



**November 2013**

# Customer Portal Migration Guide Framework Version 2 to Version 3.1

**Revision 1**

The [online version of this Customer Portal Migration Guide](#) provides a series of guided questions to lead you through only the sections that apply to your current customer portal installation.

Copyright © 2000, 2012, Oracle Corporation and/or its affiliates. All rights reserved. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

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, 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 failsafe, 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.

This software or hardware and documentation may provide access to or information on 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. 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.

For legal notices, refer to <http://www.oracle.com/us/legal/index.html>.

---

---

# Contents

Migration overview . . . . .	6
Features and benefits . . . . .	8
Changes to the Customer Portal . . . . .	10
File structure . . . . .	10
The customer folder. . . . .	11
Assets in the /euf/ folder . . . . .	12
Widget changes . . . . .	13
Widget file structure. . . . .	13
Ability to extend standard widgets . . . . .	13
Removal of widget suffixes . . . . .	14
Migration of custom widgets . . . . .	15
PHP namespacing . . . . .	16
Syndicated widgets folder . . . . .	16
Dreamweaver support and extension . . . . .	17
Enabling Framework Version 3.1. . . . .	17
Migration resources . . . . .	20
Reverting to Framework Version 2 . . . . .	21
Code editing overview. . . . .	22
Modifying display elements. . . . .	23
Using HTML5 on custom pages. . . . .	24
Changing the width of the Email Address field . . . . .	25
Editing the width of the OpenLogin dialog. . . . .	26
YUI resources . . . . .	28
Using Code Assistant to migrate YUI code . . . . .	29
Adding the PasswordInput widget. . . . .	31
Setting password length. . . . .	33
Evaluating other functionality changes . . . . .	33
Using high-contrast mode for accessibility. . . . .	34
Edit site CSS. . . . .	34
Edit widget CSS . . . . .	40
Disabling incident receipt emails . . . . .	44

Changing incident source for rules . . . . .	45
Removing customer search preferences . . . . .	45
Changing the SmartAssistantDialog widget . . . . .	45
Using the organization password . . . . .	46
Editing the CommunitySearchResults widget . . . . .	47
Adding the Japanese name suffix . . . . .	47
Login required configuration setting . . . . .	47
Using the KnowledgeSyndication widget overlay feature . . . . .	48
Redirecting WAP pages to the basic page set . . . . .	52
Deprecation of Intent Guide . . . . .	54
Deprecation of SearchTruncation widget . . . . .	55
Adding label attributes . . . . .	55
Customer portal changes on the agent desktop . . . . .	57
Guided assistance styling on the agent desktop . . . . .	58
Replacing variables on the preview page . . . . .	59
Converting custom widgets . . . . .	60
Planning your widget conversion . . . . .	60
The info.yml file . . . . .	61
Making AJAX requests from widgets . . . . .	61
Using blocks in views . . . . .	62
Using JavaScript templates . . . . .	64
Overview of the conversion process . . . . .	65
Completing the widget information file . . . . .	69
Converting the controller . . . . .	71
Specifying AJAX handlers . . . . .	73
Converting the view . . . . .	73
Converting the logic file . . . . .	74
Updating the logic file for AJAX requests . . . . .	75
Accessing and rendering EJS views . . . . .	77
JavaScript recipes . . . . .	77
Using this.Y.one to get node objects . . . . .	77
Accessing properties on node objects . . . . .	78
Iterating through sibling elements . . . . .	78
Chaining function calls . . . . .	79
Listening to events . . . . .	79

---

Updating references to assets . . . . .	80
Converting custom code . . . . .	80
Using Connect PHP API in custom scripts . . . . .	80
Direct SQL queries . . . . .	80
Using PHP5 style constructors . . . . .	81
Namespacing . . . . .	81
Custom models . . . . .	81
PHP namespace requirements . . . . .	82
Detecting abuse in custom code . . . . .	83
Optional code cleanup . . . . .	83
Replacing business objects . . . . .	84
Replacing unused widgets . . . . .	87
ChatLaunchFormOpen . . . . .	88
MobileEmailAnswerLink . . . . .	88
ContactNameInput . . . . .	88
Staging and promoting version 3.1 . . . . .	90



# Customer Portal Migration Guide

## Framework Version 2 to Version 3.1

Now that your upgrade to the November 2013 release of Oracle RightNow CX is complete, it's time to migrate your customer portal to Framework Version 3.1 so you can start taking advantage of all its features and benefits.

**Important** It's important to be clear about the difference between **migration** and **upgrading**. Migration is moving from one version of the Customer Portal framework to a newer one. Upgrading is moving from one release of Oracle RightNow CX to a newer release. Before the November 2012 release, upgrading your Oracle RightNow CX application meant you also had to migrate to the new Customer Portal at the same time, absorbing all new features whether you needed them or not. With the introduction of independent versioning in Framework Version 3, you're in control of the migration process—you migrate the features you want when it's convenient for you.

Also, when we're talking about features specific to Framework Versions 3.0 and 3.1, we'll identify them as such. When features are common to both versions, we'll just say Framework Version 3.

You don't have to rush the migration process. Your customer portal will continue to work on its current framework until you're ready to migrate. We used to call the older framework the November 09 framework, but now we call it Framework Version 2. So you'll be migrating from Framework Version 2 to Framework Version 3.1.

Even though you don't have to rush, you'll want to get your customer portal on Framework Version 3.1 as soon as possible when you learn about the following benefits.

- **Independent frameworks and versioning**—You're never forced to migrate your customer portal, even when your organization upgrades to a new Oracle RightNow CX release. And you don't have to migrate your entire customer portal when you want just a single new feature. With Framework Version 3, you control *when* you migrate and *what* you migrate.
- **Automatic updates of backward compatible changes**—You don't have to do anything to adopt security patches and changes that are completely backward compatible. And when new features are available, the differences are spelled out clearly on the framework and widget version pages to help you decide if you want to migrate.

- **Widget builder**—You can create custom widgets more quickly, simply, and accurately using the widget builder to extend standard widgets. The widget builder generates the widget code with placeholders for your custom code. As a result, you spend less time writing custom code and your production site goes live sooner.
- **Easy migration**—After you extend standard widgets to create custom widgets, your custom widgets can absorb any future enhancements to the widget you extended.

## Migration overview

**Important** The migration to Framework Version 3.1 offers many benefits to your organization, now and in the future. **But migrating your customer portal requires an investment of your time and energy, and you should not underestimate the extent of this task.** How much is involved depends on how much custom code your site uses, but even if your custom code is minimal, you'll want to review this guide carefully to be sure you understand all the steps before you get started.

Here's a quick overview of the migration process. Note that the previous framework, which has been referred to in earlier documentation as the November 09 framework, is now called Framework Version 2.

### *Overview of the migration process*

- 1 First, review what's new and different in Framework Version 3.1. Refer to [Features and benefits](#).
  - 2 Next, review [Changes to the Customer Portal](#) to understand the differences in file structure and namespacing between Framework Version 2 and Framework Version 3.1.
  - 3 Make copies of all version 2 code that you have customized. This includes templates, pages, custom widgets, CSS files, custom models, and any other custom code. You'll want to have these files as reference when you're customizing the new framework.
-



#### 4 [Enable Framework Version 3.1 on your development pages.](#)

**Note** Your production customer portal remains unchanged, so the customer experience does not change until you're ready to deploy your migrated development pages.

5 Customize your development pages to display and function the way you want them to. This process includes numerous steps, which are described in detail throughout this guide. The main steps include the following.

- a [Modify customer portal elements](#) so they display correctly.
- b [Add the new PasswordInput widget](#) on the Change Password and Create Account pages.
- c [Evaluate functionality changes](#) in version 3.1 to see if you want to include them on your migrated site.
- d [Verify and clean up all labels](#) if necessary.
- e [Make changes to the answer preview pages](#) that are used on the **agent desktop**.
- f [Edit your custom widget code](#) so the widgets work correctly with the new framework.
- g [Edit the remaining custom code.](#)

6 Perform optional code cleanup.

- a [Replace business object names](#) with their **Connect PHP API** equivalent.
- b [Replace unused widgets.](#)

7 Stage and promote your customer portal to make the new framework available on your production site.

**Important** If you had additional page sets enabled in Framework Version 2 (such as mobile) you must select Copy to Staging in the Action drop-down menu for those page sets in the Select Configurations step of the staging process. If you do not copy the page set mappings to staging, those page sets will not be available on your customer portal after you promote the staging environment to production.

8 If you decide that you migrated before you were ready, refer to [Reverting to Framework Version 2](#).

## Features and benefits

Customer Portal Framework Version 3.1 sets the stage for simplified future migration and reduced impact on the customizations you make to your customer portal. It offers control and flexibility to let you decide what features you want to adopt and when you want to adopt them. The ability to extend standard widgets means that you need to write less custom code, and an updated Customer Portal Administration site clearly identifies the version of your framework and each widget and spells out what has changed from previous versions. Major new features include the following.

<b>Framework independence and versioning</b>	The new independent versioning system of the Customer Portal framework lets you upgrade to newer versions of Oracle RightNow CX while leaving your customer portal untouched. You can now control when you want to migrate to a new Customer Portal framework, regardless of the version of Oracle RightNow CX being used by your organization.
<b>Selective adoption of changes</b>	Critical security patches and minimal changes that are completely backward compatible will be applied to your customer portal automatically. Other than these exceptions, which require no work on your part, Framework Version 3 lets you adopt only the features and widgets you want, and only when you want to. Every change is documented on the Customer Portal Administration site to let you know what version of framework and widgets you are using. You can tell if newer versions are available, what the differences are, and whether you're using any incompatible widgets.
<b>Ease of migration</b>	In Framework Version 2, it was fairly common practice to copy a standard widget and then modify it to create a custom widget. Then when you upgraded, those custom widgets were ignored, and you were required to make the changes to your custom code manually instead of automatically getting the new functionality. Now when your custom widgets are extended versions of standard widgets, any change to the standard widget is applied automatically to your custom widget as well (unless you have intentionally overridden that functionality in the custom widget).
<b>New file structure</b>	Framework Version 3 offers an intuitive, easy-to-navigate file structure that lets you clearly identify which files you can edit. Refer to <a href="#">File structure</a> .
<b>Connect PHP API</b>	Framework Version 3 is built on the <b>Connect PHP API</b> and uses data objects from the Connect PHP API rather than internal APIs. The <a href="#">Connect Object Explorer</a> uses a ROQL (RightNow Object Query Language) interface to help you become familiar with Connect more quickly. For information about converting the business objects used in Framework Version 2 to their Framework Version 3 equivalents, refer to <a href="#">Replacing business objects</a> .

<b>PHP namespaces</b>	Framework Version 2 included JavaScript namespaces to prevent naming conflicts between Customer Portal reference implementation files and your custom files. This functionality has been added for PHP in Customer Portal Framework Version 3. The two core namespaces are <code>Rightnow</code> and <code>Custom</code> , and they are designated using a backslash ( <code>\</code> ) notation.
<b>Widget changes</b>	<p>Changes to widgets in Framework Version 3 of the Customer Portal include the following.</p> <p><b>YAML information file</b>—Each widget includes a YAML information file that describes the widget’s dependencies, requirements, attributes, and URL parameters. Refer to <a href="#">The info.yml file</a>.</p> <p><b>Widget version management</b>—Widget versioning lets you adopt the widget features you want at your convenience without requiring you to replace all widgets on your customer portal.</p> <p><b>Widget information page</b>—The Customer Portal Administration site provides comprehensive information about each standard and custom widget. Each widget page includes a preview of the widget, including the available version, recent changes, and the pages and templates that use the widget.</p>
<b>Widget builder</b>	The widget builder in Framework Version 3 helps you create new widgets by extending existing widgets. Then, when functionality or bug fixes are added to a standard widget that was used to extend a custom widget, the custom widget also has access to the enhancement or fix. The widget builder generates custom widget code when you answer a series of questions about the widget you want to create, thereby reducing the amount of custom code you must write yourself. Even widgets you create from scratch using the widget builder contain all the necessary files with placeholder information.
<b>Software modernization</b>	The implementation of YUI 3.8.1 in Customer Portal Framework Version 3.1 offers enhanced performance and coding simplicity. Multiple modules are loaded automatically, and widgets can declare any additional YUI module dependencies in the widget’s YAML information file.
<b>Basic page set</b>	Framework Version 3.1 includes support for earlier versions of mobile devices and browsers that have limited or no support for JavaScript. This page set lets customers using older versions of the Japanese Keitai phones and Symbian OS devices access your customer portal. Implementing this page set significantly increases the number of people who can access your support site since these devices are still in wide use internationally. Framework Version 3.1 automatically detects the capability of the browsers your customers are using and delivers the appropriate page set.

---

<b>Code Assistant</b>	The Code Assistant tool provides specific help for migration to Framework Version 3 by helping you migrate widgets, convert from YUI 2 to YUI 3, and remove deprecated pages.
-----------------------	---

---

## Changes to the Customer Portal

Your customers may not notice anything different when they visit your migrated customer portal since the Framework Version 3 reference implementation looks the same as version 2. However, you'll notice significant changes when you start working with Framework Version 3 code. To prepare for working with the new framework, you'll want to familiarize yourself with the following changes.

- [File structure](#)
- [Widget changes](#)
- [PHP namespacing](#)
- [Syndicated widgets folder](#)
- [Dreamweaver support and extension](#)

### File structure

The customer portal uses the WebDAV protocol to help you manage your website files. WebDAV offers a familiar file structure, easy uploading and downloading of multiple files, and file security through login access. After you set up a WebDAV connection to upload and download files, you can use any text editor you want to create and edit files for your customer portal, or you can use Adobe Dreamweaver, which uses its own WebDAV protocol for site management and configuration.

**Tip** Oracle recommends using Cyberduck for WebDAV access to Customer Portal files. Cyberduck is donationware, and Customer Portal developers appreciate its speed and robust performance on Mac and PC platforms. [Download Cyberduck here.](#)

While Framework Version 2 included ten directories under its */enf/* root directory, version 3 includes a simplified file structure containing the following main folders under the */cp/* root directory.

- *core*—All the non-editable Customer Portal files that make up the framework.
-

- *customer*—The Customer Portal files you can edit, including all pages and custom widget files used on your development site, as well as assets (CSS, images, and themes) and error pages.
- *generated*—All staged files as well as all of the files you have promoted to your production site. Upon migration, the files in this folder are those from your Framework Version 2 *staging* and *production* folders.
- *logs*—Logs for each staging, promoting, and rollback operation.

The following table lets you know where Framework Version 2 folders now appear in the Framework Version 3 directory structure.

Table 1: Version 2 Folders Mapped to the Version 3 Location

Framework Version 2 folder	Framework Version 3 folder
/euf/assets/	/cp/customer/assets/
/euf/config/	/cp/customer/error/
/euf/development/	/cp/customer/development/
/euf/production/	/cp/generated/production/
/euf/rightnow/	/cp/core/framework/
/euf/staging/	/cp/generated/staging/
/euf/staging_assets/	/cp/generated/staging/assets/
/euf/system_assets/	/cp/core/assets/

## The customer folder

The *customer* folder is where you'll spend most of your time editing templates, pages, widgets, and assets. It contains the following subfolders.

- **assets**—As was true in Framework Version 2, this folder contains the assets you can add, edit, and delete. With the exception of the *themes* subfolder, all other files in the *assets* folder are shared by development and production files.

**Important** Files in the *assets* folder are shared between Framework versions 2 and 3. You will probably use Framework Version 2 as your production site while you customize the new framework. Keep in mind that if you edit an asset for the new framework that has the same name as a version 2 asset, your production site will use the edited asset.

- **development**—The *development* folder is your working folder, containing all the templates, pages, and widgets for your customer portal. You'll work with files in the *views* folder to edit pages and templates. Widget files are in the *widgets* folder, except for their presentation CSS files, which are in `/cp/customer/assets/themes/standard/widgetCss`.
- **error**—Contains three editable error pages and *splash.html*, which is a splash page displayed to customers when your site is being migrated.

