parameters ■ `event_name {String}` — The name of the event to disable.

Inherited from {@link CancelableEvents#cancelableEnable}

Returns void

EnvironmentComponent.cancelableOff(event_name, handler)

Description Allow to detach an event handler..

Parameters

- `event_name {String}` — The name of the event to detach from. Note that this parameter is mandatory.
- `handler {function}` — The event handler function that will be removed from the list of register handlers. Note this parameter is required..

Inherited from {@link CancelableEvents#cancelableOff}

Returns void

EnvironmentComponent.cancelableOn(event_name, handler)

Description Allow to attach an event handler into a particular event name.

Parameters

- `event_name {String}` — The name of the event to attach to.
- `handler {function}` — The event handler function that will be invoked when the event **event_name** is triggered. This function can receive optionally one parameter representing the action parameter. Besides optionally can return a Deferred to details the execution of the trigger's callback. If the returned Deferred is rejected the trigger's callback won't be called.

Inherited from {@link CancelableEvents#cancelableOn}

Returns void

EnvironmentComponent.cancelableTrigger(event_name, ...args)

Description Trigger the indicate event with the passed in parameters. In case that any of the event handler reject the execution (the callback won't be called).

Parameters

- `event_name {String}` — Event to trigger.
- `args {params}` — All other parameter passed to this function will be broadcaster to all event handlers.

Inherited from {@link CancelableEvents#cancelableTrigger}

Returns {@link Deferred}

EnvironmentComponent.cancelableTriggerUnsafe(event_name, ...args)

Description Trigger the indicate event with the passed in parameters WITHOUT sanitize them. In case that any of the event handler reject (returns a rejected Deferred) the execution (the callback won't be called).

Parameters

- `event_name {String}` — Event to trigger.
- `args {params}` — All other parameter passed to this function will be broadcaster to all event handlers.

Inherited from {@link CancelableEvents#cancelableTriggerUnsafe}

Returns {@link Deferred}

EnvironmentComponent.extend(componentDefinition)

Description Extends the current component to generate a child one.

Parameters

- `componentDefinition {Object}` — Any object with properties/methods that will be used to generate the Component that will be returned.

Inherited from {@link CancelableEvents#extend}

Returns {@link BaseComponent}

EnvironmentComponent.getConfig(key)

Description Get the value of given Configuration key. If no key is provided then it returns the whole Configuration object. Use keys separated by dots to access inner objects, for example, `component.get('checkout.skipLogin')`. Notice that this method

will always return a copy of the real objects so modifications on them won't have impact on the application..

Parameters

- `key {String}` — (Optional) configuration key to get the value of. It could be dot-separated to get an inner property. If not passed the whole configuration object will be returned..

Returns any

EnvironmentComponent.getSession()

Description Returns information regarding the current session, like current currency, language and pricelevel.

Returns {@link Session}

EnvironmentComponent.getSiteSetting(key)

Description Returns the [site-settings object](https://system.netsuite.com/app/help/helpcenter.nl?fid=section_N2532617.html).

Parameters

- `key {String}` — site-settings key to get the value of. It could be dot-separated to get an inner property. If not passed the whole site-settings object will be returned..

Returns any

EnvironmentComponent.isPageGenerator()

Description Returns true if the code is currently run by the SEO Page Generator and false if it currently run by the user's browser. See [Page Generator NetSuite Help Center](https://system.netsuite.com/app/help/helpcenter.nl?fid=section_4053806622.html).

Returns Boolean

EnvironmentComponent.off(event_name, handler)

Description Allow to detach an event handler. Alias for {@link CancelableEvents#cancelableOff}.

Parameters

- `event_name {String}` — The name of the event to detach from. Note that this parameter is mandatory.
- `handler {function}` —

Inherited from {@link BaseComponent#off}

Returns void

EnvironmentComponent.on(event_name, handler)

Description Allow to attach an event handler into a particular event name. Alias for [{@link CancelableEvents#cancelableOn}](#).

Parameters

- `event_name {String}` — The name of the event to attach to.
- `handler {function}` —

Inherited from [{@link BaseComponent#on}](#)

Returns void

EnvironmentComponent.setTranslation(locale, keys)

Description Adds or update a translation key for given locale. Example:

```
setTranslation('es_ES', [{key: 'Faster than light', value: 'M6s rápido que la luz'}])
```

See [{@tutorial frontend_internationalization}](#).

Parameters

- `locale {string}` —
- `keys {Array.<TranslationEntry>}` — keys and values to add for given locale.

Returns TODO

Members

application

Description The name which identify this kind of component. This name is used both for registering a new component and getting a component implementation with [{@link ComponentContainer}](#).

Type ComponentContainer

componentName

Description The name which identify this kind of component. This name is used both for registering a new component and getting a component implementation with [{@link ComponentContainer}](#).

Type String

ProductDetailsComponent

Description	The ProductDetails component let's the user interact with most important aspect of the Product Details Page, like setting options, changing quantity, and obtain item's related information.
Extends	{@link VisualComponent}
Methods	<ul style="list-style-type: none"> ■ TODO: regenerate
Members	<ul style="list-style-type: none"> ■ PDP_FULL_VIEW ■ PDP_QUICK_VIEW
Events	<ul style="list-style-type: none"> ■ TODO: list

Methods

ProductDetailsComponent.addChildView(view_id, childViewConstructor)

Description	Adds a child view to an existing View which is already appended in the DOM with the given <code><code>data-view</code></code> HTML attribute. Example:>
--------------------	---

```
checkout.addChildView('Wizard.StepNavigation', function ()
{
    return new CheckoutExtensionView({checkout:checkout});
});
```

Inherited from	{@link VisualComponent#addChildView}
Parameters	<ul style="list-style-type: none"> ■ <code>view_id {string}</code> — The identifier of the view, of the current component, that will be extended with an extra/s child view/s. ■ <code>childViewConstructor {SimpleChildViewConstructor}</code> — Identifier of the location where the new view will be located inside the specified View (<code>view_id</code>).
Returns	void

ProductDetailsComponent.addChildViews(view_id, child_views)

Description	Adds one or more child views to existing Views which is already appended in the DOM with the given <code><code>data-view</code></code> HTML attribute. Notice that this is a very flexible method but also very complex. Users rarely will need such flexibility and in general they should use {@link addChildView} which is simpler. Usage example:
--------------------	---

```
checkout.addChildViews(
    checkout.WIZARD_VIEW
    , {
        'Wizard.StepNavigation':
```

```

    {
      'CheckoutView':
      {
        childViewIndex: 1
        , childViewConstructor: function ()
        {
          return new CheckoutExtensionView({checkout:checkout});
        }
      }
    }
  );

```

Returns null if everything works as expected. An exception will be thrown otherwise.

Inherited from {@link VisualComponent#addChildViews}

Parameters

- `view_id {string}` — The identifier of the view, of the current component, that will be extended with an extra/s child view/s.
- `child_views {object}` — Identifier of the location where the new view will be located inside the specified View (`view_id`).

Returns void

ProductDetailsComponent.addToViewContextDefinition(view_id, property_name, type, callback)

Description Adds an extra property to the UI context of a view id to extend the interaction with its template.

Inherited from {@link VisualComponent#addToViewContextDefinition}

Parameters

- `view_id {string}` — The identifier of the view, of the current component, that will have its context extended..
- `property_name {string}` — Name of the new property to be added.
- `type {string}` — Name of the type of the result of the callback (function that generates the value of the new property).
- `callback {function}` — Function in charge of generating the value for the new property..

Returns void

ProductDetailsComponent.addToViewEventsDefinition(view_id, event_selector, callback)

Description Allows to add an extra event handler over a particular view for the given event selector.

Inherited from	<code>{@link VisualComponent.addToViewEventsDefinition}</code>
Parameters	<ul style="list-style-type: none"> ■ <code>view_id {string}</code> — The identifier of the view, of the current component, that will be extended with an extra event handler.. ■ <code>event_selector {string}</code> — ■ <code>callback {function}</code> — Event handler function called when the specified event occurs.
Returns	<code>void</code>

ProductDetailsComponent.getItemInfo()

Description	If the current View is a PDP it will return a representation of the transaction line represented in the current pdp, null otherwise. Notice that there could be some dynamic - user - configured extra fields..
Returns	<code>{@link ItemInfo}</code>

ProductDetailsComponent.removeChildView(view_id, placeholder_selector, view_name)

Description	Removes a child view for a given view id.
Inherited from	<code>{@link VisualComponent.removeChildView}</code>
Parameters	<ul style="list-style-type: none"> ■ <code>view_id {string}</code> — The identifier of the view, of the current component, that will be extended with an extra child view. ■ <code>placeholder_selector {string}</code> — Identifier of the location where the new view will be located inside the specified View (view_id). ■ <code>view_name {string}</code> — (Optional) Identifier of an specific view into the placeholder.
Returns	<code>void</code>

ProductDetailsComponent.removeToViewContextDefinition(view_id, property_name)

Description	Removes an extra property to the UI context of a view..
Inherited from	<code>{@link VisualComponent.removeToViewContextDefinition}</code>
Parameters	<ul style="list-style-type: none"> ■ <code>view_id {string}</code> — The identifier of the view, of the current component, that will have its context extended.. ■ <code>property_name {string}</code> — Name of the new property to be added.