## Assets in the /euf/ folder

**Important** Although all the customer portal files you can edit are found in the `/cp/customer` folder, some reference implementation files reference the Version 2 `/euf/` root directory. Themes and assets, such as JavaScript, images, and CSS, are still served from `/euf/assets`. These files are actually stored in `/cp/customer/assets`, which you can access through WebDAV, but the path you use to reference them in page code must be `/euf/assets`.

Assume, for example, that you have created a custom theme called *new\_theme*. All files for the custom theme reside in `/cp/customer/assets/themes/new_theme`. But when you define the theme in page or template code, your code will resemble the following.

```
<rn:theme path="/euf/assets/themes/new_theme" ... />
```

To eliminate confusion when you specify images and other assets, you should use paths that are relative to your theme. For example, place images in your *new\_theme* folder in an *images* subfolder (`themes/new_theme/images`) so that the code that defines an image needs to specify only the relative path, as shown in this example.

```

```

---

## Widget changes

This version of the Customer Portal introduces widgets that are versioned independently of the framework. In version 2, migrating to a new Customer Portal framework meant replacing all the widgets associated with the older version. The independence between widgets and framework means that you can now migrate to a newer version of the framework but still keep your earlier widgets. Or you can migrate only those newer widgets that have features you want to adopt (as long as they are compatible with the older framework). The following changes to Customer Portal widgets are part of the new framework.

- [Widget file structure](#)
- [Ability to extend standard widgets](#)
- [Removal of widget suffixes](#)
- [Migration of custom widgets](#)

### Widget file structure

Customer Portal Framework Version 3 uses a separate widget information file, *info.yml*, instead of calling out components in the meta section of a widget's *view.php* file and attributes in the widget's *controller.php* file. Consequently, the Customer Portal Administration site no longer contains a Widget Meta Tags page under Tag Documentation. The information file is written in YAML format, which contains no executable code. Refer to [Completing the widget information file](#).

Unlike in the meta declaration in Framework Version 2, no controller file, logic file, or CSS files are defined in the Framework Version 3 *info.yml* file. That's because the widget folder contains *controller.php*, *logic.js*, and *view.php* files, and the */cp/core/assets/default/themes* subfolders contain the base and presentation CSS files for the KeywordText widget. When the appropriate controller, logic, view, and CSS files exist for the widget, you don't need to specifically identify them in the *info.yml* file.

### Ability to extend standard widgets

Customizing widgets has been simplified in Framework Version 3 with the new widget builder. You can create a custom widget by extending a standard widget. The widget builder walks you through a series of questions, letting you define which attributes of the existing widget you want to use and also letting you create new attributes. Then, based on the widget definition you supply, the widget builder generates basic code, which you can edit to complete the customization. Instead of requiring you to write custom code to make minor modifications to standard widgets, the widget builder eliminates the need for copying, pasting, and modifying code. You can also use the widget builder to create skeleton code for new widgets that are not extended from another widget.

## Removal of widget suffixes

The following Framework Version 2 widgets have had their numerical suffix removed in Framework Version 3. For example, the widget called ChatHours2 in Framework Version 2 is renamed ChatHours in Framework Version 3, and AnswerNotificationIcon3 is now simply AnswerNotificationIcon. In Framework Version 2, the suffixes served to distinguish between different versions of the widget, but all widgets are versioned in Framework Version 3 so a name change is not necessary when a widget changes.

• chat/ChatHours2	• login/ResetPassword2
• chat/ChatStatus2	• navigation/NavigationTab2
• chat/ProactiveChat2	• notifications/ AnswerNotificationIcon3
• feedback/AnswerFeedback2	• notifications/Unsubscribe2
• feedback/SiteFeedback2	• reports/Grid2
• input/FileAttachmentUpload2	• reportsMultiline2
• knowledgebase/PreviousAnswers2	• reports/ResultInfo2
• knowledgebase/RelatedAnswers2	• reports/SearchTruncation2
• knowledgebase/RssIcon2	• search/FilterDropdown2
• knowledgebase/SearchSuggestions2	• search/KeywordText2
• knowledgebase/TopicWords2	• search/OrgList2
• login/EmailCredentials2	• search/SearchButton2
• login/LoginDialog2	• search/SearchTypeList2
• login/LoginForm2	• search/SortList2
• login/LogoutLink2	• utils/AnnouncementText2



## Migration of custom widgets

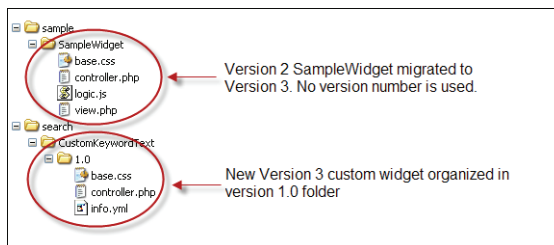
The migration process will copy your Framework Version 2 custom widgets to Framework Version 3.1, but you must edit the code in order to have the widgets function correctly on your migrated customer portal.

**Tip** In general, if your custom widget was a modified version of a standard widget, it will probably be easier for you to re-create the widget in the widget builder by extending the Framework Version 3 standard widget and then editing the code to duplicate its Version 2 functionality.

If your custom widget was created from scratch to implement functionality that was not available in the customer portal, you'll probably be better off to convert the widget code to work with Framework Version 3 (although using the widget builder to create the file structure might accelerate the process). Refer to [Converting custom widgets](#).

All files associated with a custom widget are copied to the Framework Version 3.1 site. Custom widget files can be found under the *widgets/custom* folder, but they do not have the Framework Version 3 file structure when they are migrated from your Framework Version 2 customer portal. That is, the custom widget files will not be placed in a version folder the way that Version 3 custom widgets are.

In this figure, the Version 2 *SampleWidget* files are copied to the new site in the *widgets/custom/sample/SampleWidget* folder, but the Version 2 files appear at the same level as the 1.0 folder that contains the Version 3 *SampleWidget* files. The directory structure resembles the following.



Other custom files are also migrated to the newer framework. For example, the sample custom model (*sample\_model.php* in the */enf/development/models/custom/* folder) is migrated to the */cp/customer/development/models/custom* folder. Although it appears in the *models/custom* folder, the migrated *sample\_model.php* file does not work with Framework Version 3, so you should use *Sample.php* in the same folder instead.

## PHP namespacing

Framework Version 2 included JavaScript namespacing to prevent naming conflicts between Customer Portal reference implementation files and your custom files. This functionality has been added for PHP in Customer Portal Framework Version 3.

The two core namespaces are `RightNow` and `Custom`, and they are designated using a backslash (`\`) notation. The namespaces are used for files, widgets, and libraries. The following table lists the primary namespaces in Customer Portal Framework Version 3 and notes the folder for each.

Table 2: Primary Customer Portal Namespaces in PHP

Namespace	Folder
<code>RightNow\Controllers</code>	<code>/cp/core/framework/Controllers</code>
<code>RightNow\Libraries</code>	<code>/cp/core/framework/Libraries</code>
<code>RightNow\Libraries\Widget</code>	<code>/cp/core/framework/Libraries/Widget</code>
<code>RightNow\Models</code>	<code>/cp/core/framework/Models</code>
<code>RightNow\Utils</code>	<code>/cp/core/framework/Utils</code>
<code>RightNow\Widgets</code>	<code>/cp/core/widgets/standard</code>
<code>Custom\Models</code>	<code>/cp/customer/development/models</code>
<code>Custom\Widgets</code>	<code>/cp/customer/development/widgets/custom</code>

Subfolders are appended to the namespace for custom models and custom widgets, as shown by the examples in the following table.

Table 3: Custom Namespaces and Folders

Namespace	Folder
Custom\Models\Test	/cp/customer/development/models/Test
Custom\Models\test\subtest	/cp/customer/development/models/test/subtest
Custom\Widgets\input	/cp/customer/development/widgets/custom/input
Custom\Widgets\test\subtest	/cp/customer/development/widgets/custom/test/subtest

## Syndicated widgets folder

Syndicated widgets let you use the functionality of a standard customer portal widget, including access to the knowledge base, on web pages that are not part of your customer portal page set. In Framework Version 2, files for syndicated widgets appeared in the */euf/rightnow/syndicated\_widgets/standard* folder, and you could view the code for the syndicated widgets' controller, logic, and view files in WebDAV. Except for widget CSS, which you could edit directly through WebDAV, you made all other changes to the syndicated widgets on the Customer Portal Administration site, which generated the code for your modifications. Framework Version 3 offers the same Framework Version 2 syndicated widgets with the same ability to edit CSS directly and modify functionality on the Customer Portal Administration site. The only difference between the two frameworks with respect to syndicated widgets is that you cannot view syndicated widget code (except CSS) through WebDAV in Framework Version 3.

## Dreamweaver support and extension

Although you can still use Dreamweaver as a text editor for working with your customer portal files, you won't be able to use the extension that was available in earlier frameworks. If, however, you revert to Framework Version 2 after migrating to Framework Version 3, you can use the extension again. You must install it from the Customer Portal Administration site, since it is no longer available as a download link on the CX Console.

## Enabling Framework Version 3.1

The following procedure describes how to enable the new framework on your customer portal so you can begin the migration process.

**Important** When you complete this procedure, only your development pages will be on Framework Version 3. Your staging and production areas are untouched at this point, and they will remain on version 2 until you:

- 1 Edit your custom code.
- 2 Make any other necessary version 3 changes.
- 3 Stage and promote your development site.

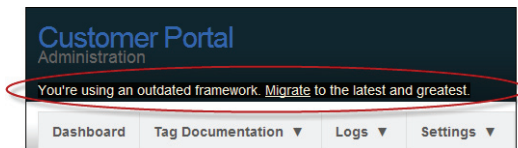
After you click the Switch to the New Framework button, your Development Area will be on version 3. **However, you must still modify your custom files before you can stage and deploy the new framework without error.** This effort can be significant, depending on the amount of customization you've done. Additionally, there are other steps you must take before deploying the new version to your production site.

You also need to keep in mind that assets are going to be shared between the two framework versions. This means that any Framework Version 3 edits you make to an asset that is also used on version 2 (a graphic, for example) will show up on your production site since it is still running Framework Version 2.

One last point to remember is that you can always change your mind and [revert your development pages to Framework Version 2](#) if you run into problems.

### *To enable Framework Version 3 on your customer portal*

- 1 If you have not already copied all of your Framework Version 2 customized files to a safe location before you begin, do so now before continuing the procedure.
  - 2 Type `https://<your_site>/ci/admin` and enter your user name and password to log in to the Customer Portal Administration site. When the dashboard opens, the page header notifies you that your framework is outdated.
-



- 3 Click the Migrate link in the header. The New Framework Migration page appears.

### New framework migration

Switch to the new framework version

Set the Development Environment to version 3.

**WARNING:** Migrating to version 3 of the Customer Portal Framework will require changes to your custom code. Development mode on your site will not be in a production-ready state until your custom code has been migrated to work with the new framework.

Things to know

- The migration work depends on the extent of your customization. Refer to the migration guide to understand the process before you start.
- Only your Development area is migrated. You'll need to stage and promote before your customers can see it.
- You can change your mind and revert to Framework Version 2.
- If you've already edited version 3 files (by migrating to version 3, editing files, and then reverting to version 2), that work will be lost when you migrate if you select Yes for the option to copy custom files to version 3.
- Assets are shared between versions 2 and 3. If you edit an asset while working on version 3, and if that asset is also used in version 2, those changes will be visible in production.

Would you like to copy custom files to version 3? Both options require you to edit your custom code after migration.

Yes [if you have significant custom code]
  No [if you have not customized your site at all]

Switch to the new framework -->

- 4 To start with a brand new Framework Version 3.1 site that does not carry over any of your Framework Version 2 customization, select No.

**Note** This option is recommended primarily if you have not customized your customer portal files.

- 5 To copy the files you have customized in Framework Version 2 (including pages, templates, widgets, models, controllers, and CSS files) to Framework Version 3.1, select Yes.
- 6 Click the Switch to the New Framework button. A confirmation page lets you know you were successful. The page also contains links to resources, including this Migration Guide, to help you complete the migration process. To review the links, refer to [Migration resources](#).

**Framework Version 3.1 is now active in the CP Development Area**

---

Now that your development pages use version 3, you can begin migrating your Customer Portal code. You must modify your custom files before you can stage and deploy the new framework without error. This effort can be significant, depending on the amount of customization you've done. There are additional steps you must take before deploying the new version to your production site.

**Next Steps and Migration Resources**

- Go to the online Migration Guide and follow the migration steps.
- 3.1 Framework Changelog
- View your development area to track your migration progress \*
- \* Custom files brought over from earlier CP frameworks may cause errors in the development pages and will likely need to be modified to work with the new framework.
- Code Assistant (A user interface for improving/migrating custom code)

**Get Help**

- Customer Portal documentation
- Professional Services Offerings
- Customer Portal API Documentation
- Revert to Framework Version 2

**Get Involved**

- Customer Portal Discussion Board
- Make Suggestions

**Additional Resources**

- Preview Error Pages
- Firebug
- YUI (Current version: 3.8.1)
- PHP.net (Current version: 5.3)
- W3schools - CSS

**Note** Before you view your migrated development pages, you'll need to copy some asset files, as described in step 7 below. These files are necessary for the correct display of your migrated site.

- 7 Open your customer portal directory structure using a WebDAV connection, locate `/cp/core/assets/default`, and copy all files in that folder.
- 8 Locate the `/cp/customer/assets` folder and paste the copied files into it.
- 9 To undo the Framework Version 3.1 migration if you run into problems or change your mind, [revert your development pages to Framework Version 2](#).

## Migration resources

The confirmation page that lets you know your migration to Framework Version 3.1 was successful contains links to many useful resources that can help you get the most from your customer portal.

**Framework Version 3.1 is now active in the CP Development Area**

---

Now that your development pages use version 3, you can begin migrating your Customer Portal code. You must modify your custom files before you can stage and deploy the new framework without error. This effort can be significant, depending on the amount of customization you've done. There are additional steps you must take before deploying the new version to your production site.

**Next Steps and Migration Resources**

- Go to the online Migration Guide and follow the migration steps.
- 3.1 Framework Changelog
- View your development area to track your migration progress \*
- \* Custom files brought over from earlier CP frameworks may cause errors in the development pages and will likely need to be modified to work with the new framework.
- Code Assistant (A user interface for improving/migrating custom code)

**Get Help**

- Customer Portal documentation
- Professional Services Offerings
- Customer Portal API Documentation
- Revert to Framework Version 2

**Get Involved**

- Customer Portal Discussion Board
- Make Suggestions

**Additional Resources**

- Preview Error Pages
- Firebug
- YUI (Current version: 3.8.1)
- PHP.net (Current version: 5.3)
- W3schools - CSS

Links on this page include the following:

- **Next Steps and Migration Resources**
  - ▷ **Go to the online Migration Guide and follow the migration steps**—Includes a link to a PDF version of this migration documentation.
  - ▷ **3.1 Framework Changelog**—Lists changelog entries that describe the changes in the current and past framework and any action you might need to take to incorporate the changes on your migrated site. The changelog identifies changes as new features, removals, API changes, or bug fixes.
  - ▷ **View your development area to track your migration process**—Opens the migrated version of your customer portal in development mode so you can see what changes you might need to make to bring them up to Framework Version 3.1.
  - ▷ **Code Assistant**—Opens the Customer Portal Code Assistant to help you migrate your custom code.
- **Get Help**
  - ▷ **Customer Portal Documentation**—Opens the Customer Portal section of the online Oracle RightNow CX User Guide.

- ▷ **Professional Services Offerings**—Links to the [Oracle RightNow Professional Services](#) page, if you should want to engage Professional Services to help you migrate your customer portal.
- ▷ **Customer Portal API Documentation**—Provides API documentation for PHP and JavaScript.
- ▷ **Revert to Framework Version 2**—Loads the page that lets you switch to the old framework version again. See [Reverting to Framework Version 2](#).
- **Get Involved**
  - ▷ **Customer Portal Discussion Board**—Opens the forum where developers share questions and solutions for common configuration issues.
  - ▷ **Make Suggestions**—Links to the Idea Lab, where you can submit product ideas and enhancement requests.
- **Additional Resources**
  - ▷ **Preview Error Pages**—Links to your site's error pages.
  - ▷ **Firebug**—Links to the web development tool for use with Mozilla Firefox.
  - ▷ **YUI**—Links to the YUI Library.
  - ▷ **PHP.net**—Links to PHP documentation.
  - ▷ **W3schools – CSS**—Links to a CSS tutorial.

## Reverting to Framework Version 2

If you have migrated from Customer Portal Framework Version 2 to Framework Version 3, you have the option of switching your development environment back to the older version. If you have already staged and promoted the Framework Version 3 pages to your production site and you want to revert them as well, you'll need to stage and promote the development pages you revert to Framework Version 2. Remember that all assets are shared between the two frameworks.

### *To revert to Framework Version 2*

- 1 Type `https://<your_site>/ci/admin` and enter your user name and password to log in to the Customer Portal Administration site.
  - 2 Click the Framework tab and select Old Framework Migration.
-



- 3 Click the Switch Back to V2 button. When the reversion is complete, the Support Home page for Framework Version 2 opens in development mode. The Framework Version 2 file structure is now available for editing through WebDAV.

**Note** The WebDAV file structure you now see uses the Version 2 */en/* root directory.

- 4 To make the Framework Version 2 pages available to customers on your production site, stage and promote the customer portal.

When you return to the Customer Portal Administration site, the header displays the message that you are using an outdated framework and offers you the link for migrating to Framework Version 3.

[When you're ready, you can enable Framework Version 3 again.](#)

**Note** If you had been working on development pages in Framework Version 3 prior to reverting to version 2 (if, for example, you needed to make a change to your production site that was still on version 2), when you enable Framework Version 3 again, your previous edits to version 3 still remain.

If you want a non-edited set of files for version 3, that is, if you want to overwrite all your previous version 3 changes and begin work with the out-of-the-box reference implementation, you can copy those files from the *core* directory and paste them into your development site.

## Code editing overview

We'll guide you through this process with a series of questions. When a section doesn't apply to your customer portal, clicking the appropriate link lets you skip over that section, moving on instead to the next area you want to edit.

The main categories include:

- [Modifying display elements](#)
- [Adding the PasswordInput widget](#)
- [Evaluating other functionality changes](#)
- [Adding label attributes](#)
- [Customer portal changes on the agent desktop](#)
- [Converting custom widgets](#)
- [Converting custom code](#)

- [Optional code cleanup](#)

You can always come back to this page if you lose your place or want to start over.

Additionally, you might find the following resources helpful as you migrate your customer portal.

- [Oracle RightNow Connect PHP API Cloud Service \(Connect PHP API\) reference documentation](#)
- [Customer Portal API Documentation for PHP and JavaScript](#)
- [YUI reference documentation](#)
- [PHP language reference documentation](#)
- [Customer Portal discussion board](#)

## Modifying display elements

If you carried over your customized Framework Version 2 files during the migration process (by selecting Yes as described in step 5 of [To enable Framework Version 3 on your customer portal](#)), you will need to make several file modifications so that your migrated customer portal correctly displays the elements that were changed in Framework Version 3. The following sections walk you through the necessary procedures.

- [Using HTML5 on custom pages](#)
- [Editing widgets that open in a separate window](#)
- [Changing the width of the Email Address field](#)
- [Editing the width of the OpenLogin dialog](#)
- [Modifying the SocialBookmarkLink widget](#)
- [YUI resources](#)

## Using HTML5 on custom pages

All standard widgets and pages in the Customer Portal reference implementation conform to the HTML5 specification. We recommend that you use the HTML5 doctype on your custom pages and templates because some Framework Version 3 widgets take advantage of it.

### *To use HTML5*

- 1 Review the code for your custom pages and templates to look for the following line of code at the top of any of the custom files.
-

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
/www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

- 2 If you find the code from step 1 in any of the files, change it to the following.

```
<!DOCTYPE html>
```

- 3 Save the file.

## Editing widgets that open in a separate window

The replacement of the Framework Version 2 YUI 2.7 libraries with YUI 3 in Framework Version 3 means you'll need to edit the standard template so that the LoginDialog, AdvancedSearchDialog, and ProductCategoryInput widgets display properly. This procedure also corrects issues with pop-up hints that display on input fields.

Code has been simplified for calling YUI modules, allowing you to use a {YUI} substitution instead of a full path, including version number. Future updates of YUI will be automatically available without requiring you to edit paths after you have replaced template code and other code that calls YUI modules with the substitution.

### *To display widgets correctly*

**Note** This procedure assumes that you use the *standard.php* reference implementation template. If you use a different template or templates, substitute your template file name for *standard.php* in this procedure, which is required for any template that specifies a theme.

- 1 Open the *standard.php* file in the *customer/development/views/templates* directory.

- 2 Locate the following line of code.

```
<rn:theme path="/euf/assets/themes/standard" css="site.css,/rnt/rnw/
yui_2.7/container/assets/skins/sam/container.css" />
```

- 3 Change the code you located in step 2 to the following.

```
<rn:theme path="/euf/assets/themes/standard" css="site.css,
{YUI}/widget-stack/assets/skins/sam/widget-stack.css,
{YUI}/widget-modality/assets/skins/sam/widget-modality.css,
{YUI}/overlay/assets/overlay-core.css,
{YUI}/panel/assets/skins/sam/panel.css" />
```

- 4 Locate the following line of code.

```
<body class="yui-skin-sam">
```

- 5 Change the code you located in step 4 to the following.

```
<body class="yui-skin-sam yui3-skin-sam">
```

- 6 Save *standard.php*.

## Changing the width of the Email Address field

Email fields have been updated in Framework Version 3 to use HTML 5 input types, so email fields are now `<input type="email">` rather than `<input type="text">`. The result on migrated sites is that the Email Address field is narrower, but you can adjust the width so it appears as it did on your Framework Version 2 site by using the following procedure.

### *To change the width of the Email Address field*

**Note** If you followed the procedure in [To edit the site CSS file for accessibility](#), these changes already appear in your code.

- 1 Open the *site.css* file in the *customer/assets/themes/standard* directory.
- 2 Locate the following lines of code.

```
input [type="text"], input [type="password"] {
  height:18px;
}
```

- 3 Change the code you located in step 2 to the following.

```
input [type="text"], input [type="password"], input [type="email"]
input [type="url"] {
  height:18px;
}
```

- 4 Save *site.css*.
- 5 Open the *TextInput.css* file in *customer/assets/themes/standard/widgetCss*.
- 6 Locate the following lines of code.

```
.rn_TextInput .rn_Text,
.rn_TextInput .rn_TextArea {
  width:60%;
}
```

- 7 Change the code you located in step 6 to the following.

```
.rn_TextInput .rn_Text,
.rn_TextInput .rn_Email,
.rn_TextInput .rn_Url,
.rn_TextInput .rn_TextArea {
width:60%;
}
```

- 8 Save *TextInput.css*.

**Note** Integer input fields do not match the width of other text input fields in Framework Version 3. Instead, in Chrome, Firefox, and Safari browsers, integer input types use up and down arrows to select numerical input values, so the narrower field is possible.

## Editing the width of the OpenLogin dialog

The dialog that opens when your customers click one of the open login icons (for example, Facebook, Twitter, or AOL) needs to be adjusted after migrating to Framework Version 3.

*To edit the width of dialog for OpenLogin widget*

- 1 Open the *OpenLogin.css* file in the *customer/assets/themes/standard/widgetCss* directory.
- 2 Add the following code to the CSS file.

```
.rn_OpenLoginDialog {
width: 330px;
}
```

- 3 Save *OpenLogin.css*.

## Modifying the SocialBookmarkLink widget

The SocialBookmarkLink in Framework Version 3 offers Facebook, Twitter, LinkedIn, and Reddit options, eliminating Delicious, Digg, and StumbleUpon, which were used in Version 2, and adding LinkedIn. If you want to continue using the earlier social networking sites with the widget, edit it according to this procedure.

*To display the Framework Version 2 social networking sites*

- 1 Open *detail.php* in the */cp/customer/development/views/pages/answers* folder.

- 2 Locate the following code.

```
<rn:widget path="utils/SocialBookmarkLink" />
```

- 3 Replace the code you found in step 2 with the following.

```
<rn:widget path="utils/SocialBookmarkLink"
sites="Delicious > Post to Delicious > http://del.icio.us/
post?url=|URL|&title=|TITLE|,
Digg > Post to Digg > http://digg.com/submit?url=|URL|&title=|TITLE|,
Facebook > Post to Facebook > http://facebook.com/sharer.php?u=|URL|,
Reddit > Post to Reddit > http://reddit.com/
submit?url=|URL|&title=|TITLE|,
StumbleUpon > Post to StumbleUpon > http://stumbleupon.com/
submit?url=|URL|&title=|TITLE|,
Twitter > Tweet this > http://twitter.com/home?status=|TITLE| |URL| />
```

- 4 Save *detail.php*.

- 5 Open *SocialBookmarkLink.css* in the */cp/customer/assets/themes/standard/widgetCss* folder.

- 6 Locate the following lines of code.

```
/* Facebook */
.rn_SocialBookmarkLink li.rn_Link1 {
    background-position:0 -32px;
}
/* Twitter */
.rn_SocialBookmarkLink li.rn_Link2 {
    background-position:0 -80px;
}
/* LinkedIn */
.rn_SocialBookmarkLink li.rn_Link3 {
    background-position:0 -96px;
}
/* Reddit */
.rn_SocialBookmarkLink li.rn_Link4 {
    background-position:0 -48px;
}
```

- 7 Replace the code you found in step 6 with the following code.

```
/* Delicious */
.rn_SocialBookmarkLink li.rn_Link1 {
    background-position:0 0;
}
```

---

```

/* Digg */
.rn_SocialBookmarkLink li.rn_Link2 {
    background-position:0 -16px;
}
/* Facebook */
.rn_SocialBookmarkLink li.rn_Link3 {
    background-position:0 -32px;
}
/* Reddit */
.rn_SocialBookmarkLink li.rn_Link4 {
    background-position:0 -48px;
}
/* StumbleUpon */
.rn_SocialBookmarkLink li.rn_Link5 {
    background-position:0 -64px;
}
/* Twitter */
.rn_SocialBookmarkLink li.rn_Link6 {
    background-position:0 -80px;
}

```

8 Save *SocialBookmarkLink.css*.

## YUI resources

The implementation of YUI 3.8.1 in Customer Portal Framework Version 3.1 will likely require additional modifications to your migrated code, particularly if your Version 2 custom widgets include calls to YUI modules. Because the YUI library is extensive and the range of customizations can vary widely, it is beyond the scope of this guide to describe all the possible migration requirements. However, the following links may provide some help as you migrate your custom code that includes YUI calls.

- [Widget page, YUI Library](#)—This page describes the YUI Widget, which is the foundation class used by all YUI 3 widgets. For example, the Customer Portal dialogs that open when a link or button is clicked are considered YUI widgets. The page describes class structure and responsibilities, basic attributes, rendering methods, progressive enhancement, rendered markup, class names and CSS, and default UI events. It also contains a section for developing your own YUI widgets and information about plugins and extensions.

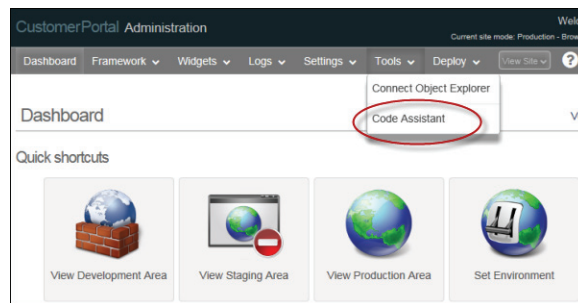
- ▷ [Class Names and CSS](#)—This page describes how the YUI Widget class uses the instance and static methods to generate class names and discusses the CSS class that is applied for the visible, disabled, and focused states.
- [Migration Table](#)—The table provides examples for migrating the YUI 2 `YAHOO.util.Dom` functionality to the YUI 3 Node utility.

## Using Code Assistant to migrate YUI code

The Code Assistant locates and identifies references to YUI 2 in your custom code and then offers help in editing the code for YUI 3. After you have migrated your customer portal to Framework Version 3, you can use Code Assistant.

### *To use Code Assistant to migrate YUI code*

- 1 On the Customer Portal Administration site, hover over Tools and select Code Assistant.





















- 2 On the Select Your Versions screen, verify that the production mode is Pre-3.0 and the development mode is version 3.1. If necessary, click the drop-down menu and make the correct selections.
- 3 Click Next Step.
- 4 On the Select an Operation from the List screen, select the YUI 2 to YUI 3 Suggestions radio button and click Continue. A list of JavaScript files containing a YUI version earlier than YUI 3 appears.



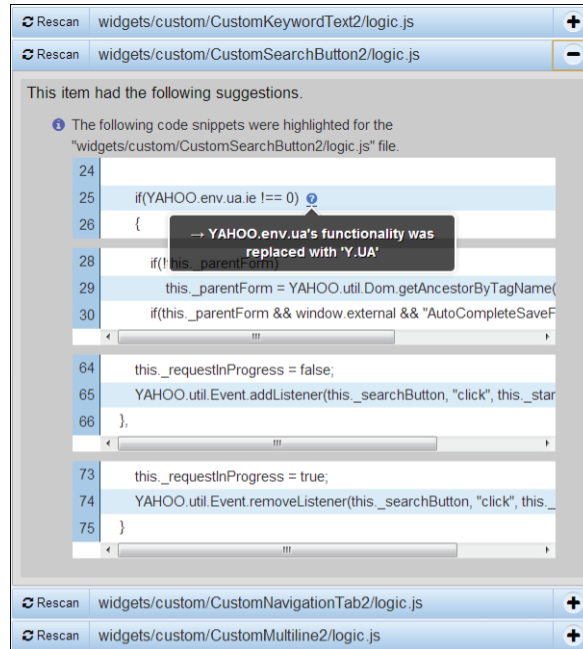
### 3. Select an Option to View Suggestions

The following files use pre YUI 3.0 in their JavaScript code. Pre YUI 3.0 is no longer supported with CP Framework 3. *In order for custom code to work correctly in CP Framework 3 these references should be removed or replaced with similar functionality using the YUI 3 framework. Please look through your code and manually make changes to each of these files.* When some changes have been made, you can use the rescans button to remove old references.

 Rescan	widgets/custom/CustomKeywordText2/logic.js	
 Rescan	widgets/custom/CustomSearchButton2/logic.js	
 Rescan	widgets/custom/CustomNavigationTab2/logic.js	
 Rescan	widgets/custom/CustomMultiline2/logic.js	
 Rescan	widgets/custom/CustomFilterDropdown2/logic.js	
 Rescan	widgets/custom/CustomSiteFeedback2/logic.js	
 Rescan	widgets/custom/CustomLoginForm2/logic.js	
 Rescan	widgets/custom/CustomAdvancedSearchDialog/logic.js	
 Rescan	widgets/custom/CustomDisplaySearchFilters/logic.js	

[Home](#) Start Over →

- 5 Click the + sign for the file to expand the suggestions for making changes so the file complies with YUI 3. Mouse over the question mark icons to see how the functionality was changed in YUI 3.



6 Manually edit each of the *logic.js* files to make the Code Assistant suggestions for the file.

## Adding the PasswordInput widget

Framework Version 3 includes the new PasswordInput widget on the Change Your Password page to replace the two FormInput widgets that created the Password and Verify Password fields in Version 2. Because the Verify Password field is added automatically with the new widget, you need to remove the `contacts.password_verify` object from your Version 2 code to avoid an error in development mode. It's a good idea to also add the PasswordInput widget to the Create an Account page.