Returns void

ProductDetailsComponent.removeToViewEventsDefinition(view_id, event_selector)

Description Allows to remove and an extra event handler added previously..

Inherited from {@link VisualComponent.removeToViewEventsDefinition}

Parameters

- `view_id {string}` — The identifier of the view, of the current component..
- `event_selector {string}` —

Returns void

ProductDetailsComponent.setChildViewIndex(view_id, placeholder_selector, view_name, index)

Description Change the position of a Child View inside a container. Returns null if everything works as expected. An exception will be thrown otherwise.

Inherited from {@link VisualComponent.setChildViewIndex}

Parameters

- `view_id {string}` — The identifier of the view, of the current component, that will have the Child View to change the index.
- `placeholder_selector {string}` — Identifier of the location where the view is located inside the specified View (view_id).
- `view_name {string}` — Identifier of an specific view into the placeholder.
- `index {number}` — The new index to position the Child View.

Returns void

ProductDetailsComponent.setOption(cart_option_id, value)

Description Sets an option of the current PDP. It will only work if the current view is a Product Details Page.
Returns a Deferred that is resolved with true or false accordingly if the action was successful or not

Inherited from {@link VisualComponent.setOption}

Parameters

- `cart_option_id {String}` — Identifier of the option.
- `value {String}` — Optional value. In case of invalid or not specified value the option selected will be cleaned.

Returns {@link Deferred}

ProductDetailsComponent.setQuantity(quantity)

Description	Set the quantity of the current PDP. If the current View is a PDP. Resturns a Deferred that is resolved with true or false accordingly if the action was successful or not
Inherited from	{@link VisualComponent.setQuantity}
Parameters	<ul style="list-style-type: none"> ■ <code>quantity {Number}</code> —
Returns	{@link Deferred}

Members

PDP_FULL_VIEW

Description	Name of the PDP main full view. Please use this name to reference views in methods like {@link addChildViews}, {@link addToViewContextDefinition}, etc.
Type	String

PDP_QUICK_VIEW

Description	Name of the PDP quick-view. Please use this name to reference views in methods like {@link addChildViews}, {@link addToViewContextDefinition}, etc.
Type	String

Events

afterImageChange

Description	Triggered after the main image displayed in the Details page changes.
Parameters	<ul style="list-style-type: none"> ■ <code>TODO {boolean}</code> — TODO.
Type	TODO

afterOptionSelection

Description	Triggered after an option is selected. TODO: type.
Parameters	<ul style="list-style-type: none"> ■ <code>TODO {boolean}</code> — TODO.

Type `TODO`

afterQuantityChange

Description Triggered after the quantity is changed.

Parameters ■ `TODO {boolean}` — `TODO`.

Type `TODO`

beforeImageChange

Description Cancelable event triggered before the main image displayed in the Details page changes. See {@link CancelableEvents}

Parameters ■ `TODO {boolean}` — `TODO`.

Type `TODO`

beforeOptionSelection

Description Cancelable event triggered before an option is selected. See {@link CancelableEvents}

Parameters ■ `TODO {boolean}` — `TODO`.

Type `TODO`

beforeQuantityChange

Description Cancelable event triggered before the quantity is changed. See {@link CancelableEvents}

Parameters ■ `TODO {boolean}` — `TODO`.

Type `TODO`

ProductListPageComponent

Description The ProductListPage component let's the user interact with most important aspect of the Product List Page (a.k.a "search"), like setting options, pagination, filters, ordering, and obtain item's related information.

Note some methods of this class only make sense to be called when the current page is the "search" page. In other case, they will return a {@link Deferred} object resolved with false and print a warning message in the browser's console. An instance of this component can be obtained by calling `container.getComponent("PLP")`.

Extends	VisualComponent
Functions	<ul style="list-style-type: none"> ■ TODO: list
Properties	<ul style="list-style-type: none"> ■ PLP_VIEW

Methods

ProductListPageComponent.addChildView(view_id, childViewConstructor)

Description Adds a child view to an existing View which is already appended in the DOM with the given `data-view` HTML attribute. Example:

```
checkout.addChildView('Wizard.StepNavigation', function ()
{
    return new CheckoutExtensionView({checkout:checkout});
});
```

Returns null if everything works as expected. An exception will be thrown otherwise.

Inherited from [VisualComponent.addChildView\(view_id, childViewConstructor\)](#)

Parameters

- `view_id`{string} — The identifier of the view, of the current component, that will be extended with an extra/s child view/s.
- `childViewConstructor`{[@link SimpleChildViewConstructor](#)} — Identifier of the location where the new view will be located inside the specified View (`view_id`).

Throws Error

Returns void

ProductListPageComponent.addChildViews(view_id, child_views)

Description Adds one or more child views to existing Views which is already appended in the DOM with the given `data-view` HTML attribute. Notice that this is a very flexible method but also very complex. Users rarely will need such flexibility and in general they should use {@link addChildView} which is simpler. Usage example:

```
checkout.addChildViews(
    checkout.WIZARD_VIEW
```

```

    , {
      'Wizard.StepNavigation':
      {
        'CheckoutView':
        {
          childViewIndex: 1
          , childViewConstructor: function ()
          {
            return new CheckoutExtensionView({checkout:checkout});
          }
        }
      }
    }
  );

```

Returns null if everything works as expected. An exception will be thrown otherwise.

Inherited from

[VisualComponent.addChildViews\(view_id, child_views\)](#)

Parameters

- **view_id** {string} — The identifier of the view, of the current component, that will be extended with an extra child view.
- **child_views** {object} — Identifier of the location where the new view will be located inside the specified View (**view_id**).

Throws Error

Returns void

ProductListPageComponent.addToViewContextDefinition(view_id, property_name, type, callback)

Description

Adds an extra property to the UI context of a view id to extend the interaction with its template.
Returns null if everything works as expected. An exception will be thrown otherwise.

Inherited from

[VisualComponent.addToViewContextDefinition\(view_id, property_name, type, callback\)](#)

Parameters

- **view_id** {string} — The identifier of the view, of the current component, that will have its context extended..
- **property_name** {string} — Name of the new property to be added.
- **type** {string} — Name of the type of the result of the callback (function that generates the value of the new property).
- **callback** {function} — Function in charge of generating the value for the new property..

Throws Error

Returns void

ProductListPageComponent.addToViewEventsDefinition(view_id, event_selector, callback)

Description	Allows to add an extra event handler over a particular view for the given event selector. Returns null if everything works as expected. An exception will be thrown otherwise.
Inherited from	VisualComponent.addToViewEventsDefinition(view_id, event_selector, callback)
Parameters	<ul style="list-style-type: none"> ■ <code>view_id {string}</code> — The identifier of the view, of the current component, that will be extended with an extra event handler.. ■ <code>event_selector {string}</code> — ■ <code>callback {function}</code> — Event handler function called when the specified event occurs.
Throws	Error
Returns	void

ProductListPageComponent.getAllDisplay()

Description	Get the info of all the display option including which is the active one. Will only work if the current page is the search page..
Returns	Array.<{@link Display}>

ProductListPageComponent.getAllFilters()

Description	Get the info of all the filters - one of them is active. Will only work if the current page is the search page..
Returns	Array.<{@link Filter}>

getAllPageSize()

Description	Get the info of all the page size option. Will only work if the current page is the search page..
Returns	Array.<{@link PageSize}>

ProductListPageComponent.getAllSorting()

Description	Get the info of all the sorting option.
--------------------	---

Returns Array.<{@link Sorting}>

getDisplay()

Description Get the info of the active display option. Will only work if the current page is the search page..

Returns {@link Display}

ProductListPageComponent.getFilters()

Description Get the info of the active filters that are currently applied in the current search page. Will only work if the current page is the search page..

Returns Array.<{@link Filter}>

ProductListPageComponent.getItemsInfo()

Description Get the info of all the items in the PLP.

Parameters Array.<{@link SearchItemInfo}>

ProductListPageComponent.getPageSize()

Description Get the info of the active page size option. Will only work if the current page is the search page..

Parameters {@link PageSize}

ProductListPageComponent.getPagination()

Description Return the info of the current options in the PLP.

Returns {@link Pagination}

ProductListPageComponent.getSearchText()

Description get the current text query that is being search in the current search page or null if the current page is not the search page..