The files for the PasswordInput widget are in the *core/widgets/standard/input* folder.

*To add the PasswordInput widget to the Change Your Password page*

- 1 Open the *change\_password.php* file in the *development/views/pages/account* folder.
- 2 Locate the following lines of code.

```
<rn:widget path="input/FormInput "
```

```

name="contacts.password"
required="false"
label_input="#rn:msg:CURRENT_PASSWORD_LBL#"
initial_focus="true" />
<rn:widget path="input/FormInput"
name="contacts.password_new"
required="false"
label_input="#rn:msg:ENTER_NEW_PASSWD_LBL#" />
<rn:widget path="input/FormInput"
name="contacts.password_verify"
required="false"
label_input="#rn:msg:CONFIRM_NEW_PASSWD_LBL#" />

```

- 3 Replace the code you located in step 2 with the following.

```

<rn:widget path="input/PasswordInput"
name="Contact.NewPassword"
require_validation="true"
require_current_password="true"
label_input="#rn:msg:PASSWORD_LBL#"
label_validation="#rn:msg:VERIFY_PASSWD_LBL#"
initial_focus="true"/>

```

- 4 Save *change\_password.php*.

### *To add the PasswordInput widget to the Create an Account page*

- 1 Open the *create\_account.php* file.

- 2 Locate the following lines of code.

```

<rn:widget path="input/FormInput" name="contacts.password_new" />
<rn:widget path="input/FormInput" name="contacts.password_verify" />

```

- 3 Replace the code you located in step 2 with the following.

```

<rn:widget path="input/FormInput"
name="Contact.NewPassword"
require_validation="true"
label_input="#rn:msg:PASSWD_LBL#"
label_validation="#rn:msg:VERIFY_PASSWD_LBL#" />

```

- 4 Save *create\_account.php*.

## Setting password length

In Framework Version 2, you could set the required password length differently on different interfaces using the `MYSEC_MIN_PASSWD_LEN` **configuration setting**. In Framework Version 3, the password requirements, including password length, that you define on the Contact Password Configuration editor of the **CX Console** apply to all interfaces. When you migrate to Oracle RightNow CX, the password length that now appears on the Contact Password Configuration editor is the value from the Version 2 interface that has the longest password requirement.

## Evaluating other functionality changes

The Framework Version 3.1 reference implementation includes some differences in functionality from Framework Version 2 that you may want to adopt on your migrated customer portal to make it more similar to the new framework. Or, alternately, you may want to modify the default Version 3.1 behavior so that it resembles what you had in Version 2. You can make the following changes.

- [Using high-contrast mode for accessibility](#)—Make these changes if your customer portal pages must be viewable in high-contrast mode.
  - [Disabling incident receipt emails](#)—To disable the automatic sending of **email receipts** to customers who submit questions, which is the default behavior in Framework Version 3, follow the procedure in this section.
  - [Changing incident source for rules](#)—If you have been using Connect PHP API in Framework Version 2, you will need to change **incident rules** on the **CX Console**.
  - [Removing customer search preferences](#)—Review your code to remove the use of customer search preferences, which are not available in Framework Version 3.
  - [Changing the SmartAssistantDialog widget](#)—The **SmartAssistantDialog** widget includes a Cancel link by default, which you can remove or style as a button.
  - [Using the organization password](#)—Letting customers associate themselves with an **organization** now requires that they also enter the organization's password.
  - [Editing the CommunitySearchResults widget](#)—The `display_initial_posts` attribute of this widget has been removed in this framework.
  - [Adding the Japanese name suffix](#)—A new method exists for adding the Japanese suffix “-san” to names.
  - [Login required configuration setting](#)—If you are migrating from a pre-August 2010 release and want to require login on your customer portal pages, you'll need to edit certain pages that should not require login, for example, the Create an Account page.
-

- [Using the KnowledgeSyndication widget overlay feature](#)—The KnowledgeSyndication widget can now be opened in an overlay rather than requiring a page turn.
- [Redirecting WAP pages to the basic page set](#)—If customers have been accessing your support site from a **WAP interface**, you should now redirect them to the basic page set, which is the WAP replacement.
- [Deprecation of Intent Guide](#)—Intent Guide has been deprecated so you'll need to remove pages if you have been using this feature.
- [Deprecation of SearchTruncation widget](#)—Because the SearchTruncation widget has been deprecated, you must remove it from any pages that are currently using it.

## Using high-contrast mode for accessibility

If you need your customer portal pages to be viewable in high-contrast mode for **accessibility** purposes, you must manually migrate changes to your CSS files to modify the following items.

- The closing x on dialog windows
- Borders on buttons and text fields
- OpenLogin labels
- The arrow on the product/category selector
- The expand/collapse icons on the product/category trees
- Password requirements

The following files must be edited to add high-contrast mode.

- [site.css](#)
- Widget CSS
  - ▷ [AnswerFeedback](#)
  - ▷ [OpenLogin](#)
  - ▷ [ProductCategoryInput](#)
  - ▷ [ProductCategorySearchFilter](#)

### Edit site CSS

*To edit the site CSS file for accessibility*

- 1 Open the *site.css* file in the */cp/customer/assets/themes/standard* folder.
- 2 Locate the following line of code in the Index (around line 19).

## 13. Chat Related

- 3 Add the following line of code immediately after the code you located in step 2.

## 14. High Contrast Mode

- 4 Locate the following line of code (around line 147).

```
input [type="text"], input [type="password"] {
```

- 5 Change the code you located in step 4 to the following.

```
input [type="text"], input [type="password"], input [type="email"],
input [type="url"] {
```

**Note** If you followed the procedure in [To change the width of the Email Address field](#), this change already appears in your code.

- 6 Locate the following line of code for the width attribute (around line 215) under #rn\_Container.

```
width:942px;
```

- 7 Change the code you located in step 6 to the following.

```
width:1025px;
```

- 8 Locate the following line of code for the line height attribute (around line 472) under #rn\_PageTitle.rn\_AnswerDetail.

```
line-height: 1.2;
```

- 9 Change the code you located in step 8 to the following.

```
line-height: 1.2em;
```

- 10 Locate the following line of code for the color attribute (around line 527) under .rn\_HintBox.

```
color:#222
```

- 11 Add the following line immediately below the code you located in step 10.

```
display:block;
```

- 12 Locate the following line of code (around line 538).

```
.rn_Required, .rn_ErrorLabel {
```

- 13 Add the following code immediately above the code you located in step 12:

```
.rn_HintText {
  color:#222;
```

---

```

        display:block;
        font-size:.963em;
        font-style:italic;
        max-width:500px;
        width:99%;
    }

```

14 Locate the following lines of code (around line 586).

```

/*****
Module Box
*****/

```

15 Add the following code immediately above the code you located in step 14:

```

/**Overwriting the Width of yui3-button-close(Panel.css) from 13px to
16px.
    Ticket# 2532849 was opened with YUI for the issue*/
.yui3-skin-sam .rn_Dialog.yui3-panel-content .yui3-widget-hd .yui3-
button-close {
    width: 16px;
}
.rn_MessageDialogIcon.rn_HelpContent,
.rn_MessageDialogIcon.rn_WarningContent,
.rn_MessageDialogIcon.rn_TipContent,
.rn_MessageDialogIcon.rn_AlertContent,
.rn_MessageDialogIcon.rn_AlarmContent,
.rn_MessageDialogIcon.rn_BlockContent {
    background-image: url(images/icons/messageDialog.png);
    background-repeat: no-repeat;
    display: inline-block;
    height: 17px;
    margin-right: 10px;
    vertical-align: sub;
    width: 20px;
}
.rn_MessageDialogIcon.rn_HelpContent {
    background-position:0 -50px;
}
.rn_MessageDialogIcon.rn_WarningContent {
    background-position:0 3px;
}
.rn_MessageDialogIcon.rn_TipContent {

```

```

        background-position:0 -31px;
        height: 17px;
    }
    .rn_MessageDialogIcon.rn_AlertContent {
        background-position:0 -14px;
    }
    .rn_MessageDialogIcon.rn_AlarmContent {
        background-position:0 -86px;
    }
    .rn_MessageDialogIcon.rn_BlockContent {
        background-position:0 -68px;
        height: 17px;
    }
}

```

**16** Locate the following lines of code (around line 646).

```

/*****
Misc Common rules
*****/

```

**17** Add the following code immediately above the code you located in step 16:

```

/*****
Icons for attachments
*****/
.rn_FileTypeIcon {
    background:url(images/icons/fileIcons.png) no-repeat;
    display:inline-block;
    vertical-align:top;
    height:16px;
    width:16px;
}
.rn_FileTypeIcon.rn_url {
    background-position: 0px -17px;
}
.rn_FileTypeIcon.rn_doc,
.rn_FileTypeIcon.rn_docx {
    background-position: 0px -34px;
}
.rn_FileTypeIcon.rn_xls,
.rn_FileTypeIcon.rn_xlsx {
    background-position: 0px -51px;
}

```

---



```

}
.rn_FileTypeIcon.rn_ppt,
.rn_FileTypeIcon.rn_pptx {
    background-position: 0px -68px;
}
.rn_FileTypeIcon.rn_txt,
.rn_FileTypeIcon.rn_rtf {
    background-position: 0px -85px;
}
.rn_FileTypeIcon.rn_gif,
.rn_FileTypeIcon.rn_png,
.rn_FileTypeIcon.rn_jpg,
.rn_FileTypeIcon.rn_jpeg {
    background-position: 0px -102px;
}
.rn_FileTypeIcon.rn_zip {
    background-position: 0px -119px;
}
.rn_FileTypeIcon.rn_pdf {
    background-position: 0px -136px;
}
.rn_FileTypeIcon.rn_rightnow {
    background-position: 0px -153px;
}

```

**18** Locate the following line of code (around line 846).

```
#rn_PageContent.rn_Home .rn_Module ul li img{
```

**19** Change the code you located in step 18 to the following.

```
#rn_PageContent.rn_Home .rn_Module ul li span.rn_FileTypeIcon{
```

**20** Locate the following line of code (around line 907).

```
margin:10px 0px 40px 40px;
```

**21** Change the code you located in step 20 to the following.

```
margin:10px 0px 40px 15px;
```

**22** Locate the following lines of code (around line 997).

```

/*****
13. Chat Related

```

```
*****/
```

**23** Add the following code below the code you located in step 16:

```
#rn_ChatLaunchFormDiv.rn_ChatForm.rn_Loading {
    background: url(images/loading.gif) no-repeat center center;
    height:auto !important;
    height:66px;
    min-height:66px;
}
```

**24** Add the following code to the bottom of the file.

```
/* *****
14. High Contrast Mode
Microsoft Windows has an accessibility
feature known as High Contrast Mode.
This mode sets all background-images
to `none`, so remediation steps are
needed to add back in icon indicators.
The following rules take advantage of
the :before pseudo element, so IE > IE7
display unicode character equivalents
for icons in backgrounds (e.g. x, >, <).
*****/
.rn_HighContrastMode input,
.rn_HighContrastMode input[type="submit"],
.rn_HighContrastMode button,
.rn_HighContrastMode textarea,
.rn_HighContrastMode select {
    /* In FF, if border is set to anything other than
       the UA default then the border won't show up.
       Strangely, it does become visible if you set
       the border style to inset or outset.*/
    border: 1px outset;
}
/* Dialog close icon */
.rn_HighContrastMode.yui-skin-sam .container-close {
    text-indent: 0;
}
.rn_HighContrastMode.yui-skin-sam .container-close:before {
```

---

```

        content: "\00D7";
        display: block;
        font-size: 24px;
        width: 20px;
        height: 20px;
    }
/* YUI TreeView */
.rn_HighContrastMode .ygtvfocus {
    border: 1px solid #000;
}
.rn_HighContrastMode .ygtvtp:before,
.rn_HighContrastMode .ygtvlp:before,
.rn_HighContrastMode .ygtvlpb:before,
.rn_HighContrastMode .ygtvtph:before,
.rn_HighContrastMode .ygtvloading:before {
    content: "\25BA";
    position: relative;
    top: 6px;
}
.rn_HighContrastMode .ygtvtm:before,
.rn_HighContrastMode .ygtvlm:before,
.rn_HighContrastMode .ygtvlmh:before,
.rn_HighContrastMode .ygtvtmh:before {
    content: "\25BC";
    position: relative;
    top: 6px;
}
}

```

25 Save *site.css*.

## Edit widget CSS

Select a widget to see the procedure for editing the widget's CSS file.

- [AnswerFeedback](#)
- [OpenLogin](#)
- [ProductCategoryInput](#)
- [ProductCategorySearchFilter](#)

### *To change the display of the AnswerFeedback widget*

- 1 Open the *AnswerFeedback.css* widget in `/cp/customer/assets/themes/standard/widgetCss`.
- 2 Search for the occurrences of `AnswerFeedback2` and remove the `2` from the widget name.
- 3 Edit the first line (`.yui-panel.rn_Dialog.rn_FeedbackDialog {`) to the following:
 

```
.rn_AnswerFeedbackDialog {
```
- 4 Locate the following line of code (from which you've already removed the "2" suffix in the widget name):
 

```
.rn_AnswerFeedback .rn_RatingMeter a.rn_RatingCellOver {
```
- 5 Edit the code you located in step 4 to the following:
 

```
.rn_HighContrastMode .rn_AnswerFeedback .rn_RatingMeter
a.rn_RatingCell:before {
    content: "\2606";
}
.rn_AnswerFeedback .rn_RatingMeter a.rn_RatingCellOver {
```
- 6 Locate the following line of code (from which you've already removed the "2" suffix in the widget name):
 

```
.rn_AnswerFeedbackForm .rn_DialogSubtitle {
```
- 7 Edit the code you located in step 6 to the following:
 

```
.rn_HighContrastMode .rn_AnswerFeedback .rn_RatingMeter
a.rn_RatingCellOver:before {
    content: "\2605";
}
.rn_AnswerFeedbackForm .rn_DialogSubtitle {
```
- 8 Save *AnswerFeedback.css*.

### *To change the display of the OpenLogin widget*

- 1 Open the *OpenLogin.css* widget in `/cp/customer/assets/themes/standard/widgetCss`.
  - 2 Locate the following line of code:
 

```
.rn_OpenLogin .rn_LoginProvider{
```
  - 3 Add the following code immediately above the code you located in step 2:
 

```
.rn_HighContrastMode .rn_OpenLogin .rn_Header em {
    display: none;
```
-

```
}

```

- 4 Locate the following line of code:

```
.rn_OpenLogin .rn_LoginProvider:hover,
```

- 5 Add the following code immediately above the code you located in step 4:

```
.rn_HighContrastMode .rn_OpenLogin .rn_LoginProvider {
    color: #000;
    font-size: 1.4em;
    height: 22px;
    padding: 10px 0 4px 30px;
    text-decoration: none;
    width: 100px;
}
```

- 6 Locate the following line of code:

```
.rn_OpenLogin a.rn_Facebook{
```

- 7 Add the following code immediately above the code you located in step 6:

```
.rn_HighContrastMode .rn_OpenLogin .rn_LoginButton.rn_Facebook,
.rn_HighContrastMode .rn_OpenLogin .rn_LoginButton.rn_Twitter,
.rn_HighContrastMode .rn_OpenLogin .rn_LoginButton.rn_Yahoo {
    color: #FFF;
    background: none;
    border: 1px outset;
    font-size: 1em;
}
```

- 8 Locate the following line of code:

```
.rn_OpenLogin.rn_OpenLoginDialog .rn_ActionArea{
```

- 9 Add the following code immediately above the code you located in step 8:

```
.rn_OpenLoginDialog {
    width: 330px;
}
```

- 10 Save *OpenLogin.css*.