Returns String | null

ProductListPageComponent.getSorting()

Description	Get the info of the active sorting option. The current view must be the search page for this to work.
Returns	{@link Sorting} null

ProductListPageComponent.removeChildView(view_id, placeholder_selector, view_name)

Description	Removes a child view for a given view id. Returns null if everything works as expected. An exception will be thrown otherwise.
Inherited from	VisualComponent.removeChildView(view_id, placeholder_selector, view_name)
Parameters	<ul style="list-style-type: none"> ■ <code>view_id {string}</code> — The identifier of the view, of the current component, that will be extended with an extra child view. ■ <code>placeholder_selector {string}</code> — Identifier of the location where the new view will be located inside the specified View (view_id). ■ <code>view_name {string}</code> — Identifier of an specific view into the placeholder.
Throws	Error
Returns	void

ProductListPageComponent.removeToViewContextDefinition(view_id, property_name)

Description	Removes an extra property to the UI context of a view.. Returns null if everything works as expected. An exception will be thrown otherwise.
Inherited from	VisualComponent.removeToViewContextDefinition(view_id, property_name)
Parameters	<ul style="list-style-type: none"> ■ <code>view_id {string}</code> — The identifier of the view, of the current component, that will have its context extended.. ■ <code>property_name {string}</code> — Name of the new property to be added.
Throws	Error
Returns	void

ProductListPageComponent.removeToViewEventsDefinition(view_id, event_selector)

Description	Allows to remove and an extra event handler added previously..
--------------------	--

	Returns null if everything works as expected. An exception will be thrown otherwise.
Inherited from	VisualComponent.removeToViewEventsDefinition(view_id, event_selector)
Parameters	<ul style="list-style-type: none"> ■ <code>view_id {string}</code> — The identifier of the view, of the current component.. ■ <code>event_selector {string}</code> —
Throws	Error
Returns	void

ProductListPageComponent.setChildIndex(view_id, placeholder_selector, view_name, index)

Description	Change the position of a Child View inside a container. Returns null if everything works as expected. An exception will be thrown otherwise.
Inherited from	VisualComponent.setChildIndex(view_id, placeholder_selector, view_name, index)
Parameters	<ul style="list-style-type: none"> ■ <code>view_id {string}</code> — The identifier of the view, of the current component, that will have the Child View to change the index. ■ <code>placeholder_selector {string}</code> — Identifier of the location where the view is located inside the specified View (view_id). ■ <code>view_name {string}</code> — Identifier of an specific view into the placeholder. ■ <code>index {number}</code> — The new index to position the Child View.
Throws	Error
Returns	void

ProductListPageComponent.setDisplay(options)

Description	Set the display option in the PLP. Will only work if the current page is the search page.. Will resolve to false if the current page is not the search page or if there is an error - different than false otherwise.
Parameters	<ul style="list-style-type: none"> ■ <code>options {Display}</code> —
Returns	{@link Deferred}

ProductListPageComponent.setFilters(filters)

Description	Set the filters for the PLP.
-------------	------------------------------

Parameters ■ `filters`
 `{Array.<Filter>} —`

Returns `{@link Deferred}`

ProductListPageComponent.setPage(page)

Description Navigates to given page. The current view must be the search page for this to work..
 That will resolved with false if there is any error or with non-false value otherwise.

Parameters ■ `page {number} —`

Returns `{@link Deferred}`

ProductListPageComponent.setPageSize(options)

Description Set the page size in the current search page. Will only work if the current page is the search page..
 Resolves with false if there is any error or with non-false value otherwise.

Parameters ■ `options {PageSize} —`

Returns `{@link Deferred}`

ProductListPageComponent.setSearchText(options)

Description Triggers a search using given keywords in the current search page..
 Resolve to false if the current page is not the search page or if there is an error - different than false otherwise.

Parameters ■ `options {Object} —`

Returns `{@link Deferred}`

ProductListPageComponent.setSorting(sorting)

Description Sets the sorting of the current search page. Will only work if the current page is the search page.
 will resolve to false if the current page is not the search page or if there is an error - different than false otherwise.

Parameters ■ `sorting {Sorting}` —

Returns `{@link Deferred}`

Members

PLP_VIEW

Description Name of the cart main view. Use this name to reference views in methods like `{@link addChildViews}`, `{@link addToViewContextDefinition}`, etc.

Type `String`

VisualComponent

Description Base abstract class supporting components which include a visual aspect. For example, concrete components like `{@link ProductDetailsComponent}` or `{@link ProductListPageComponent}` allow users to manipulate its views and composition, so they inherit from here.

Among other things allow users to programatically manipulate child views, DOM event handlers, DOM input data bindings, child views context data, etc.

In general users needs to work against the [View](#) interface, and special HTML attributes like `data-view` for referencing DOM elements that will remain compatible (for example to insert a new child view in a certain place).

See TODO: `{@link How to Implement New Views}` to learn how to implement new Views.

See TODO: `{@link How to Manipulate Views}` to learn how to manipulate existing views and add new ones.

Inherits from [BaseComponent](#)

Functions

- `VisualComponent.addChildView(view_id, childViewConstructor)`
- `VisualComponent.addChildViews(view_id, child_views)`
- `VisualComponent.addToViewContextDefinition(view_id, property_name, type, callback)`
- `VisualComponent.addToViewEventsDefinition(view_id, event_selector, callback)`
- `VisualComponent.cancelableDisable(event_name)`
- `VisualComponent.cancelableEnable(event_name)`
- `VisualComponent.cancelableOff(event_name, handler)`
- `VisualComponent.cancelableOn(event_name, handler)`
- `VisualComponent.cancelableTrigger(event_name, ...args)`
- `VisualComponent.cancelableTriggerUnsafe(event_name, ...args)`
- `VisualComponent.extend(componentDefinition)(componentDefinition)`
- `VisualComponent.off(event_name, handler)`
- `VisualComponent.on(event_name, handler)`

- `VisualComponent.removeChildView(view_id, placeholder_selector, view_name)`
- `VisualComponent.removeToViewContextDefinition(view_id, property_name)`
- `VisualComponent.removeToViewEventsDefinition(view_id, event_selector)`
- `VisualComponent.setChildViewIndex(view_id, placeholder_selector, view_name, index)`

Properties

- `componentName`
- `application`

Methods

VisualComponent.addChildView(view_id, childViewConstructor)

Description

Adds a child view to an existing View which is already appended in the DOM with the given `data-view` HTML attribute. Example:

```
checkout.addChildView('Wizard.StepNavigation', function ()
{
    return new CheckoutExtensionView({checkout:checkout});
});
```

Parameters

- `view_id {string}` — The identifier of the view, of the current component, that will be extended with an extra/s child view/s.
- `childViewConstructor {SimpleChildViewConstructor}` — Identifier of the location where the new view will be located inside the specified View (`view_id`).

Returns `void`

VisualComponent.addChildViews(view_id, child_views)

Description

Adds one or more child views to existing Views which is already appended in the DOM with the given `<code>data-view</code>` HTML attribute. Notice that this is a very flexible method but also very complex. Users rarely will need such flexibility and in general they should use `{@link addChildView}` which is simpler. Usage example: ..

```
class="prettyprint source lang-javascript"><code>checkout.addChildViews( checkout.WIZARD_VIEW , { 'Wizard.StepNavigation': { 'CheckoutView': { childViewIndex: 1 , childViewConstructor: function () { return new CheckoutExtensionView({checkout:checkout}); } } } );</code></pre>
```

Parameters

- `view_id {string}` — The identifier of the view, of the current component, that will be extended with an extra/s child view/s.
- `child_views {object}` — Identifier of the location where the new view will be located inside the specified View (`view_id`).

Returns `void`

VisualComponent.addToViewContextDefinition(view_id, property_name, type, callback)

Description	Adds an extra property to the UI context of a view id to extend the interaction with its template.
Parameters	<ul style="list-style-type: none"> ■ <code>view_id {string}</code> — The identifier of the view, of the current component, that will have its context extended.. ■ <code>property_name {string}</code> — Name of the new property to be added. ■ <code>type {string}</code> — Name of the type of the result of the callback (function that generates the value of the new property). ■ <code>callback {function}</code> — Function in charge of generating the value for the new property..
Returns	<code>void</code>

VisualComponent.addToViewEventsDefinition(view_id, event_selector, callback)

Description	Allows to add an extra event handler over a particular view for the given event selector.
Parameters	<ul style="list-style-type: none"> ■ <code>view_id {string}</code> — The identifier of the view, of the current component, that will be extended with an extra event handler.. ■ <code>event_selector {string}</code> — ■ <code>callback {function}</code> — Event handler function called when the specified event occurs.
Returns	<code>void</code>

VisualComponent.cancelableDisable(event_name)

Description	Allow to disable all the handlers of a particular event.
Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to disable.
Returns	<code>void</code>

VisualComponent.cancelableEnable(event_name)

Description	Allow to enable (restore) all the handlers of a particular event.
Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to disable.

Returns void

VisualComponent.cancelableOff(event_name, handler)

Description Allow to detach an event handler..

Parameters

- `event_name {String}` — The name of the event to detach from. Note that this parameter is mandatory.
- `handler {function}` — The event handler function that will be removed from the list of register handlers. Note this parameter is required..

Returns void

VisualComponent.cancelableOn(event_name, handler)

Description Allow to attach an event handler into a particular event name.

Parameters

- `event_name {String}` — The name of the event to attach to.
- `handler {function}` — The event handler function that will be invoked when the event `event_name` is triggered. This function can receive optionally one parameter representing the action parameter. Besides optionally can return a `Deferred` to details the execution of the trigger's callback. If the returned `Deferred` is rejected the trigger's callback won't be called.

Returns void

VisualComponent.cancelableTrigger(event_name, ...args)

Description Trigger the indicate event with the passed in parameters. In case that any of the event handler reject the execution (the callback won't be called).

Parameters

- `event_name {String}` — Event to trigger.
- `args {params}` — All other parameter passed to this function will be broadcaster to all event handlers.

Returns {@link Deferred}

VisualComponent.cancelableTriggerUnsafe(event_name, ...args)

Description Trigger the indicate event with the passed in parameters WITHOUT sanitize them. In case that any of the event handler reject (returns a rejected `Deferred`) the execution (the callback won't be called).

Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — Event to trigger. ■ <code>args {params}</code> — All other parameter passed to this function will be broadcaster to all event handlers.
Returns	<code>{@link Deferred}</code>

VisualComponent.extend(componentDefinition) (componentDefinition)

Description	Extends the current component to generate a child one.
Parameters	<ul style="list-style-type: none"> ■ <code>componentDefinition {Object}</code> — Any object with properties/methods that will be used to generate the Component that will be returned.
Returns	<code>BaseComponent</code>

VisualComponent.off(event_name, handler)

Description	Allow to detach an event handler. Alias for <code>{@link CancelableEvents#cancelableOff}</code> .
Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to detach from. Note that this parameter is mandatory. ■ <code>handler {function}</code> —
Returns	<code>void</code>

VisualComponent.on (event_name, handler)

Description	Allow to attach an event handler into a particular event name. Alias for <code>{@link CancelableEvents#cancelableOn}</code> .
Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to attach to. ■ <code>handler {function}</code> —
Returns	<code>void</code>

VisualComponent.removeChildView(view_id, placeholder_selector, view_name)

Description	Removes a child view for a given view id.
--------------------	---

Parameters	<ul style="list-style-type: none"> ■ <code>view_id {string}</code> — The identifier of the view, of the current component, that will be extended with an extra child view. ■ <code>placeholder_selector {string}</code> — Identifier of the location where the new view will be located inside the specified View (<code>view_id</code>). ■ <code>view_name {string}</code> — Identifier of an specific view into the placeholder.
Returns	<code>void</code>

VisualComponent.removeToViewContextDefinition(`view_id`, `property_name`)

Description	Removes an extra property to the UI context of a view..
Parameters	<ul style="list-style-type: none"> ■ <code>view_id {string}</code> — The identifier of the view, of the current component, that will have its context extended.. ■ <code>property_name {string}</code> — Name of the new property to be added.
Returns	<code>void</code>

VisualComponent.removeToViewEventsDefinition(`view_id`, `event_selector`)

Description	Allows to remove and an extra event handler added previously..
Parameters	<ul style="list-style-type: none"> ■ <code>view_id {string}</code> — The identifier of the view, of the current component.. ■ <code>event_selector {string}</code> —
Returns	<code>void</code>

VisualComponent.setChildViewIndex(`view_id`, `placeholder_selector`, `view_name`, `index`)

Description	Change the position of a Child View inside a container.
Parameters	<ul style="list-style-type: none"> ■ <code>view_id {string}</code> — The identifier of the view, of the current component, that will have the Child View to change the index. ■ <code>placeholder_selector {string}</code> — Identifier of the location where the view is located inside the specified View (<code>view_id</code>). ■ <code>view_name {string}</code> — Identifier of an specific view into the placeholder. ■ <code>index {number}</code> — The new index to position the Child View.

Returns void

Members

application

Description The name that identifies this kind of component. This name is used both for registering a new component and getting a component implementation with [ComponentContainer](#).
 TODO: both members have the same description

Type [ComponentContainer](#)

componentName

Description The name which identify this kind of component. This name is used both for registering a new component and getting a component implementation with [ComponentContainer](#).

Type String

CustomContentTypeBaseView

Description Base CCT class from where all the custom CCT extend from inherits the basic render and destroy methods from Backbone.View

Extends [View](#)

Members

- [TODO: list](#)

Properties

- [bindings](#)
- [events](#)
- [template](#)

Methods

CustomContentTypeBaseView.destroy()

Description TODO

Inherited from [View.destroy\(\)](#)

Returns void

CustomContentTypeBaseView.getContext()

Description Returns the object which will be consumed by this views template.

Inherited from [View.getContext\(\)](#)

Returns Object

CustomContentTypeBaseView.getContextDataRequest()

Description TODO: what are valid values here ? how do I know which values I can use? Returns the list of contexts that this CCT is requesting the list of contexts that you may need to run the CCT..
For example, if this CCT is meant to be dropped in over an "item", for example in a PDP or in an item cell of the search page then this method should return ['item']

Returns Array.<String>

CustomContentTypeBaseView.initialize(...options)

Description this is executed when a new view is instantiated - equivalent to the instance constructor..

Inherited from [View.initialize\(...options\)](#)

Parameters ■ options {any} — TODO

Returns VOID

CustomContentTypeBaseView.install

Returns install

Description This method is called the first time a CCT is dragged from the admin panel to the application. It indicates when does the CCT instance has finished making all the Ajax call that are necessary for rendering. Also it initializes the CCT settings. If the CCT needs to do some async initialization, for example doing ajax first or other logic, this method should be overridden and and return a {@link Deferred}. Any error that cause that the CustomContentType object could not be installed, means that the CCT will be removed from the DOM and destroyed.

Parameters ■ settings {any} — the settings of the CCT. TODO: does this object have some minimal API ?.
■ context_data {any} — TODO

Returns {@link Deferred}

CustomContentTypeBaseView.render()

Description	TODO
Inherited from	View.render()
Returns	void

CustomContentTypeBaseView.update(settings)

Description	Called when the user edits the CCT settings in SMT panel. Each specific CCT will have to override this method in order to update the settings, validating the input and probably trigger a re-render of this view for visual feedback. Validation is done by returning a resolved or rejected {@link Deferred} . Return rejected means given settings are invalid for this instance.
Parameters	<ul style="list-style-type: none"> ■ <code>settings {any}</code> — the new settings of the CCT.
Returns	{@link Deferred}

CustomContentTypeBaseView.validateContextDataRequest(context_data)

Description	Validate if the context that the CCT receives is enough for the CCT to work returning true or false otherwise. if true it means everything is fine for this CCT, if false it will prevent the 'save' action to take effect
Parameters	<ul style="list-style-type: none"> ■ <code>context_data {any}</code> — An object with the contexts found.
Returns	Boolean

Members

bindings

Description	TODO
Inherited from	{@link View#bindings}
Type	Object

events

Description	TODO
--------------------	------

Inherited from	{@link View#events}
Type	Object

template

Description	The template for this view, in general a function that accepts this view context object and returns an HTML string. If it is a string an AMD module with that name is tried to be loaded. If it is undefined the view will be rendered without errors as an empty string.
Inherited from	{@link View#template}
Type	function

Utils

Description	A collection of miscellaneous utility methods that goes from object manipulation, text formatting, data validation, environment information, internationalization, URL manipulation, etc. You can require the module by the Utils, example:
-------------	---

```
define('MyCustomModule', ['Utils'], function (Utils)
{
    var translated = Utils.translate('There are $(0) apples in the tree', 55);
});
```

Also, all methods in Utils are mixed in underscore, so for example, you is the same to call `Utils.translate()` and `_.translate()`

Methods	<ul style="list-style-type: none"> ■ TODO: list
---------	--

Members

Utils.dateToString(date)

Description	Converts a date object to string using international format YYYY-MM-dd. Useful for inputs of type="date".
Parameters	<ul style="list-style-type: none"> ■ date {Date} —
Returns	String

Utils.deepCopy(obj, obj)

Description	Deep Copy of the object taking care of Backbone models.
Parameters	<ul style="list-style-type: none"> ■ obj {any} —TODO

■ `obj {any}` —TODO
Returns `TODO`

Utils.formatCurrency(value, symbol)

Description Formats given number and symbol to a currency like string. Example:
`Utils.formatCurrency(10, '£')`.
Parameters ■ `value {String}` —
 ■ `symbol {String}` —
Returns `String`

Utils.formatPhone(phone, format)

Description Will try to reformat a phone number for a given phone Format. If no format is
 given, it will try to use the one in site settings..
Parameters ■ `phone {String}` —TODO
 ■ `format {String}` —TODO
Returns `String`

Utils.formatQuantity(number)

Description Formats with commas as thousand separator (e.g. for displaying quantities).
Parameters ■ `number {String}` —TODO
Returns `String`

Utils.getAbsoluteUrl(path)

Description returns the absolute URL of given relative path.
Parameters ■ `path {String}` —
Returns `String`

Utils.getPathFromObject(object, path, default_value)

Description Gets given object property supporting deep property references by separating
 names with dots (`'.'`;
Parameters ■ `object {any}` — The object to get the property from.

- `path {String}` — a path with values separated by dots.
- `default_value {any}` — value to return if nothing is found..

Returns `TODO`

Utils.isCheckoutDomain()

Description Determines if we are in a secure checkout domain or in a secure single domain environment.
Returns true if in checkout domain.

Returns `Boolean`

Utils.isInCheckout()

Description Determines if we are in checkout or my account ssp.
Returns true if in checkout or in single domain

Returns `Boolean`

Utils.isInShopping()

Description Determines if we are in shopping ssp used when there are frontend features only shown in the shopping domain.
Returns true if in shopping domain, false if in checkout or myaccount.

Returns `Boolean`

Utils.isShoppingDomain()

Description Determines if we are in shopping domain (secure or non secure) or single domain.
Returns true if in checkout or in single domain function.

Returns `Boolean`

Utils.isSingleDomain()

Description Determines if we are in a single domain environment.
Returns true if single domain.