### *To change the display of the ProductCategoryInput widget*

- 1 Open the *ProductCategoryInput.css* widget in */cp/customer/assets/themes/standard/widgetCss*.

- 2 Locate the following line of code in the attributes under `.rn_ProductCategoryInput` `button.rn_DisplayButton` {:
 

```
padding:4px 20px 4px 4px;
```
- 3 Add the following code below the code you located in step 2:
 

```
position: relative;
```
- 4 Locate the following line of code:
 

```
.rn_ProductCategoryInput .rn_Panel {
```
- 5 Add the following code immediately above the code you located in step 4:
 

```
.rn_HighContrastMode .rn_ProductCategoryInput button.rn_DisplayButton
{
  border: 1px inset;
}
.rn_HighContrastMode .rn_ProductCategoryInput
button.rn_DisplayButton:after {
  content: "\25BC";
  position: absolute;
  right: 4px;
}
```
- 6 Save *ProductCategoryInput.css*.

### *To change the display of the ProductCategorySearchFilter widget*

- 1 Open the *ProductCategorySearchFilter.css* widget in `/cp/customer/assets/themes/standard/widgetCss`.
  - 2 Locate the following line of code in the attributes under `.rn_ProductCategorySearchFilter` `button.rn_DisplayButton` {, which is the first line:
 

```
padding:4px 20px 4px 4px;
```
  - 3 Add the following code below the code you located in step 2:
 

```
position: relative;
```
  - 4 Locate the following line of code:
 

```
.rn_ProductCategorySearchFilter .rn_Panel {
```
  - 5 Add the following code immediately above the code you located in step 4:
 

```
.rn_HighContrastMode .rn_ProductCategorySearchFilter
```
-

```

button.rn_DisplayButton {
    border: 1px inset;
}
.rn_HighContrastMode .rn_ProductCategorySearchFilter
button.rn_DisplayButton:after {
    content: "\25BC";
    position: absolute;
    right: 4px;
}

```

- 6 Add the following code to the bottom of the file:

```

.rn_ProductCategoryLinks {
    width: 400px;
}

```

- 7 Save *ProductCategorySearchFilter.css*.

## Disabling incident receipt emails

The migration process automatically adds a rule as the first rule in the initial state to send an **email receipt** when a customer submits a question on the Ask a Question page. (If an initial **rule state** does not exist, the migration process will automatically create it.) If you do not want to send your customers an email when they submit an incident, the following procedure shows you how to disable the rule.

### *To disable automatic incident receipts*

- 1 On the **CX Console**, click the Configuration button.
- 2 Double-click Rules under Site Configuration. Incident rules open by default.
- 3 Click Edit on the ribbon.
- 4 Expand the Initial folder under States in the left column.
- 5 Right-click Send Receipt Email and select Disable.
- 6 Activate the **rule base**.

## Changing incident source for rules

If you have already been using the **Connect PHP API** before migrating to Framework Version 3, you may need to edit your **business rules** for incidents on the **CX Console**.

Review your incident rules to see if any of them have an IF **condition** of Incident.Source equals End-User Pages > Ask a Question. If they do, edit the rule to also add the condition IF Incident.Source equals End-User Pages > End-User Connect.

## Removing customer search preferences

The **Connect PHP API** does not store contact preferences in the database, so the use of *profile\_default* attributes is no longer supported for Framework Version 3. Check to see if you are using any of the following business objects as input widgets on your customer portal. If so, they will be meaningless in the new framework, so remove them from your code.

- List items per page (*contacts.lines\_per\_page*)
- Default product (*contacts.prod*)
- Default category (*contacts.cat*)
- Default search (*contacts.search\_type*)
- Default search text (*contacts.search\_text*)

## Changing the SmartAssistantDialog widget

Several default values for the SmartAssistantDialog widget have been changed in Framework Version 3 to simplify the code. In Framework Version 2, the following code defined the SmartAssistantDialog widget on the *ask.php* page in the reference implementation.

```
<rn:widget path="input/SmartAssistantDialog"
label_cancel_button="#rn:msg:EDIT_QUESTION_CMD#"
label_solved_button="#rn:msg:MY_QUESTION_IS_ANSWERED_MSG#"
display_answers_inline="true"
label_accesskey="<span
class='rn_ScreenReaderOnly'>#rn:msg:PREFER_KEYBOARD_PCT_S_PLUS_1_PCT_
D_LBL#</span>"
label_prompt="#rn:msg:FOLLOWING_ANS_HELP_IMMEDIATELY_MSG#"
display_button_as_link="label_cancel_button"
dialog_width="800px"/>
```

In Framework Version 3, the widget uses the default values for its attributes. As a result, the widget code on the *ask.php* page is simply:

```
<rn:widget path="input/SmartAssistantDialog"/>
```

---



You may want to clean up your code by removing the SmartAssistantDialog widget's migrated attributes.

**Note** In Framework Version 2, the default value for the SmartAssistantDialog widget's *label\_solved\_button* attribute was null, so the button did not appear. In Framework Version 3, the attribute has a default value (which is "My Question is Answered") so it appears as a link.

## Using the organization password

In Framework Version 2, you could let contacts associate themselves with an **organization** by adding the Organization login field to the page. An organization password was not necessary to create the association. In Framework Version 3, if you want the organization login entry to associate the contact with an organization, you must also add the organization password field to the page (typically the Create an Account page) if the organization has a password. Customers must enter both the organization login and organization password to associate themselves with an organization.

### *To add the organization password to the Create an Account page*

- 1 Open the *create\_account.php* file in the *development/views/pages/utills* folder.
- 2 Locate the line of code that you used to add the organization login. Your code may resemble the following.

```
<rn:widget path="input/FormInput"
name="contacts.organization_login"
label_input="Organization Login" />
```

- 3 Add the following line below the line you located in step 2.

```
<rn:widget path="input/FormInput"
name="contacts.organization_password"
label_input="Organization Password" />
```

- 4 To use the Connect PHP API naming conventions, edit both lines of code using the following example.

```
<rn:widget path="input/FormInput"
name="Contact.Organization.Login"
label_input="Organization Login" />
```

```
<rn:widget path="input/FormInput"
name="Contact.Organization.NewPassword"
label_input="Organization Password" />
```

5 Save *create\_account.php*.

## Editing the CommunitySearchResults widget

If you used the CommunitySearchResults widget on your customer portal in Framework Version 2, check to see if you have used the *display\_initial\_posts* attribute. This attribute does not appear in the Framework Version 3 widget because community posts are displayed by default when the widget loads on a page. You can remove the attribute from your widget code.

## Adding the Japanese name suffix

Framework Version 2 offered a *jp\_suffix* business object that allowed you to attach “-san” to Japanese names. To accomplish this in Framework Version 3, add the NAME\_SUFFIX\_LBL message base to the field name display, as shown in the following example.

```
<rn:field name="Contact.LookupName"/><rn:condition language_in="ja-JP">#rn:msg:NAME_SUFFIX_LBL#</rn:condition>
```

**Note** This example displays the suffix only when the site’s language is Japanese.

## Login required configuration setting

The August 2010 release introduced a new **configuration setting**, CP\_CONTACT\_LOGIN\_REQUIRED, which lets you require your customers to log in to most customer portal pages. Even if you have set CP\_CONTACT\_LOGIN\_REQUIRED to Yes, the reference implementation does not require login on the following pages, where requiring customers to be logged in doesn’t make sense.

- Reset Your Password—*account/reset\_password.php*
  - Finish Account Creation—*account/setup\_password.php*
  - Mobile reset password—*mobile/account/reset\_password.php*
  - Mobile finish account creation—*mobile/account/setup\_password.php*
  - Error page—*error.php*
  - Mobile error page—*mobile/error.php*
  - Error 404 page—*error404.php*
  - Mobile error page—*mobile/error404.php*
  - Account Assistance—*utils/account\_assistance.php*
  - Mobile account assistance—*mobile/utils/account\_assistance.php*
  - Create an Account—*utils/create\_account.php*
-

- Mobile create account—*mobile/ utils/ create\_account.php*
- Log In—*utils/ login\_form*
- Mobile login—*mobile/ utils/ login\_form*

If you are migrating from a pre-August 2010 version of the Customer Portal (November 2009, February 2010, or May 2010) and if you set CP\_CONTACT\_LOGIN\_REQUIRED to Yes, you must edit each of the pages listed above to add `login_required="false"` to the page's meta tag, which is the first line of each of the files. For example, line 1 of the Create an Account page should be:

```
<rn:meta title="#rn:msg:CREATE_NEW_ACCT_HDG#"
template="standard.php"
login_required="false"
redirect_if_logged_in="account/overview"
force_https="true" />
```

## Using the KnowledgeSyndication widget overlay feature

When you place the KnowledgeSyndication widget on a page that is external to your customer portal, your customers can access the **knowledge base** from that page instead of being required to go to your support site. Its default behavior is to open a new page, but there may be times when you don't want customers leaving the page they are on, for example, if they are in the middle of a transaction. A new attribute for the KnowledgeSyndication widget opens the widget as an overlay on the current page.

*To open answers in an overlay instead of a separate window*

- 1 In a browser, type `https://<your_site>/ci/tags/syndicated_widgets/standard/KnowledgeSyndication` to open the page for the KnowledgeSyndication widget.
- 2 Locate the attribute called *display\_answers\_in\_overlay* and select the check box.
- 3 On the right side of the widget page, click the top Select Text button to select the code in the first yellow box and press **Ctrl+c** to copy the code. For more information about this step and the following ones, refer to [To add the KnowledgeSyndication widget to a web page](#).
- 4 Open the source code for the web page you want to add the widget to and locate the current code for the KnowledgeSyndication widget.
- 5 Replace the current code with the code you copied from the widget page.

Next, you'll need to edit the widget's CSS file in order for the overlay to work properly.

*To edit the CSS file for the KnowledgeSyndication widget*

- 1 Open the *KnowledgeSyndication.css* file in */cp/customer/assets/css/syndicated\_widgets/standard*.
- 2 Select all the code in the file and delete it.
- 3 Copy the following code.

```
div.rn_Hide{
    display:none;
}
div.rn_Show{
    display:block;
}

div.rn_SearchArea{
    padding-bottom: 5px;
}

div.rn_SearchBox,
div.rn_SearchButton{
    display:inline;
}

div.rn_Corrections{
    padding-bottom: 5px;
}

div.rn_Documents .rn_List li.rn_Item span.rn_Title,
div.rn_Documents .rn_List li.rn_Item span.rn_Description,
div.rn_Corrections .rn_WordCorrection,
div.rn_Content .rn_List li.rn_Item span.rn_Title,
div.rn_Content .rn_List li.rn_Item span.rn_Description,
div.rn_Documents .rn_List li.rn_Item span.rn_Title a,
div.rn_Content .rn_List li.rn_Item span.rn_Title a{
    font-size:1em;
    font-weight:normal;
}

div.rn_Suggestions,
div.rn_Documents .rn_List,
div.rn_Content .rn_List,
```

---

```
div.rn_Corrections,  
div.rn_Navigation a,  
div.rn_Documents h3{  
    font-size:.75em;  
    font-weight:normal;  
}  
  
div.rn_Documents{  
    margin-top: 2px;  
    width:100%;  
    border:1px solid black;  
    background-color:whitesmoke;  
    padding-bottom: 1px;  
    padding-left: 2px;  
}  
  
div.rn_Documents h3{  
    font-weight:bold;  
}  
  
div.rn_Documents .rn_List li.rn_Item span.rn_Title a,  
div.rn_Content .rn_List li.rn_Item span.rn_Title a{  
    text-decoration:underline;  
}  
  
div.rn_Content{  
    padding-bottom: 5px;  
}  
  
div.rn_Navigation{  
    position: relative;  
}  
  
div.rn_Navigation a{  
    text-decoration:underline;  
}  
  
.rn_screen_reader_only{  
    position:absolute;  
    left:-10000px;
```

```
        top:auto;
        width:1px;
        height:1px;
        overflow:hidden;
    }

    div.rn_AnswerOverlay{
        position:fixed;
        top:75px;
        left:50%;
        width:70%;
        margin-left:-35%;
        height:70%;
        width:70%;
        overflow:auto;
        background-color:#FFF;
        -moz-box-shadow: 0 0 5px 5px #333;
        -webkit-box-shadow: 0 0 5px 5px #333;
        box-shadow: 0 0 5px 5px #333;
        border: 1px solid black;
        color:black;
    }

    div.rn_AnswerContent{
        padding:10px;
    }

    a.rn_CloseOverlay{
        position:absolute;
        top:0;
        right:0;
        height:30px;
        width:30px;
        text-indent:100%;
        white-space:nowrap;
        overflow:hidden;
        background: url(data:image/
png;base64,iVBORw0KGgoAAAANSUhEUgAAABUAAAVCAyAAACpF6WWAAAAAXNSR0
IArs4c6QAAAArnQU1BAACxjwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAAAadEV
YdFNvZnR3YXJlAFBhaW50Lk5FVCB2My41LjEwMPRyoQAAA05JREFUOE+1lEEKAjEM
RQuuBEEQxL0rcTUbL+UFhFl6BxE3CoJ4TfPA1DR0dKa0gcAk+XkNbach1NtMWufim
```

---

```

ymITsQn8bv42zgxeeoY8L+2EsXZgSzUfqND/xO8FcFtJFDh6OnL2rIAaMH0M/
HC0ns34VPi68DU5KnbraA/2eNdBniQ3Fr84mrE5Kl7MJxoR9f4kHj/
rVqwAilRR2enhRPNT4PQg8mxwBCQOpxoL7eirm7BKs5NqHo40YbuoeaLtE0mbbKnT
U6/yT3lIGr9UfZQQ4l/
PwFqUP2VUnDJe5qdMJfsJDnm5R8NnCPMXv4PF4OjA8GBa2cAAAAASUVORK5CYII=)
no-repeat center center;
}
div.rn_AnswerSummary{
    font-size:2em;
    line-height:2em;
}
div.rn_AnswerInfo{
    color:#666;
    margin:6px 0 20px;
}
div.rn_AnswerDescription{
    border-bottom:1px solid #E2E2E0;
    margin-bottom:5px;
    padding-bottom:5px;
}
div.rn_AnswerAttachments{
    border-top:1px solid #E2E2E0;
    padding-top:15px;
}
div.rn_AnswerAttachments span{
    font-weight:bold;
    font-size:125%;
}

```

- 4 Paste the copied code into the file.
- 5 Save *KnowledgeSyndication.css*.

## Redirecting WAP pages to the basic page set

Once you have implemented the basic page set (refer to [Enabling the basic page set](#)), you'll need to edit the *mapping.php* file so that **WAP** page requests get redirected to the corresponding basic page.

- 1 Disable WAP.
  - a Log in to the CX Console.

- b** Click Configuration > Site Configuration > Configuration Settings.
- c** Search for WAP\_INTERFACE\_ENABLED and set the value to 0.

- 2 Open the *mapping.php* file in the */cp/customer/development/config* folder.
- 3 Add the following code to the bottom of the file.

```

$globalMappingWap['p_faqid'] = 'a_id';
$globalMappingWap['p_iid'] = 'i_id';
$globalMappingWap['p_key'] = array(
    // the 'p' and 'c' logic works for the answer report used in the
    standard implementation of WAP
    'p' => array('p_list', '0'),
    'c' => array('p_list', '1'),
    // the 'kw' logic should work for any report
    'kw' => array('p_type', '3'),
);

$pageMappingWap['/' ] = array('new_page' => 'home');
$pageMappingWap['login.php'] = array('new_page' => 'utils/
login_form');
$pageMappingWap['listans.php'] = array('new_page' => 'answers/list');
$pageMappingWap['search.php'] = array('new_page' => 'answers/list');
$pageMappingWap['ans.php'] = array('new_page' => 'answers/detail');
$pageMappingWap['myq.php'] = array('new_page' => 'account/questions/
list');
$pageMappingWap['inc.php'] = array('new_page' => 'account/questions/
detail');
$pageMappingWap['myq_upd.php'] = array('new_page' => 'account/
questions/detail');
$pageMappingWap['myprofile.php'] = array('new_page' => 'account/
profile');
$pageMappingWap['ask.php'] = array('new_page' => 'ask');
$pageMappingWap['acct_new.php'] = array('new_page' => 'utils/
create_account');
$pageMappingWap['acct_assistance.php'] = array('new_page' => 'utils/
account_assistance');
$pageMappingWap['passwd_change.php'] = array('new_page' => 'account/
change_password');
$pageMappingWap['passwd_reset.php'] = array('new_page' => 'account/
reset_password');
$pageMappingWap['passwd_setup.php'] = array('new_page' => 'account/
setup_password');

```

---



#### 4 Save *mapping.php*.

Now when customers try to access one of the old WAP pages, they are redirected to the equivalent page in the basic page set.

## Deprecation of Intent Guide

Customer Portal Framework Version 3.1 no longer supports the **Intent Guide** so you'll need to remove references to it. You can use the Code Assistant to remove the two Intent Guide pages and then you can remove other references manually.

**Note** The IntentGuideDisplay widget folder and all the files it contains do not appear in the Framework Version 3.1 file structure after you migrate.

### *To use Code Assistant to remove Intent Guide pages*

- 1 On the Customer Portal Administration site, hover over Tools and select Code Assistant.
- 2 On the Select Your Versions screen, verify that the production mode is Pre-3.0 and the development mode is version 3.1. If necessary, click the drop-down menu and make the correct selections.
- 3 Click Next Step.
- 4 On the Select an Operation from the List screen, select the Intent Guide Cleanup radio button and click Continue.
- 5 Select the Select All pages to have the Code Assistant remove both Intent Guide pages, and click Confirm Changes.
- 6 Click OK on the confirmation message. A Finish Up section confirms that the pages have been removed.

### *To remove Intent Guide references manually*

- 1 If the `/cp/core/framework/Models/Intentguide.php` file appears in the customer portal file structure, delete it.
- 2 Search your files for the CombinedSearchResults widget and remove the following attributes if they appear in your code.

- *label\_intent\_guide\_category*
- *label\_intent\_guide\_link*
- *label\_intent\_guide\_results\_beading*
- *label\_more\_intent\_guide\_results*
- *label\_single\_intent\_guide\_result*
- *intent\_guide\_category\_id*
- *intent\_guide\_referrer*
- *intent\_guide\_results*
- *intent\_guide\_siteref*
- *maximum\_intent\_guide\_results*

## Deprecation of SearchTruncation widget

Customer Portal Framework Version 3.1 no longer supports the SearchTruncation widget. If you are using this widget on your customer portal, you will need to remove it.

## Adding label attributes

Because of the data structure of the **Connect PHP API**, some business objects use more generic labels than those used in Framework Version 2. For example, each of the following business objects is labeled “Number” by default, although each had a unique label in Framework Version 2.

- Contact.Phones.ASST.Number
- Contact.Phones.FAX.Number
- Contact.Phones.HOME.Number
- Contact.Phones.MOBILE.Number
- Contact.Phones.OFFICE.Number

To create a label that is more specific than “Number,” you must add the *label\_input* attribute to each widget. If you want to create input fields for every phone number, you might add the following code.

```
<rn:widget path="input/FormInput"
name="Contact . Phones . ASST . Number"
label_input="Assistant Phone" />
<rn:widget path="input/FormInput"
name="Contact . Phones . FAX . Number"
label_input="Fax Phone" />
```

---

```

<rn:widget path="input/FormInput"
name="Contact.Phones.HOME.Number"
label_input="Home Phone" />
<rn:widget path="input/FormInput"
name="Contact.Phones.MOBILE.Number"
label_input="Mobile Phone" />
<rn:widget path="input/FormInput"
name="Contact.Phones.OFFICE.Number"
label_input="Office Phone" />

```

The following table contains a list of business objects that have potentially confusing or ambiguous labels. You may decide to add the *label\_input* attribute to the input widget using the suggested value or another one of your choosing. Or, if a **message base** exists, you may want to reference that.

**Tip** If you use message bases and decide to change the label in the future, you can accomplish all of the edits that use that message base by simply changing the message base once instead of changing individual labels throughout your customer portal.

The procedure is the same for display widgets except that you will use the *label* attribute instead of *label\_input*. You can also use the `<rn:field>` tag to output the value of a business object.

Table 4: Updating Business Object Labels

Business Object	Default Label	Add label_input=" [value]" to widget	Or add label_input=" #rn:msg[value]#"
Contact.Address.PostalCode	PostalCode	Postal Code	POSTAL_CODE_LBL
Contact.Address.StateOrProvince	StateOrProvince	State or Province	STATE_LBL or PROVINCE_LBL
Contact.ChannelUsernames.TWITTER.Username	Username	Twitter Username	—
Contact.ChannelUsernames.YOUTUBE.Username	Username	YouTube Username	—
Contact.ChannelUsernames.FACEBOOK.Username	Username	Facebook Username	—

Table 4: Updating Business Object Labels (Continued)

Business Object	Default Label	Add label_input=" [value]" to widget	Or add label_input="#rn:msg[value]#"
Contact.Emails.PRIMARY.Address	Address	Email Address	EMAIL_ADDR_LBL
Contact.Emails.ALT1.Address	Address	Alternate Email 1	ALTERNATE_EMAIL_ADDRESS_1_LBL
Contact.Emails.ALT2.Address	Address	Alternate Email 2	ALTERNATE_EMAIL_ADDRESS_2_LBL
Contact.Organization.Name	Login	Organization Name	ORGANIZATION_NAME_LABEL
Contact.Phones.ASST.Number	Number	Assistant Phone	ASST_PHONE_LBL
Contact.Phones.FAX.Number	Number	Fax Phone	FAX_PHONE_LBL
Contact.Phones.HOME.Number	Number	Home Phone	HOME_PHONE_LBL
Contact.Phones.MOBILE.Number	Number	Mobile Phone	MOBILE_PHONE_LBL
Contact.Phones.OFFICE.Number	Number	Office Phone	OFFICE_PHONE_LBL
Incident.SLAInstance	AssignedSLA Instance	Service Level Agreement	—

## Customer portal changes on the agent desktop

Included in the `/cp/development/views` folder are a few files that impact the display of answers on the **agent desktop**. The `agent.php` file in the `views/templates` folder controls how guides appear when they are previewed on the Oracle RightNow CX administration interface (also known as the agent desktop). The `views/admin` folder contains files for previewing an answer on the agent desktop.

## Guided assistance styling on the agent desktop

Changes have been added to improve the display of **guides** on the CX Console when agents select **guided assistance** to help them resolve a customer issue. These changes make the desktop styling more consistent with what customers experience on your customer portal. You can style the guided assistance template for agents however you choose, but the following procedure offers some basic CSS styling to reset the font family settings.

### *To set fonts for agent guided assistance*

- 1 Open the *agent.php* file in */cp/customer/development/views/templates*.
- 2 Locate the following line of code.

```
<title><rn:page_title/></title>
```

- 3 Add the following block of code below the line you located in step 2 and above the closing `</head>` line.

```
<style type='text/css'>
  /* embedding the CSS rules directly into the template rather than
  creating a theme because, without an explicit theme, widget CSS is
  automatically loaded from the standard theme */
  input,textarea,select{font-family:inherit;}
  html {
    font-family:Helvetica, Arial, sans-serif;
  }
  html[lang="ja-JP"],
  html[lang="ja-JP"] input, html[lang="ja-JP"] textarea,
  html[lang="ja-JP"] select {
    font-family:"Hiragino Kaku Gothic Pro", "?????? Pro W3",
    Meiryo, "?????", "MS PGothic", "MS P????", Helvetica, Arial,
    sans-serif;
  }
  body {
    font-size:.75em;
    line-height:1.250em;
    text-align:left;
  }
  h2, h3, h4, h5, h6 {
    font-family:Arial, sans-serif;
  }
</style>
```

```
</style>
```

- 4 Save *agent.php*.

## Replacing variables on the preview page

The Customer Portal file structure contains answer preview pages that are displayed on the CX Console so that staff members working with answers can see what the answer will look like on your customer portal. In Framework Version 3, these files are located in */cp/customer/development/views/admin*.

The *answer\_full\_preview.php* page lets you display additional answer fields by using PHP answer **variables**. If you customized your version 2 answer preview page with these variables, you'll need to edit the file to use a new naming convention that results from all the answer data being contained in a single `$answer` variable.

As an example of how those variables changed, the following table shows the version 3 equivalent of the version 2 name.

Table 5: Answer Variables on the Answer Preview Page of the Agent Desktop

Framework Version 2	Framework Version 3
<pre>&lt;h1 id="rn_Summary"&gt;&lt;?=\$summary;?&gt;&lt;/h1&gt;</pre>	<pre>&lt;h1 id="rn_Summary"&gt;&lt;?=\$answer-&gt;Summary;?&gt;&lt;/h1&gt;</pre>
<pre>\$fileAttachments</pre>	<pre>\$answer-&gt;FileAttachments</pre>

### *To migrate answer variables on the answer preview page*

- 1 Open the *answer\_preview\_full.php* file in the */cp/customer/development/admin* folder.
- 2 Locate all answer variables in the code and modify them by adding `$answer->` before the variable and using an uppercase initial character. For example, rename `$fileAttachments` to `$answer->FileAttachments`.
- 3 Save *answer\_preview\_full.php*.

## Converting custom widgets

There are a number of changes that you must make to your custom widgets when migrating from Framework Version 2 to Framework Version 3. You may also want to make additional changes to take advantage of new Framework Version 3 features. The following topics are covered in this section.

- [Planning your widget conversion](#)
- [Overview of the conversion process](#)
- [Completing the widget information file](#)
- [Converting the controller](#)
- [Converting the view](#)
- [Converting the logic file](#)
- [Updating references to assets](#)

We can't anticipate every widget customization you've made on your customer portal, so we'll guide you to general sections, which contain links to more detailed information, instead of walking you through specific steps as earlier sections of this Migration Guide did.

### Planning your widget conversion

Framework Version 3 introduced the following changes to widgets.

- The [info.yml file](#), which is required for all widgets and defines its details, including attributes.
- The ability for widgets to handle their own [AJAX requests](#).
- The [<rn:block> element](#) in views, which you can use to add customization hook points.
- [JavaScript templates](#), which allow you to create EJS (Embedded JavaScript) template files used by the widget to render dynamic views.
- The Code Assistant that helps you migrate custom widgets by updating the folder structure for widgets, removing the unused `rn:meta` tag, and generating a new `info.yml` file.

Before you start migrating your custom widgets to Framework Version 3, you need to understand the actions and decisions you must make before migrating. The following sections describe the changes to custom widgets.

## The info.yml file

In Framework Version 2, widget controllers contained attribute declarations and URL parameter information. In Framework Version 3, the attributes and URL parameters are specified in the *info.yml* file, which also contains information about its version, dependencies, and other widgets that it extends.

When you use the widget builder to create a widget, an *info.yml* file is created automatically and includes most of the required information. You can edit the file to add the remaining information. The attributes specified in the *info.yml* file can be accessed anywhere in the *controller.php* and *view.php* files as:

```
$this->data['attrs']['attr_name']
```

or in the *logic.js* file as:

```
this.data.attrs.attr_name
```

In each case, *attr\_name* is the name of the attribute as specified in the *info.yml* file.

For step-by-step instructions for creating the YAML file, refer to [Completing the widget information file](#). For general information about YAML, refer to the [YAML website](#).



Before you start migrating a custom widget:

- Open the widget's Framework Version 2 *controller.php* file so you know which attributes and URL parameters to specify in the *info.yml* file.
- Determine if the widget extends from another widget and, if so, which one.
- Determine whether your custom widget contains other widgets within its view.

## Making AJAX requests from widgets

In Customer Portal versions prior to Framework Version 3, widgets requested data by firing named events. Handler functions in *RightNow.Event.js* subscribed to these events, collected the data, sent the AJAX request for data to the server, and fired the correct response events when data came back from the server.

However, in Framework Version 3, widgets can make their own requests to the server, as described in the following sequence.

- You specify the location of a handler for an AJAX request (the endpoint URL) through a widget attribute.



- Before a widget makes a request, the *logic.js* file fires an event, and subscribers to the event can customize the data to be sent in the request. Refer to [Firing an event before submitting the request](#).
- The widget's *logic.js* file makes the AJAX request and calls a handler method in the widget's controller, passing the necessary data for the request. Refer to [Making AJAX requests in the widget JavaScript](#).
- The handler method in the controller can access any parameters passed in the request and calls model functions as required to return the output and results. Refer to [Handling AJAX requests in the widget controller](#).
- The widget's *logic.js* file has a callback method for the response from the server. After a widget receives a response, the *logic.js* file fires an event, and subscribers to the event can customize behavior, for example, to modify any data in the response. Refer to [Firing an event after getting a response](#).



Before you start migrating:

- Look in the Framework Version 2 *logic.js* file to see if there are any custom AJAX requests.
- Determine whether your custom widget is the only widget making requests to the AJAX endpoint. If so, the AJAX endpoint should be defined in the widget attributes. If there are several widgets making requests, the AJAX endpoints should be defined in a custom controller (which is how all AJAX endpoints were defined in Framework Version 2).

## Using blocks in views

Framework Version 3 includes the `<rn:block>` element in widget *view.php* files using the following syntax:

```
<rn:block id="postFeedbackInput" />
```

Every `<rn:block>` element has an `id` attribute with a value that describes its context, for example, `postFeedbackInput`. Block elements act as customization hook points in widgets and are present as empty placeholder elements in all standard widgets. Empty placeholder blocks are self-closing, that is, they open and close the block in a single line of syntax, as demonstrated in the line above. However, if custom widgets have blocks that do contain content, the format looks like the following, which includes a separate closing line.

```
<rn:block id="postFeedbackInput">
  content
</rn:block>
```

The content you add to a self-closing block gets added to the widget. For example, assume the parent widget's *view.php* file contains this code:

```
<rn:block id="main"/>
<div><?=$this->data['js']['moreresults'] ?></div>
```

Also assume that the custom widget that extends the parent widget includes the following:

```
<rn:block id="main">
    <span><?=$this->data['js']['results'] ?></span>
</rn:block>
```

The resulting view file contains this code:

```
<span><?=$this->data['js']['results'] ?></span>
<div><?=$this->data['js']['moreresults'] ?></div>
```

**Important** Extending the *view.php* file of a widget that contains a self-closing block means that the content of the widget you extend gets added to the parent widget's view file to create the new view file. However, if the parent widget already contains content in the `rn:block`, that is, it is closed with a separate `</rn:block>` tag, the content you add to the block for the new widget **replaces** the original content.

Blocks are placed in view files:

- Immediately after the container `<div>` or `<span>` start tags.
- Immediately before the container `<div>` or `<span>` end tags.
- Inside loops.
- Before and after loops.

The following block is used after the opening container tag:

```
<rn:block id="top"/>
```

The following block is used before the closing container tag:

```
<rn:block id="bottom"/>
```

For other blocks, the naming convention is as follows:

```
"preName" / "postName"
```

where *Name* describes the tag being customized by the block.

The following shows how blocks are added to an iterative block of code.

```
<rn:block id="preList"/>
<ul>
    <rn:block id="preListItem"/>
```

```

    <li>
    <rn:block id="topListItem"/>
    ...
    <rn:block id="bottomListItem"/>
    </li>
    <rn:block id="postListItem"/>
    ...
</ul>
<rn:block id="postList"/>

```



Before you start migrating:

- Decide whether you want to add any `<rn:block>` elements that are in parent widgets to your custom widgets.
- Decide whether you want to add any new `<rn:block>` elements to your custom widgets to use as customization points for any widgets that extend from your custom widget. Remember that you can modify a custom widget's view directly, which is not the case with standard widgets.