Returns `Boolean`

Utils.setPathFromObject(object, path, value)

Returns	setPathFromObject
Description	Sets given object property supporting deep property references by separating names with dots (.).
Parameters	<ul style="list-style-type: none"> object {any} — the object to modify. path {String} — a path with values separated by dots. value {any} — the value to set.
Returns	TODO

Utils.stringToDate(str_date, options)

Description	Parse a string date into a date object..
Parameters	<ul style="list-style-type: none"> str_date {String} — TODO options {Object} — format is the String format that specify the format of the input string. By Default YYYY-MM-dd. plusMonth: Number that indicate how many month offset should be applied whne creating the date object..
Returns	TODO

Utils.translate(text)

Description	Used on all of the hardcoded texts in the templates. Gets the translated value from SC.Translations object literal. Please always use the syntax <code>_('string').translate(1, 2)</code> instead of the syntax <code>_.translate('string', 1, 2)</code> if possible. Example using the Utils object:
-------------	---

```
Utils.translate('There are $(0) apples in the tree', appleCount)
```

Sample example using the `_.translate` (underscore mixing):

```
_('There are $(0) apples in the tree').translate(appleCount)
```

Parameters	<ul style="list-style-type: none"> text {String} — @return {String}..
------------	--

Returns	String
---------	--------

Utils.validatePhone(phone)

Description	TODO
-------------	------

Returns an error message if the passed phone is invalid or false if it is valid

Parameters ■ phone {String} — TODO

Returns String

Utils.validateSecurityCode(code)

Description TODO
Returns a non empty string with a internationalized warning message.

Parameters ■ value {String} — TODO

Returns String

Utils.validateState(value, valName, options)

Description TODO
Returns an error message if the passed state is invalid or false if it is valid.

Parameters ■ value {String} —
 ■ valName {String} —
 ■ options {Object} —

Returns String

Utils.validateZipCode(value, valName, options)

Description TODO
Returns an error message if the passed zip code is invalid or false if it is valid function.

Parameters ■ value {String} — TODO
 ■ valName {String} — TODO
 ■ options {Object} — TODO

Returns String

View

Description Subset of SuiteCommerce View exposed for Extensions developers for declaring new views, adding child-views, etc.

Internally is implemented with Backbone.View which is enhanced with handlebars templates, render plugins, context, SEO, composition, navigation, etc.

Methods	<ul style="list-style-type: none"> ■ <code>TODO: list</code>
Members	<ul style="list-style-type: none"> ■ <code>bindings</code> ■ <code>events</code> ■ <code>template</code>

Members

View.extend(view_definition)

Description	TODO
Parameters	<ul style="list-style-type: none"> ■ <code>view_definition {View}</code> — an object with some of this class members..
Returns	{@link Class}.<View>

View.destroy()

Description	TODO
Returns	void

View.getContext()

Description	Returns the object which will be consumed by this view's template.
Returns	Object

View.initialize(...options)

Description	This is executed when a new view is instantiated - equivalent to the instance constructor..
Parameters	<ul style="list-style-type: none"> ■ <code>options {any}</code> —
Returns	void

View.render()

Description	TODO
Returns	void

Members

bindings

Description	TODO
Type	Object

events

Description	TODO
Type	Object Object

template

Description	The template for this view, in general a function that accepts this view context object and returns an HTML string. If it is a string an AMD module with that name is tried to be loaded. If it is undefined the view will be rendered without errors as an empty string.
Type	function String

Deferred

Description	Deferred object. In SCA the implementation of deferred object is actually <code>jQuery.Deferred</code> . For simplifying, this implementation unifies both the <code>jQuery.Deferred</code> and the Promise Object into one. See the jQuery API documentation . Extension implementers can safely require jquery by the AMD name <code>jQuery</code> and instantiate Deferred like this:
-------------	--

```
define('MyExtension', ['jQuery'], function(jQuery) {
  var promise = jQuery.Deferred();
});
```

Functions	<ul style="list-style-type: none"> ■ <code>TODO: list</code>
-----------	---

Methods

always(alwaysCallbacks)

Description	Add handlers to be called when the Deferred object is either resolved or rejected. The <code>deferred.always()</code> method receives the arguments that were used to <code>.resolve()</code> or <code>.reject()</code> the Deferred object, which are often very different.
-------------	--

Parameters ■ `alwaysCallbacks {function}` — A function, or array of functions, that is called when the Deferred is resolved or rejected..

Returns {@link Deferred}

done(doneCallbacks)

Description Add handlers to be called when the Deferred object is resolved. The `deferred.done()` method accepts one or more arguments, all of which can be either a single function or an array of functions. When the Deferred is resolved, the `doneCallbacks` are called. Callbacks are executed in the order they were added.

Parameters ■ `doneCallbacks {function}` —TODO

Returns {@link Deferred}

fail(failCallbacks)

Description Add handlers to be called when the Deferred object is rejected. The `deferred.fail()` method accepts one or more arguments, all of which can be either a single function or an array of functions. When the Deferred is rejected, the `failCallbacks` are called..

Parameters ■ `failCallbacks {function}` —

Returns {@link Deferred}

notify(args)

Description Call the `progressCallbacks` on a Deferred object with the given `args`.

Parameters ■ `args {Object}` — Optional arguments that are passed to the `progressCallbacks`..

Returns {@link Deferred}

progress(progressCallbacks)

Description Add handlers to be called when the Deferred object generates progress notifications..

Parameters ■ `progressCallbacks {function}` — A function, or array of functions, to be called when the `Deferred.notify()` is called..

Returns {@link Deferred}

promise()

Description Return a Deferred Promise object..

Returns {@link Deferred}

reject(args)

Description Reject a Deferred object and call any failCallbacks with the given args..

Parameters

- `args {Object}` — Optional arguments that are passed to the failCallbacks..

Returns {@link Deferred}

resolve(args)

Description Resolve a Deferred object and call any doneCallbacks with the given args..

Parameters

- `args {object}` —
TODO

Returns {@link Deferred}

state()

Description Determine the current state of a Deferred object.
Returns 'pending' | 'resolved' | 'rejected'

Returns String

then(doneFilter, failFilter, progressFilter)

Description TODO

Parameters


- `doneFilter {function}` — A function that is called when the Deferred is resolved..
- `failFilter {function}` — An optional function that is called when the Deferred is rejected..
- `progressFilter {function}` — An optional function that is called when progress notifications are sent.

Returns {@link Deferred}

SuiteCommerce Back-end JavaScript APIs

The SuiteCommerce Back-end JavaScript APIs include the following classes:

- `BackendBaseComponent`
- `BackendCancelableEvents`
- `BackendCartComponent`
- `BackendComponentContainer`
- `SCErrors`
- `SCEventWrapper`
- `SCModel`
- `ServiceController`

 **Note:** This documentation is in progress. For the JsDoc version of this documentation, see [SuiteCommerce Back-end JavaScript APIs](#).

BackendBaseComponent

Description	Base abstract class for backend-end components based on SuiteScript. Use method <code>BackendBaseComponent.extend()</code> to build concrete component implementations. Note for implementers: For each method defined in a component, an equivalent synchronous operation method will be created automatically. For example if your implementation has a method <code>foo()</code> that returns a <code>Deferred</code> , and your component extends <code>BackendBaseComponent</code> , the extra method <code>fooSync()</code> will be created automatically that will return the value on which the <code>Deferred</code> was resolved, instead of the actual <code>Deferred</code> .
Functions	<ul style="list-style-type: none"> ■ <code>extend</code> ■ <code>on</code> ■ <code>off</code> ■ <code>cancelableOn</code> ■ <code>cancelableOff</code> ■ <code>cancelableDisable</code> ■ <code>cancelableEnable</code> ■ <code>cancelableTrigger</code> ■ <code>cancelableTriggerUnsafe</code>
Properties	<ul style="list-style-type: none"> ■ <code>componentName</code>

Methods

extend

Function name	<code>extend</code>
Function description	<p>Extends the current component to generate a child one</p>

Function Parameters	<ul style="list-style-type: none"> ■ <code>componentDefinition {Object}</code> — Any object with properties/methods that will be used to generate the Component that will be returned.
----------------------------	---

on

Function name	on
Function description	Allow to attach an event handler into a particular event name. Alias for {@link BackendCancelableEvents#cancelableOn}.
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {string}</code> — The name of the event to attach to. ■ <code>handler {function}</code> — The event handler function that will be invoked when the event <code>event_name</code> is triggered. This function can receive optionally one parameter representing the action parameter. Besides optionally can return a Deferred to details the execution of the trigger's callback. If the returned Deferred is rejected the trigger's callback wont be called.

off

Function name	off
Function description	Allow to detach an event handler. Alias for {@link BackendCancelableEvents#cancelableOff}.
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {string}</code> — The name of the event to detach from. Note that this parameter is mandatory. ■ <code>handler {function}</code> — The event handler function that will be removed from the list of register handlers. Note this parameter is required..

cancelableOn

Function name	cancelableOn
Function description	Allow to attach an event handler into a particular event name.
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to attach to. ■ <code>handler {function}</code> — The event handler function that will be invoked when the event <code>event_name</code> is triggered. This function can receive optionally one parameter representing the action parameter. Besides optionally can return a Deferred to details the execution of the trigger's callback. If the returned Deferred is rejected the trigger's callback wont be called.

cancelableOff

Function name	cancelableOff
----------------------	---------------

Function description	Allow to detach an event handler..
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to detach from. Note that this parameter is mandatory. ■ <code>handler {function}</code> — The event handler function that will be removed from the list of register handlers. Note this parameter is required..

cancelableDisable

Function name	cancelableDisable
Function description	Allow to disable all the handlers of a particular event.
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to disable.

cancelableEnable

Function name	cancelableEnable
Function description	Allow to enable (restore) all the handlers of a particular event.
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to disable.

cancelableTrigger

Function name	cancelableTrigger
Function description	Trigger the indicate event with the passed in parameters. In case that any of the event handler reject the execution (the callback WONT be called).
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — Event to trigger. ■ <code>args {params}</code> — All other parameter passed to this function will be broadcaster to all event handlers.

cancelableTriggerUnsafe

Function name	cancelableTriggerUnsafe
Function description	Trigger the indicate event with the passed in parameters WITHOUT sanitize them. In case that any of the event handler reject (returns a rejected Deferred) the execution (the callback WONT be called).
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — Event to trigger. ■ <code>args {params}</code> — All other parameter passed to this function will be broadcaster to all event handlers.