## Using JavaScript templates

In Framework Version 3, you can use embedded JavaScript (EJS) in template files to dynamically render views. Refer to [EJS JavaScript Templates](#).

The following is an example of code from a *view.ejs* file.

```

<rn:block id="preList"/>
<ul id="<%= listID %>">
  <% for (var i = 0; i < links.length; i++) { %>
    <rn:block id="listItem">
      <li><a href="<%= links[i].href %>"/><%= links[i].label %></a></li>
    </rn:block>
  <% } %>
</ul>
<rn:block id="postList"/>

```

The highlighted code shows the JavaScript that is dynamically rendered when the widget logic file loads the *view.ejs* file and the view is rendered.

JavaScript views allow separation of code and HTML for cleaner dynamic widgets, which can be particularly important when the widget contains a large amount of HTML.

EJS is well suited for AJAX handlers in widgets. For example, if you have a widget that uses AJAX to fetch data from the server, it can use EJS to dynamically update the page with the returned data.

You can also leverage EJS in other ways. In Framework Version 2, both standard and custom widgets had to modify the document object model (DOM) directly. For example:

```
var element = document.createElement("<h1>");
element.innerHTML = response.text;
document.getElementById('myTitle').appendChild(element);
```

This method still works. However, in Framework Version 3, you can add the following in a *view.ejs* file (or assign it to a variable in JavaScript):

```
<h1><%= text %></h1>,
```

You then use this code to render the view:

```
// using inline template or JS variable
this.Y.one('#myTitle') .set('innerHTML', new EJS({text:
  RightNow.Widget[this.data.info.class_name].templates.view}).
  render(response));
```

You can also use `<rn:block>` elements in *view.ejs* files. See [Using blocks in views](#).

Before you start migrating, consider whether you want to use JavaScript templates.

## Overview of the conversion process

Before you begin the following procedure, it may be helpful to have a general understanding of the process.

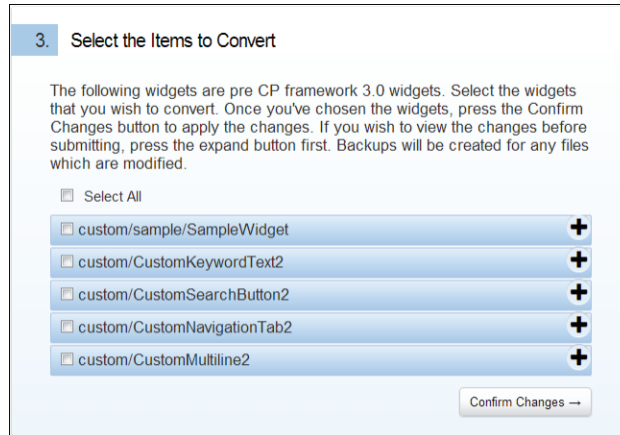
- First, you'll copy all of your custom widgets from your Framework Version 2 site to a location where you can work with them. Later in the procedure, you'll be using this code as a reference for your customizations.
- Next, you'll migrate to Framework Version 3.1, selecting the option that copies all your custom files.
- Then you'll delete all the custom widget folders that were just copied during migration because you're going to rebuild the custom widgets. Even though you're deleting the migrated custom widgets, it was necessary to select Yes to the question about migrating your custom files so that the rest of your customized Version 2 files, including templates, pages, CSS, and assets, are available to you in Framework Version 3.1.
- Now you'll rebuild each of your custom widgets.
  - ▷ You'll create a basic file structure and skeleton files for your custom widget using the widget builder on the Customer Portal Administration site.

- ▷ Then you'll refer to your Version 2 widget code (which you copied to a local site in the first step) and edit the new Framework Version 3.1 widget code generated by the widget builder to include the Version 2 logic.

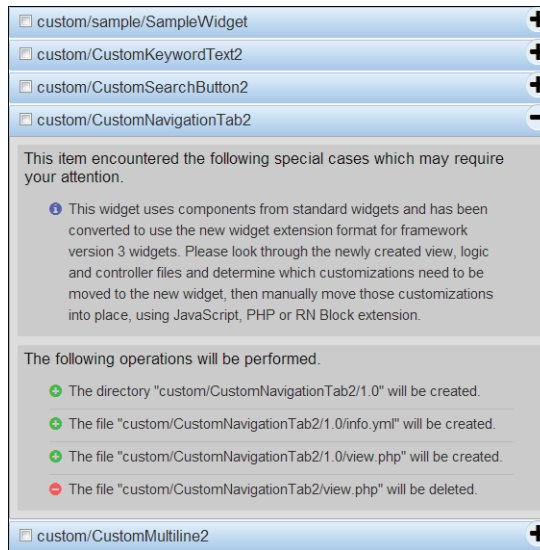
**Important** Do not copy your Version 2 code and replace the Version 3 skeleton code with it because new widget requirements (such as namespaces) are provided in the code generated by the widget builder. Those requirements will be lost if you overwrite the file.

### *To convert custom widgets*

- 1 Download all Framework Version 2 custom widgets to a local folder on your computer using WebDAV.
- 2 Log in to the Customer Portal Administration site and migrate from Framework Version 2 to Framework Version 3.1. Select Yes to copy all custom files. For the complete procedure, refer to [Enabling Framework Version 3.1](#). The custom widgets are copied to the *customer/development/widgets/custom* directory with no changes to the files or file structure.
- 3 Using a WebDAV connection, go to */cp/customer/development/widgets/custom* and delete the custom widget folders that were migrated. (You don't need these files because you're going to use the widget builder to create the Framework Version 3.1 file structure for each widget.)
- 4 On the Customer Portal Administration site, use the Code Assistant to create a file structure and *info.yml* file for the custom widgets you want to convert.
  - a Hover over Tools and select Code Assistant.
  - b On the Select Your Versions screen, verify that the production mode is Pre-3.0 and the development mode is version 3.1. If necessary, click the drop-down menu and make the correct selections.
  - c Click Next Step.
  - d On the Select an Operation from the List screen, select the Widget Migration radio button and click Continue. The Select the Items to Convert screen opens, displaying the custom widgets on your migrated customer portal.



- e To see the changes the Code Assistant will apply to a widget, click the + sign for the widget to expand the section. A list of additions, deletions, and suggestions appear.



- f Click Select All or select the check boxes for the specific widgets you want Code Assistant to convert and click Confirm Changes.
- g Click OK on the confirmation message. The Finish Up screen describes the conversions.

**4. Finish Up**

The following widgets have been converted. The controller.php and logic.js content from the original widget has been inserted into a new Customer Portal Framework Version 3 widget template. Please manually move your content into the appropriate locations in those files. For the logic.js content, there is a YUI 2 to YUI 3 conversion tool, which may be helpful when migrating the JavaScript content.

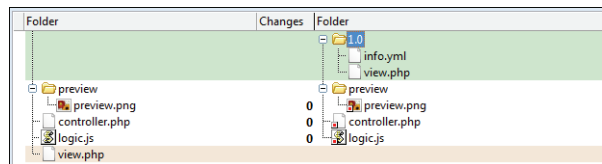
custom/CustomKeywordText2 +

custom/CustomSearchButton2 +

The widget(s) have been successfully migrated, but could not be activated. Please review the migrated widget code then visit the Version Management interface to activate the widgets.

[Home](#) Start Over →

The following screen capture shows the differences in file structure of one of the converted widgets. You can see that in this case no file content was changed, so you'll still need to modify the file as described in the remaining steps of this procedure.



- 5 Edit the *info.yml* file as described in [Completing the widget information file](#).
- 6 If the widget has a controller, edit the generated *controller.php* file. Following the guidelines in [Converting the controller](#), incorporate the controller functionality from the Version 2 custom widget into the new skeleton Version 3.1 *controller.php* file.
- 7 If the widget has a view, edit the generated *view.php* file. Following the guidelines in [Converting the view](#), incorporate the functionality from the Version 2 custom widget into the new skeleton Version 3.1 *view.php* file.
- 8 If the widget has a logic file, edit the generated *logic.js* file. Following the guidelines in [Converting the logic file](#), incorporate the functionality from the Version 2 custom widget into the new skeleton Version 3.1 *logic.js* file.

- 9 If the widget has a *view.ejs* file, make the changes described in [Accessing and rendering EJS views](#).
- 10 Updates references to assets, if required. See [Updating references to assets](#).
- 11 If the widget uses a custom model, update the namespacing. Refer to [Custom models](#).
- 12 Repeat steps 4 through 11 for each custom widget you want to migrate.

## Completing the widget information file

When you use widget builder to create skeleton widget files for Framework Version 3, an *info.yml* file is created. This file contains sections for the version, dependency, and any extension information for the widget. It also contains any attributes that you added using the widget builder. For example:

```
version: "1.0"
requires:
  jsModule:
    - standard
attributes:
  ...
extends:
  widget: standard/feedback/AnswerFeedback
  components:
    - view
    - js
    - php
```

The attributes you define for a widget can be one of the following types.

- **String**—A string value.
- **Boolean**—A true/false option.
- **AJAX endpoint**—A controller endpoint (typically */cc/myCustomController/action*).
- **Option**—A selection of menu options that you define.
- **File path**—Defines an asset relative to the current theme.
- **Integer**—An integer value.

The following procedure describes how to complete the *info.yml* file.

---



*To complete the widget information file*

**Note** To prevent errors, be sure that any changes you make to the *info.yml* file conform to YAML standards. Refer to the [YAML website](#).

- 1 If you have not already done so, add the attributes from your Framework Version 2 *controller.php* file to the *info.yml* file. For example:

```
attributes:
  height:
    name: Height
    type: string
    description: Height of Video
    default: 485
```

- 2 If the widget is to do its own AJAX handling, add attributes in the *info.yml* file that specify the AJAX endpoint. For example:

```
attributes:
  ...
  submit_rating_ajax:
    name: Submit Rating AJAX
    type: ajax
    description: Endpoint to make the answer rating AJAX request
    to
    default: /ci/ajax/widget
  submit_feedback_ajax:
    name: Submit Feedback AJAX
    type: ajax
    description: Endpoint to make the answer feedback AJAX request
    to
    default: /ci/ajax/widget
```

- 3 Add a description in the *info:* section. For example:

```
info:
  description: This widget extends standard AnswerFeedback
  functionality in order to save additional custom fields on the
  created Incident.
```

- 4 Add any URL parameters that are used by the widget to the *info:* section in the *info.yml* file. For example:

```
info:
  description: ...
```

```
urlParameters:
  a_id:
    name: Answer ID
    description: ID of the answer to display
    example: a_id/123
```

**Important** The URL parameter information is no longer inherited from parent widgets as it was for Framework Version 2. Make sure that you copy any URL parameters from parent widgets into the *info.yml* file.

- 5 If required, add a `contains:` section. The `contains:` section is an optional section used when the widget contains one or more other widgets within its view. You must add it manually. For example:

```
contains:
- widget: standard/input/SelectionInput
versions: ["1.0"]
- widget: standard/input/DateInput
versions: ["1.0"]
```

For more information about creating an *info.yml* file, refer to [Widget info files](#).

## Converting the controller

To convert a controller for a custom widget, perform the following edits to the code from the Framework Version 2 *controller.php* file as you incorporate it into the Framework Version 3.1 skeleton file generated by the widget builder.

### *To convert a controller*

- 1 In the constructor for the controller file, remove the attributes declarations, such as the following.

```
$this->attrs['height'] = new Attribute('Height', 'STRING', 'Height of
Video', "485");
```

- 2 Remove the `generateWidgetInformation()` function.

- 3 Locate the following line of code.

```
if (!defined('BASEPATH')) exit('No direct script access allowed');
```

- 4 Replace the code you located in step 3 with the namespace for the widget. For example, for a widget with the path and name of *search/CustomKeywordSearch*, add the following code.

```
namespace Custom\Widgets\search;
```

- 5 Locate the class declaration. For example:

```
class WidgetName extends Widget
```

- 6 Replace the code you located in step 5 with code as follows:

- a If your widget does not extend another widget, enter:

```
class WidgetName extends \RightNow\Libraries\Widget\Base
```

- b If your widget extends a standard widget, enter a line including `extends \RightNow\Widgets\WidgetName`. For example:

```
class WidgetName extends \RightNow\Widgets\KeywordText
```

- c If your widget extends another custom widget, enter a line including `extends \Custom\Widgets\pathname`. For example:

```
class AnotherKeywordText extends
\Custom\Widgets\search\CustomKeywordText
```

- 7 Change the constructor to accept a single `$attrs` parameter as follows:

```
function __construct($attrs) {
    parent::__construct($attrs);
    ...
}
```

- 8 Locate and remove code that loads a model. For example:

```
$this->CI->load->model('standard/Report');
```

- 9 Locate code that calls a model. For example:

```
$this->CI->Report->getDataHTML(...)
```

- 10 Replace the code you located in step 9 with code like the following.

```
$this->CI->model('Report')->getDataHTML(...)
```

- 11 If your Framework Version 2 widget makes AJAX request(s) to `/ci/ajaxRequest` and it is the only widget to do so:

- a Move the functions from `ajaxRequest` into the widget's controller.

- b If the moved functions do not need to access any widget instance variables, make them static.

## Specifying AJAX handlers

If your widget is to handle its own AJAX requests, make sure that you specify the AJAX handlers in the constructor of the Framework Version 3.1 controller. The keys must have the same name as the attributes that specify the AJAX endpoint, for example, `submit_rating_ajax`. The values must be the names of the widget's methods, for example, `submitAnswerRating()`.

```
function __construct($attrs)
{
    parent::__construct($attrs);

    $this->setAjaxHandlers(array(
        'submit_rating_ajax' => 'submitAnswerRating',
        'submit_feedback_ajax' => 'submitAnswerFeedback',
    ));
}
```

You can also specify the clickstream for the widget here, for example:

```
function __construct($attrs) {
    parent::__construct($attrs);

    $this->setAjaxHandlers(array(
        'submit_rating_ajax' => 'submitAnswerRating',
        'submit_feedback_ajax' => array(
            'handler' => 'submitAnswerFeedback',
            'clickstream' => 'submitFeedback',
        )
    ));
}
```

For more information, see [Making AJAX requests from widgets](#).

## Converting the view

To convert a view for a custom widget, perform the following edits to the code from the Framework Version 2 `view.php` file as you incorporate it into the Framework Version 3.1 skeleton file generated by the widget builder.

### *To convert a view*

- 1 Remove the `<rn:meta>` element.

**Note** In Framework Version 2, the meta section in the widget's `view.php` file contained information about the component (controller, view and CSS) of the widget. It is not required in Framework Version 3.

- 2 Remove `tabIndex` calls.
- 3 Add any blocks that are required. For more information, see [Using blocks in views](#).

## Converting the logic file

To convert a logic file for a custom widget, perform the following edits to the code from the Framework Version 2 `logic.js` file as you incorporate it into the Framework Version 3 skeleton file generated by the widget builder.

### *To convert a logic file*

- 1 Locate the first line of code in the `logic.js` file. For example:

```
RightNow.Widget.DisplaySearchFilters = function(data, instanceID) {
    this.data = data;
```

- 2 Replace the code you found in step 1 with code as follows:

- a If your widget does not extend from another widget, it must now extend from `RightNow.Widgets`:

```
RightNow.Widget.DisplaySearchFilters = RightNow.Widgets.extend({
    constructor: function() {
        ...
```

- b If your widget extends a standard widget, enter a line with the name of the standard widget. For example:

```
RightNow.Widget.WidgetName =
RightNow.Widgets.KeywordText.extend({
    constructor: function() {
        ...
```

- c If your widget extends another custom widget, enter a line with the name of that custom widget. For example:

```
RightNow.Widget.AnotherWidgetName =
Custom.Widgets.WidgetName.extend({
    constructor: function() {
```

...