Members

componentName

Property description	The name which identify this kind of component. This name is used both for registering a new component and getting a component implementation with <code>{@link ComponentContainer}</code> .
Property Type	string

BackendCancelableEvents

Description	xx _ need a description _ xx
Functions	<ul style="list-style-type: none"> cancelableOn cancelableOff cancelableDisable cancelableEnable cancelableTrigger cancelableTriggerUnsafe
Properties	None.
Events	None.

Methods

cancelableOn

Function name	cancelableOn
Function description	Allow to attach an event handler into a particular event name.
Function Parameters	<ul style="list-style-type: none"> <code>event_name {String}</code> — The name of the event to attach to. <code>handler {function}</code> — The event handler function that will be invoked when the event <code>event_name</code> is triggered. This function can receive optionally one parameter representing the action parameter. Besides optionally can return a Deferred to details the execution of the trigger's callback. If the returned Deferred is rejected the trigger's callback wont be called.

cancelableOff

Function name	cancelableOff
---------------	---------------

Function description	Allow to detach an event handler..
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to detach from. Note that this parameter is mandatory. ■ <code>handler {function}</code> — The event handler function that will be removed from the list of register handlers. Note this parameter is required..

cancelableDisable

Function name	cancelableDisable
Function description	Allow to disable all the handlers of a particular event.
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to disable.

cancelableEnable

Function name	cancelableEnable
Function description	Allow to enable (restore) all the handlers of a particular event.
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to disable.


cancelableTrigger

Function name	cancelableTrigger
Function description	Trigger the indicate event with the passed in parameters. In case that any of the event handler reject the execution (the callback WONT be called).
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — Event to trigger. ■ <code>args {params}</code> — All other parameter passed to this function will be broadcaster to all event handlers.


cancelableTriggerUnsafe

Function name	cancelableTriggerUnsafe
Function description	Trigger the indicate event with the passed in parameters WITHOUT sanitize them. In case that any of the event handler reject (returns a rejected Deferred) the execution (the callback WONT be called).
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — Event to trigger. ■ <code>args {params}</code> — All other parameter passed to this function will be broadcaster to all event handlers.

Class Properties

 **Note:** This class does not include any class properties.

Class Events

 **Note:** This class does not include any class events.

BackendCartComponent

Description xx _ need a description _ xx

Functions

- [addLine](#)
- [addLineSync](#)
- [addLines](#)
- [addLinesSync](#)
- [getLines](#)
- [getLinesSync](#)
- [removeLine](#)
- [removeLineSync](#)
- [updateLine](#)
- [updateLineSync](#)
- [getSummary](#)
- [getSummarySync](#)
- [submit](#)
- [submitSync](#)
- [addPayment](#)
- [addPaymentSync](#)
- [getPaymentMethods](#)
- [getPaymentMethodsSync](#)
- [addPromotion](#)
- [addPromotionSync](#)
- [removePromotion](#)
- [removePromotionSync](#)
- [getPromotions](#)
- [getPromotionsSync](#)
- [estimateShipping](#)
- [estimateShippingSync](#)
- [removeShipping](#)
- [removeShippingSync](#)
- [clearEstimateShipping](#)
- [clearEstimateShippingSync](#)
- [getAddresses](#)

- getAddressessSync
- getShipAddress
- getShipAddressSync
- getBillAddress
- getBillAddressSync
- getShipMethods
- getShipMethodsSync
- getShipMethod
- getShipMethodSync
- voidLine
- voidLineSync
- unvoidLine
- unvoidLineSync
- updateCustomer
- updateCustomerSync
- extend
- on
- off
- cancelableOn
- cancelableOff
- cancelableDisable
- cancelableEnable
- cancelableTrigger
- cancelableTriggerUnsafe

Properties

Events

- componentName
- beforeUpdateLine
- afterUpdateLine
- beforeRemoveLine
- afterRemoveLine
- beforeEstimateShipping
- afterEstimateShipping
- beforeClearEstimateShipping
- afterClearEstimateShipping
- beforeAddPromotion
- afterAddPromotion
- beforeRemovePromotion
- afterRemovePromotion
- beforeAddPayment
- afterAddPayment
- beforeSubmit
- afterSubmit

- [beforeAddLine](#)
- [afterAddLine](#)

Class Functions

addLine

Function name	addLine
Function description	Adds a new line into the cart. Returns a Deferred that is resolved into the added line id {@link String}, in the case the operation was done successfully, or rejected with an error message..
Function Parameters	<ul style="list-style-type: none"> ■ data {Object} —

addLineSync

Function name	addLineSync
Function description	Synchronous version of {@link BackendCartComponent#addLine}..
Function Parameters	<ul style="list-style-type: none"> ■ data {Object} —

addLines

Function name	addLines
Function description	Adds new lines into the cart. Returns a Deferred that is resolved with the an array of line ids (Array of strings) into the added line id in the case the operation or rejected with an error message. was done successfully..
Function Parameters	<ul style="list-style-type: none"> ■ lines {Object} —

addLinesSync

Function name	addLinesSync
Function description	Synchronous version of {@link BackendCartComponent#addLines}..
Function Parameters	<ul style="list-style-type: none"> ■ lines {Object} —

getLines

Function name	getLines
---------------	----------

Function description returns the lines of the cart as a Deferred that is resolved in the case the operation was done successfully or the promise is rejected with an error message. The promise resolves with an array of {@link Line}.

Function Parameters

getLinesSync

Function name getLinesSync

Function description Synchronous version of {@link BackendCartComponent#getLines}..

Function Parameters

removeLine

Function name removeLine

Function description Removes a line from the cart. Returns a Deferred that is resolved in the case the operation was done successfully or rejected with an error message..

Function Parameters

- `internalid {String}` — id of the line to remove.

removeLineSync

Function name removeLineSync

Function description Synchronous version of {@link BackendCartComponent#removeLine}..

Function Parameters

updateLine

Function name updateLine

Function description Updates a cart's line. Returns a Deferred that is resolved with {@link Line} in the case the operation was done successfully, or rejected with an error message..

Function Parameters

- `line {Line}` —

updateLineSync

Function name updateLineSync

Function description Synchronous version of {@link BackendCartComponent#updateLine}..

Function Parameters

- `line {Object}` —

getSummary

Function name	getSummary
Function description	Returns the summary of the cart as a Deferred that is resolved with a {@link Summary} in the case the operation was done successfully or rejected with an error message..
Function Parameters	

getSummarySync

Function name	getSummarySync
Function description	Synchronous version of {@link BackendCartComponent#getSummary}..
Function Parameters	

submit

Function name	submit
Function description	Submits the order. Returns a Deferred that is resolved with a {@link ConfirmationSubmit} object in the case the operation was done successfully, or rejected with an error message..
Function Parameters	

submitSync

Function name	submitSync
Function description	Synchronous version of {@link BackendCartComponent#submitSync}..
Function Parameters	

addPayment

Function name	addPayment
Function description	Adds a payment method //TODO: returns? - Deferred resolved with ? // TODO: data parameter ? what's the structure?.
Function Parameters	

addPaymentSync

Function name	addPaymentSync
---------------	----------------

Function description Synchronous version of {@link BackendCartComponent#addPayment}.
TODO.

Function Parameters

getPaymentMethods

Function name getPaymentMethods

Function description returns the payment methods added to the order.

Function Parameters

getPaymentMethodsSync

Function name getPaymentMethodsSync

Function description Synchronous version of {@link BackendCartComponent#addPayment}.
TODO.

Function Parameters

addPromotion

Function name addPromotion

Function description Adds a promotion.

Function Parameters ■ the {Object} — promocode to add.

addPromotionSync

Function name addPromotionSync

Function description Synchronous version of {@link BackendCartComponent#addPromotion}.
TODO.

Function Parameters

removePromotion

Function name removePromotion

Function description Removes a promotion.

Function Parameters ■ data {Object} —

removePromotionSync

Function name	removePromotionSync
Function description	Synchronous version of {@link BackendCartComponent#removePromotion}. TODO.
Function Parameters	

getPromotions

Function name	getPromotions
Function description	Returns the promotions' codes added to the cart.
Function Parameters	

getPromotionsSync

Function name	getPromotionsSync
Function description	Synchronous version of {@link BackendCartComponent#getPromotions}. TODO.
Function Parameters	

estimateShipping

Function name	estimateShipping
Function description	Returns the estimated shipping costs.
Function Parameters	<ul style="list-style-type: none"> locationData {Object} — TODO: returns ?.

estimateShippingSync

Function name	estimateShippingSync
Function description	Synchronous version of {@link BackendCartComponent#estimateShipping}. TODO.
Function Parameters	

removeShipping

Function name	removeShipping
---------------	----------------

Function description Removes the shipping method // TODO: returns ? TODO @params?.

Function Parameters

removeShippingSync

Function name removeShippingSync

Function description Synchronous version of {@link BackendCartComponent#removeShipping}. TODO.

Function Parameters

clearEstimateShipping

Function name clearEstimateShipping

Function description Clears the shopping estimations in the cart TODO: returns ? TODO: params?.

Function Parameters

clearEstimateShippingSync

Function name clearEstimateShippingSync

Function description Synchronous version of {@link BackendCartComponent#clearEstimateShipping}. TODO.

Function Parameters

getAddresses

Function name getAddresses

Function description Returns the addresses of the order.

Function Parameters

getAddressesSync

Function name getAddressesSync

Function description Synchronous version of {@link BackendCartComponent#getAddresses}. TODO.

Function Parameters

getShipAddress

Function name	getShipAddress
Function description	Returns the shipping address of the order.
Function Parameters	

getShipAddressSync

Function name	getShipAddressSync
Function description	Synchronous version of {@link BackendCartComponent#getShipAddress}. TODO.
Function Parameters	

getBillAddress

Function name	getBillAddress
Function description	Returns the billing address of the order.
Function Parameters	

getBillAddressSync

Function name	getBillAddressSync
Function description	Synchronous version of {@link BackendCartComponent#getBillAddress}. TODO.
Function Parameters	

getShipMethods

Function name	getShipMethods
Function description	Returns the ship methods of the order.
Function Parameters	

getShipMethodsSync

Function name	getShipMethodsSync
Function description	Synchronous version of {@link BackendCartComponent#getShipMethods}. TODO.
Function Parameters	

getShipMethod

Function name	getShipMethod
Function description	Returns the selected shipping method of the order.
Function Parameters	

getShipMethodSync

Function name	getShipMethodSync
Function description	Synchronous version of {@link BackendCartComponent#getShipMethod}. TODO.
Function Parameters	

voidLine

Function name	voidLine
Function description	Voids a line. Implemented only for SCIS //TODO: params ? // TODO return ?.
Function Parameters	

voidLineSync

Function name	voidLineSync
Function description	Synchronous version of {@link BackendCartComponent#voidLine}. TODO.
Function Parameters	

unvoidLine

Function name	unvoidLine
Function description	Unvoids a line. Implemented only for SCIS. //TODO: params ? // TODO return ?.
Function Parameters	

unvoidLineSync

Function name	unvoidLineSync
Function description	Synchronous version of {@link BackendCartComponent#unvoidLine}. TODO.

Function Parameters

updateCustomer

Function name	updateCustomer
Function description	Updates a customer data. Implemented only for SCIS //TODO: params ? // TODO return ?.

Function Parameters

updateCustomerSync

Function name	updateCustomerSync
Function description	Synchronous version of {@link BackendCartComponent#updateCustomer}. TODO.

Function Parameters

extend

Function name	extend
Function description	Extends the current component to generate a child one.
Function Parameters	<ul style="list-style-type: none"> ■ <code>componentDefinition {Object}</code> — Any object with properties/methods that will be used to generate the Component that will be returned.

on

Function name	on
Function description	Allow to attach an event handler into a particular event name. Alias for {@link BackendCancelableEvents#cancelableOn}.
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {string}</code> — The name of the event to attach to. ■ <code>handler {function}</code> — The event handler function that will be invoked when the event <code>event_name</code> is triggered. This function can receive optionally one parameter representing the action parameter. Besides optionally can return a Deferred to details the execution of the trigger's callback. If the returned Deferred is rejected the trigger's callback wont be called.

off

Function name	off
---------------	-----

Function description	Allow to detach an event handler. Alias for {@link BackendCancelableEvents#cancelableOff}.
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {string}</code> — The name of the event to detach from. Note that this parameter is mandatory. ■ <code>handler {function}</code> — The event handler function that will be removed from the list of register handlers. Note this parameter is required..

cancelableOn

Function name	cancelableOn
Function description	Allow to attach an event handler into a particular event name.
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to attach to. ■ <code>handler {function}</code> — The event handler function that will be invoked when the event event_name is triggered. This function can receive optionally one parameter representing the action parameter. Besides optionally can return a Deferred to details the execution of the trigger's callback. If the returned Deferred is rejected the trigger's callback wont be called.

cancelableOff

Function name	cancelableOff
Function description	Allow to detach an event handler..
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to detach from. Note that this parameter is mandatory. ■ <code>handler {function}</code> — The event handler function that will be removed from the list of register handlers. Note this parameter is required..

cancelableDisable

Function name	cancelableDisable
Function description	Allow to disable all the handlers of a particular event.
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to disable.

cancelableEnable

Function name	cancelableEnable
----------------------	------------------

Function description	Allow to enable (restore) all the handlers of a particular event.
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — The name of the event to disable.

cancelableTrigger

Function name	cancelableTrigger
Function description	Trigger the indicate event with the passed in parameters. In case that any of the event handler reject the execution (the callback WONT be called).
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — Event to trigger. ■ <code>args {params}</code> — All other parameter passed to this function will be broadcaster to all event handlers.

cancelableTriggerUnsafe

Function name	cancelableTriggerUnsafe
Function description	Trigger the indicate event with the passed in parameters WITHOUT sanitize them. In case that any of the event handler reject (returns a rejected Deferred) the execution (the callback WONT be called).
Function Parameters	<ul style="list-style-type: none"> ■ <code>event_name {String}</code> — Event to trigger. ■ <code>args {params}</code> — All other parameter passed to this function will be broadcaster to all event handlers.

Class Properties

componentName

Property description	The name which identify this kind of component. This name is used both for registering a new component and getting a component implementation with {@link ComponentContainer}.
Property Type	string

Class Events

beforeUpdateLine

Event description	Cancelable event triggered before a cart's line is updated. See {@link BackendCancelableEvents} TODO: type.
Event Parameters	<ul style="list-style-type: none"> ■ <code>TODO {boolean}</code> — TODO.

afterUpdateLine

Event description Triggered after a cart's line is updated See TODO: type.

Event Parameters ■ TODO {boolean} — TODO.

beforeRemoveLine

Event description Cancelable event triggered before a cart's line is removed. See {@link BackendCancelableEvents} TODO: type.

Event Parameters ■ TODO {boolean} — TODO.

afterRemoveLine

Event description Triggered after a cart's line is removed TODO: type.

Event Parameters ■ TODO {boolean} — TODO.

beforeEstimateShipping

Event description Cancelable event triggered before doing an estimate shipping in the cart. See {@link BackendCancelableEvents} TODO: type.

Event Parameters ■ TODO {boolean} — TODO.

afterEstimateShipping

Event description Triggered after an estimate shipping is done in the cart TODO: type.

Event Parameters ■ TODO {boolean} — TODO.

beforeClearEstimateShipping

Event description Cancelable event triggered before clearing an estimate shipping in the cart. See {@link BackendCancelableEvents} TODO: type.

Event Parameters ■ TODO {boolean} — TODO.

afterClearEstimateShipping

Event description Triggered after an estimate shipping is cleared in the cart TODO: type.

Event Parameters ■ TODO {boolean} — TODO.

beforeAddPromotion

Event description	Triggered before a promotion is added to the cart. See {@link BackendCancelableEvents} TODO: type.
Event Parameters	■ TODO {boolean} — TODO.

afterAddPromotion

Event description	Triggered before a promotion is added to the cart TODO: type.
Event Parameters	■ TODO {boolean} — TODO.

beforeRemovePromotion

Event description	Triggered before a promocode is removed from the cart. See {@link BackendCancelableEvents} TODO: type.
Event Parameters	■ TODO {boolean} — TODO.

afterRemovePromotion

Event description	Triggered after a promocode is removed from the cart TODO: type.
Event Parameters	■ TODO {boolean} — TODO.

beforeAddPayment

Event description	Triggered before a payment method is added to the order. See {@link BackendCancelableEvents} TODO: type.
Event Parameters	■ TODO {boolean} — TODO.

afterAddPayment

Event description	Triggered after a payment method is added to the order TODO: type.
Event Parameters	■ TODO {boolean} — TODO.

beforeSubmit

Event description	Triggered before the order is submitted. See {@link BackendCancelableEvents} TODO: type.
Event Parameters	■ TODO {boolean} — TODO.

afterSubmit

Event description	Triggered after the order is submitted TODO: type.
Event Parameters	<ul style="list-style-type: none"> ■ <code>TODO {boolean}</code> — TODO.

beforeAddLine

Event description	Cancelable event triggered before adding a new cart's line. See {@link BackendCancelableEvents} TODO: type.
Event Parameters	<ul style="list-style-type: none"> ■ <code>TODO {boolean}</code> — TODO.

afterAddLine

Event description	Triggered after a new line is added in the cart TODO: type.
Event Parameters	<ul style="list-style-type: none"> ■ <code>TODO {boolean}</code> — TODO.

BackendComponentContainer

Description	xx _ need a description _ xx
Functions	<ul style="list-style-type: none"> ■ registerComponent ■ getComponent
Properties	None.
Events	None.

Class Functions

registerComponent


Function name	registerComponent
Function description	Allows to register a new component into the running application it also seals the component, so as to not add new properties or messing up with the available components APIs..
Function Parameters	<ul style="list-style-type: none"> ■ <code>component {BackendBaseComponent}</code> — Component to be registered.

getComponent


Function name	getComponent
---------------	--------------

Function description	Returns the requested component based on its name if there is no component with that name registered in this container.
Function Parameters	<ul style="list-style-type: none"> ■ <code>component_name {String}</code> —

Class Properties

 **Note:** This class does not include any class properties.

Class Events

 **Note:** This class does not include any class events.

SCError

Description	xx _ need a description _ xx
Functions	<ul style="list-style-type: none"> ■ <code>get</code> ■ <code>put</code> ■ <code>post</code> ■ <code>delete</code> ■ <code>patch</code>
Properties	<ul style="list-style-type: none"> ■ <code>status</code> ■ <code>code</code> ■ <code>message</code>
Events	None.

Class Functions

get

Function name	<code>get</code>
Function description	xx _ need a description _ xx
Function Parameters	

put

Function name	<code>put</code>
Function description	xx _ need a description _ xx

Function Parameters

post

Function name	post
Function description	xx _ need a description _ xx
Function Parameters	

delete

Function name	delete
Function description	xx _ need a description _ xx
Function Parameters	

patch

Function name	patch
Function description	xx _ need a description _ xx
Function Parameters	

Class Properties

status

Property description	http status code corresponding to this error.
Property Type	number


code

Property description	a string-code for this error - like <code>ERR_BAD_REQUEST</code>.
Property Type	String

message

Property description	xx _ need a description _ xx
Property Type	string

Class Events

 **Note:** This class does not include any class events.

SCEventWrapper

Description	xx _ need a description _ xx
Functions	<ul style="list-style-type: none"> extend
Properties	<ul style="list-style-type: none"> name
Events	None.

Class Functions

extend


Function name	extend
Function description	Create a new class with provided name and methods that will automatically trigger <code>after</code> and <code>before</code> events when methods are called to provide AOP-similar functionality..
Function Parameters	<ul style="list-style-type: none"> <code>definition {Object}</code> — The definition for the new model that contain a name and the model methods to wrap. The name is mandatory..

Class Properties

name

Property description	The name of this model. Is mandatory that al SCEventWrappers to have a name..
Property Type	string

Class Events

 **Note:** This class does not include any class events.

SCModel

Description	xx _ need a description _ xx
Functions	<ul style="list-style-type: none"> validate
Properties	<ul style="list-style-type: none"> validation

	<ul style="list-style-type: none"> name
Events	None.

Class Functions

validate

Function name	validate
Function description	Validate provided data using {@link SCModel#validation} rules like documented in Backbone.Validation .
Function Parameters	<ul style="list-style-type: none"> data {object} —

Class Properties


validation

Property description	a Backbone.Validation like object to validate given json data provided in {@link SCModel#validate}.
Property Type	object

name

Property description	The name of this model. Is mandatory that all SCEventWrappers to have a name..
Property Type	string

Class Events

 **Note:** This class does not include any class events.

ServiceController

Description	xx _ need a description _ xx
Functions	<ul style="list-style-type: none"> sendContent sendError get

	<ul style="list-style-type: none"> post put delete patch
Properties	<ul style="list-style-type: none"> request response method data options name
Events	None.

Class Functions

sendContent

Function name	sendContent
Function description	Writes the given content in the request object using the right headers, and content type.
Function Parameters	<ul style="list-style-type: none"> options {ServiceControllerOptions} — content {String} —

sendError

Function name	sendError
Function description	Process given error and then writes it in the http response.
Function Parameters	<ul style="list-style-type: none"> e {nlobjError} —

get

Function name	get
Function description	Users should implement this method for the service to support the http get operation.
Function Parameters	

post

Function name	post
---------------	------

Function description	Users should implement this method for the service to support the http post operation and use {@link ServiceController#sendContent} to write the response. Requested data is available in {@link ServiceController#data}.
Function Parameters	

put

Function name	put
Function description	Users should implement this method for the service to support the http put operation.
Function Parameters	

delete

Function name	delete
Function description	Users should implement this method for the service to support the http delete operation.
Function Parameters	

patch

Function name	patch
Function description	Users should implement this method for the service to support the http patch operation.
Function Parameters	

Class Properties

request

Property description	The request made to this service.
Property Type	nlobjRequest

response

Property description	the response object to write the data to respond by the call.
-----------------------------	---

Property Type	nlobjResponse
---------------	---------------

method

Property description	the REST http method currently requested.
----------------------	---

Property Type	String
---------------	--------

data

Property description	the JSON object in the request body.
----------------------	--------------------------------------

Property Type	any
---------------	-----

options

Property description	The options which configure this service controller.
----------------------	--


Property Type	ServiceControllerOptions
---------------	--------------------------

name

Property description	The name of this model. Is mandatory that all SCEventWrappers to have a name..
----------------------	--

Property Type	string
---------------	--------

Class Events

 **Note:** This class does not include any class events.

Deferred

Description	xx _ need a description _ xx
-------------	------------------------------

Functions	<ul style="list-style-type: none"> ■ <code>reject</code> ■ <code>promise</code> ■ <code>notify</code> ■ <code>fail</code> ■ <code>state</code> ■ <code>done</code>
-----------	--

- `progress`
- `always`
- `then`
- `resolve`

Properties None.

Events None.

Class Functions

reject

Function name `reject`

Function description Reject a Deferred object and call any failCallbacks with the given args..

Function Parameters ■ `args {Object}` — Optional arguments that are passed to the failCallbacks..

promise

Function name `promise`

Function description Return a Deferred Promise object..

Function Parameters

notify

Function name `notify`

Function description Call the progressCallbacks on a Deferred object with the given args..

Function Parameters ■ `args {Object}` — Optional arguments that are passed to the progressCallbacks..

fail

Function name `fail`

Function description Add handlers to be called when the Deferred object is rejected. The `deferred.fail()` method accepts one or more arguments, all of which can be either a single function or an array of functions. When the Deferred is rejected, the failCallbacks are called..

Function Parameters ■ `failCallbacks {function}` —

state

Function name	state
Function description	Determine the current state of a Deferred object..
Function Parameters	

done

Function name	done
Function description	Add handlers to be called when the Deferred object is resolved. The deferred.done() method accepts one or more arguments, all of which can be either a single function or an array of functions. When the Deferred is resolved, the doneCallbacks are called. Callbacks are executed in the order they were added..
Function Parameters	<ul style="list-style-type: none"> doneCallbacks {function} —

progress

Function name	progress
Function description	Add handlers to be called when the Deferred object generates progress notifications..
Function Parameters	<ul style="list-style-type: none"> progressCallbacks {function} — A function, or array of functions, to be called when the Deferred.notify() is called..

always

Function name	always
Function description	Add handlers to be called when the Deferred object is either resolved or rejected. The deferred.always() method receives the arguments that were used to .resolve() or .reject() the Deferred object, which are often very different..
Function Parameters	<ul style="list-style-type: none"> alwaysCallbacks {function} — A function, or array of functions, that is called when the Deferred is resolved or rejected..

then


Function name	then
Function description	xx _ need a description _ xx
Function Parameters	<ul style="list-style-type: none"> doneFilter {function} — A function that is called when the Deferred is resolved..

- `failFilter {function}` — An optional function that is called when the Deferred is rejected..
- `progressFilter {function}` — An optional function that is called when progress notifications are sent..


resolve

Function name	resolve
Function description	Resolve a Deferred object and call any doneCallbacks with the given args..
Function Parameters	<ul style="list-style-type: none"> ■ <code>args {object}</code> —

Class Properties

 **Note:** This class does not include any class properties.

Class Events

 **Note:** This class does not include any class events.

Class

Description	xx _ need a description _ xx
Functions	<ul style="list-style-type: none"> ■ <i>m</i>
Properties	None.
Events	None.

Published Documentation Reference

The following topics are published in the NetSuite Help Center, Developer's Portal, and SuiteAnswers. These topics contain important information that might be useful during theme and extension development. These include:

- [Site Management Tools](#) – This section of the Help Center introduces you to Site Management Tools.
- [SuiteCommerce Advanced Architecture](#) – This section of the Help Center describes the Model View Controller (MVC) and Backbone architecture that is the framework behind any SuiteCommerce Standard and SuiteCommerce Advanced site.
- [Design Architecture](#) – This section describes SCS/SCA design hierarchy, Sass definitions, and style guide creation instructions.
- **Custom Content Types (CCT)** – These topics explain everything you need to build your own CCTs.
 - [Create a CCT Module](#) – This topic explains how to build a CCT module within your extension's code.
 - [Custom Content Type](#) – This topic explains how to implement your CCT using Site Management Tools.
- **Configuration** – These topics include instructions on how to configure properties for a domain and how to build your own JSON configuration files or modify existing ones.
 - [Configuration](#) – Introduces you to everything you need to know about configuring a domain, including how to use the SuiteCommerce Configuration record in NetSuite.
 - [JSON Configuration Files](#) – This topic provides a brief overview of the configurationManifest.json file, individual module JSON files, and how they impact the SuiteCommerce Configuration record.
 - [Create a Custom JSON Configuration File](#) – Read this topic to familiarize yourself with the JSON configuration schema when building new JSON for an extension.
 - [Modify JSON Configuration Files](#) – Follow these instructions to introduce changes to pre-existing JSON configuration files.
 - [Configurable Properties Reference](#) – This topic explains each configurable property.