3 Convert all YAHOO calls to their `this.Y` equivalents. Refer to [YUI resources](#).

4 Locate all occurrences of the following code.

```
"rn_" + this.instanceID
```

5 Replace all the code you found in step 4 with:

```
this.baseDomID
```

**Note**

If you are using `Y.node`, replace the code you found in step 4 with `this.baseSelector`. The code `this.baseDomID` includes the `rn_` prefix. Therefore if you are using a different prefix, continue appending `this.instanceID` to the end of that prefix.

6 Locate and remove the following code, because it is now set in the base widget constructor.

```
this.data = data;
this.instanceID = instanceID;
```

7 Simplify EventObject setup by passing data in the constructor.

```
var eventObject = new RightNow.Event.EventObject(this, {data: {
  a_id: this.data.js.answerID,
}});
```

Refer to [JavaScript recipes](#) for examples of how you can make some changes to the `logic.js` file for conversion to Framework Version 3.

## Updating the logic file for AJAX requests

If your widget is to handle its own AJAX requests, the `logic.js` file must make the AJAX request and call a handler method in the widget's controller, passing the necessary data for the request. It must have a callback method for the response from the server. See [Making AJAX requests from widgets](#).

### *To update the logic file for AJAX requests*

1 Wrap AJAX requests with conditional events, passing the eventObject to be sent in the AJAX request. For example:

```
var eventObject = new RightNow.Event.EventObject(this, {data: {
  a_id: this.data.js.answerID,
  rate: this._rate,
```

```

    email: this.data.js.email || this._emailField.value,
    message: this._feedbackField.value,
    threshold: this.data.attrs.dialog_threshold,
    options_count: this.data.attrs.options_count,
    submitfeedback: true,
    w_id: this.data.info.w_id
  });
  if(RightNow.Event.fire("evt_answerFeedbackRequest",
    eventObject)){

    RightNow.Ajax.makeRequest(this.data.attrs.submit_feedback_ajax,
      eventObject.data, {successHandler: this._onResponseReceived,
        scope: this, data: eventObject, json: true});
  }

```

This allows subscribers to the event to cancel the AJAX request or modify its data. In this example, the `this._onResponseReceived()` callback function is called when a response is received. The response will come back JSON decoded.

- 2 Replace the listener functions with a callback function that fires an event immediately after a response is received. For example:

```

_onResponseReceived: function(response, originalEventObj)
{
  if(RightNow.Event.fire("evt_answerFeedbackResponse", {data:
    originalEventObj, response: response})) {
    if(this._incidentCreateFlag){
      this._incidentCreateFlag = false;
      var dialogOptions;
      if(response.error){
        dialogOptions = {icon: "WARN", exitCallback: {fn:
          this._enableDialog, scope: this}};
      }
      else{
        dialogOptions = {exitCallback: {fn:
          this._closeDialog, scope: this}};
      }

      RightNow.UI.Dialog.messageDialog(response.message,
        dialogOptions);
    }
    else{
      this._closeDialog();
    }
  }
}

```

```

    }
  },
},

```

This allows subscribers of the event to modify any of the data in the response.

- 3 For events that are fired either before sending a request, or after receiving a response, add subscribers to the event to customize behavior, if required.

## Accessing and rendering EJS views

If you have decided to use JavaScript templates in your custom widget (refer to [Using JavaScript templates](#)), update the EJS view by including the following code in the *logic.js* file.

```

this._contentDiv.set("innerHTML", new EJS({text:
  RightNow.Widget[this.data.info.class_name].templates.view}).render(new
  wdata));

```

This updates the *view.ejs* file. The part after `templates.` indicates the EJS filename.

Note that `this.data.info.class_name` references the widget name. This is to let custom widgets extend standard widget JavaScript templates without extending and customizing the standard widget JavaScript. Because the EJS file may be in the standard widget folder, you must use `this.data.info.class_name` to make sure that the correct EJS view is used.

## JavaScript recipes

This section discusses various aspects of JavaScript coding that you can update in your custom widgets to take advantage of Framework Version 3.

Each subsection shows the `grep` command expression that you can use on source files to find code to be updated. Examples of Framework Version 2 code and the equivalent Framework Version 3 code are also shown. Select any of the following JavaScript modifications.

- [Using this.Y.one to get node objects](#)
- [Accessing properties on node objects](#)
- [Iterating through sibling elements](#)
- [Chaining function calls](#)
- [Listening to events](#)

When you are finished with the JavaScript edits, continue with the widget conversion process.

## Using this.Y.one to get node objects

The `document.getElementById()` method gets a node object in a document object model (DOM) tree. In other words, it gets the first instance of an element in a Web page with a specified ID value. In Framework Version 3, you can instead use the `this.Y.one()` function.

---



The `this.Y.one()` function expects a CSS-style selector, so the string passed in must have a `#` character at the beginning when using the ID of the element.

Use a `grep` command with the following expression on your Framework Version 2 files to find instances of the `document.getElementById()` method:

```
document.getElementById
```

The `grep` command finds code such as the following:

```
var element = document.getElementById("rn_" + this.instanceID +
  "_Button"),
  element2 = document.getElementById(this.data.attrs.element);
```

The equivalent code for Framework Version 3 is:

```
var element = this.Y.one(this.baseSelector + "_Button"),
  element2 = this.Y.one('#' + this.data.attrs.element);
```

## Accessing properties on node objects

In Framework Version 3, the code now manipulates node objects instead of `HtmlElement` objects. Therefore some properties must now be accessed with functions.

Use a `grep` command with the following expression on your Framework Version 2 files to find code that accesses the `value`, `style`, or `display` properties:

```
(\.value|\.style\.display)
```

The `grep` command finds code such as the following:

```
var thisValue = element.value, id = element.id;
element.value = "this";
element.style.display = 'none';
var thisDisplay = element.style.display;
```

The equivalent code for Framework Version 3 is as follows:

```
var thisValue = element.get('value'), id = element.get('id');
element.set('value', "this");
element.setStyle('display', 'none');
var thisDisplay = element.getStyle('display');
var thisDisplay = element.getComputedStyle('display');
```

## Iterating through sibling elements

In Framework Version 3, it is now much easier to iterate through sibling elements when using node objects, especially to find the next non-text element (that is, skipping any whitespace nodes).

Use a grep command with the following expression on your Framework Version 2 files to find code that iterates through sibling elements.

```
\.nextSibling
```

The grep command finds code such as the following:

```
var current = toggle.nextSibling;
while(current) {
    if(current.nodeType === 1)
        break;
    else
        current = current.nextSibling;
}
```

The equivalent code for Framework Version 3.1 is as follows:

```
var current = toggle.next();
```

## Chaining function calls

In Version 3, you can use chaining to make multiple changes on a single element.

Use a grep command with an expression such as the following on your Framework Version 2 files to find code that makes multiple changes on a single element.

```
Dom.(add|remove)Class
```

The grep command finds code such as the following:

```
YAHOO.util.Dom.addClass(element, 'rn_Enabled');
YAHOO.util.Dom.removeClass(element, 'rn_Disabled');
```

The equivalent code for Framework Version 3 is as follows:

```
element.addClass('rn_Enabled').removeClass('rn_Disabled');
```

## Listening to events

Use a grep command with the following expression on your Framework Version 2 files to find instances of the addListener function():

```
\.addListener
```

The grep command finds code such as the following:

```
YAHOO.util.Event.addListener(this._displayField, "click",
this._toggleProductCategoryPicker, null, this);
```

The equivalent code for Framework Version 3 is as follows:

```
this._displayField.on('click', this._toggleProductCategoryPicker,
this);
```

## Updating references to assets

Some widget's attributes reference assets. For example, the *icon\_path* attribute for the Print-PageLink widget has a default value of `images/Print.png`. The following procedure applies to standard widgets only since custom widgets manage themes within the *assets/themes* directory.

### *To update references to assets*

- 1 In the *info.yml* file, change the type key for the attribute from `string` to `filepath`. For example:
 

```
attributes:
  icon_path:
    ...
    type: filepath
    default: images/Print.png
```
- 2 If no other widgets or CSS use the asset:
  - a Move the asset into the *themesPackageSource* folder.
  - b If the asset is used in both themes, copy it to the second theme.
- 3 If other widgets or CSS use the asset, copy the asset into the other *themesPackageSource* folders.

## Converting custom code

### Using Connect PHP API in custom scripts

If your Framework Version 2 site includes custom PHP scripts that call normally unavailable product API functions (that is, functions you could use only after applying for access through the whitelist process), you cannot use these functions in Framework Version 3. Instead, check for equivalent functions in the Connect for PHP API. If you find a function in your version 2 site that does not seem to have an equivalent through Connect, contact your Oracle account manager or submit a question on the [Developer Forum](#).

Additionally, the custom PHP scripts on your Framework Version 2 site would have used the following code anywhere you needed Connect functionality:

```
get_cfg_var('doc_root').'/ConnectPHP/Connect_init.php'
```

This code is no longer necessary in Framework Version 3.

## Direct SQL queries

Customer Portal Framework Version 3 does not support direct SQL queries.

## Using PHP5 style constructors

Instead of using the old PHP4 style PHP constructors (where the method name that matches the class name is the constructor), you should use PHP5 style constructors. For example, change this code:

```
function DeviceServiceModelHook () {
    parent::Model ();
}
```

To this:

```
function __construct () {
    parent::__construct ();
}
```

You should convert PHP4 to PHP5 for all custom classes, models, libraries, and other custom code. Refer to [Constructors and Destructors](#) on the PHP site for details.

## Namespacing

Because Framework Version 3 uses PHP namespacing to distinguish between your custom code and the Customer Portal reference implementation files, you'll need to namespace your custom controllers, libraries, models, utilities, and widgets. The following sections provide examples of namespacing for various types of files.

It's a good idea to namespace all your code. When you make a call to a helper or library, add a leading `\` to the call.

For example, if your call to a CI library in Framework Version 2 was:

```
$enc=new CI_Encrypt ();
```

Edit it to this:

```
$enc = new \CI_Encrypt ();
```

By default, PHP is going to look into the current namespace (`\Custom\Models`) for that class, so you need to add the leading backslash so that you escape out into the global scope.

## Custom models

Your custom models must use namespacing to indicate that they are extending RightNow models. The following is an example of using the `Custom` namespace in `/cp/customer/development/models/custom/Sample.php`, where the Base model is extended.

If your code resembles this:

```
class Sample extends Model {
    function Sample() {
        parent::Model();
    }
}
```

You should change it to something like this:

```
namespace Custom\Models;
class Sample extends \RightNow\Models\Base

    function __construct() {
        parent::__construct();
    }
}
```

The following is an example of how you can use namespace aliases in your custom code:

```
use RightNow\Connect\v1_2 as Connect;
```

This offers a shorthand way of referencing namespaces within a file. In this case, you can use the alias `Connect`:

```
Connect\NamedIDLabel();
```

instead of the following when referencing the namespace:

```
RightNow\Connect\v1_2\NamedIDLabel();
```

The model loader is case sensitive, so classes need to match the case of the model. For example, `models/custom/test.php` defines the class `test`. A best practice is to name the file the same as the class name. For example, the class `MyCustomIncident` would be in the `models/custom/MyCustomIncident.php` file, not `mycustomincident.php`.

## PHP namespace requirements

Keep in mind the following requirements for PHP namespaces in your customer portal code:

- When you refer to global objects, such as classes, interfaces, functions, or constants, from within a namespaced class, you must prepend the object with a backslash as shown in the following example.
 

```
throw new \Exception('Invalid state');
```
- When you refer to a namespace in a string, you must escape each backslash as shown in the following example.
 

```
$controller="\RightNow\Controllers\$library";
```
- You must place the namespace declaration at the top of the file before any other code, with the exception of a `declare` statement.

## Detecting abuse in custom code

The code in your custom models or custom controllers may be susceptible to abuse such as SQL injections and other attacks under certain circumstances. Standard models and standard controllers call the native abuse detection methods, so if you use standard controllers with standard models, you do not need to customize your code with abuse detection calls. However, if your custom code performs any direct API calls for database updates or back-end actions, for example, to create or update incidents or contacts, log in a user, or perform answer retrieval, it must call the `\RightNow\Libraries\AbuseDetection::check()` function.

For example, custom code like the following that uses a Connect PHP call to save a contact object must call the `check()` function before the abusible API call.

```
use RightNow\Libraries\AbuseDetection;
AbuseDetection::check();
$myContact->save();
```

Calling the standard model functions from within your parent models usually results in the `check()` function being called. In some cases however, the `check()` function is called in a controller and not a model. For example, the `check()` function is called from the `getAnswer()` function in the `AjaxRequest` controller, rather than from the `get()` function of the `Answer` model. Therefore in such cases, if you create a custom controller endpoint to override a standard controller that calls the `check()` function, your custom code must call `RightNow\Libraries\AbuseDetection::check()` at the top of the relevant function. For example:

```
function submitAsk()
{
    AbuseDetection::check($this->input->post('f_tok'));
    // Code to create an incident
    ...
}
```

If abuse is detected, a CAPTCHA screen is displayed for the user to fill out. Failing to call the `check()` function can result in Connect PHP calls throwing an exception.

## Optional code cleanup

Although it's not necessary, you might want to take the time now to clean up Framework Version 2 code now so that you won't be required to do it at some future migration.

---

## Replacing business objects

Business objects used for input and display widgets now reflect the Connect PHP API naming convention. For example, this is the code in Framework Version 2:

```
<rn:widget path="input/FormInput" name="contacts.login" />
```

In Framework Version 3, the *name* attribute uses the Connect PHP API convention.

```
<rn:widget path="input/FormInput" name="Contact.Login" />
```

Display fields also use the Connect PHP API naming convention, for example:

```
<rn:field name="Answer.Summary" />
```

Although the new framework maps Framework Version 2 business objects to their Framework Version 3 equivalent, it is good practice to edit your code for future use. The following table maps Framework Version 2 business objects with their Framework Version 3 equivalent.

**Note** Note that a custom field example is shown at the end of each section.

Table 6: Version 2 Business Object Equivalents in Version 3

Framework Version 2 Business Object	Framework Version 3 Business Object
Answers	
answers.a_id	Answer.ID
answers.access_level	No Framework Version 3 equivalent
answers.categories	Answer.Categories
answers.created	Answer.CreatedTime
answers.description	Answer.Question
answers.fattach	Answer.FileAttachments
answers.guideID	Answer.GuidedAssistance
answers.lang_id	Answer.Language
answers.m_id (Meta-Answer ID)	No Framework Version 3 equivalent
answers.products	Answer.Products

Table 6: Version 2 Business Object Equivalents in Version 3 (Continued)

<b>Framework Version 2 Business Object</b>	<b>Framework Version 3 Business Object</b>
answers.solution	Answer.Solution
answers.status_id	Answer.StatusWithType.Status
answers.summary	Answer.Summary
answers.type	Answer.AnswerType
answers.updated	Answer.UpdatedTime
answers.url	Answer.URL
answers.c\$answer_custom_field	Answer.CustomFields.c.answer_custom_field
<b>Contacts</b>	
contacts.alt_first_name	Contact.NameFurigana.First
contacts.alt_last_name	Contact.NameFurigana.Last
contacts.c_id	Contact.ID
contacts.channel\$11	Contact.ChannelUsernames.TWITTER. Username
contacts.channel\$12	Contact.ChannelUsernames.YOUTUBE. Username
contacts.channel\$14	Contact.ChannelUsernames.FACEBOOK. Username
contacts.city	Contact.Address.City
contacts.country_id	Contact.Address.Country
contacts.disabled	Contact.Disabled
contacts.email	Contact.Emails.PRIMARY.Address
contacts.email_alt1	Contact.Emails.ALT1.Address
contacts.email_alt2	Contact.Emails.ALT2.Address
contacts.first_name	Contact.Name.First
contacts.full_name	Contact.LookupName



Table 6: Version 2 Business Object Equivalents in Version 3 (Continued)

<b>Framework Version 2 Business Object</b>	<b>Framework Version 3 Business Object</b>
contacts.last_name	Contact.Name.Last
contacts.login	Contact.Login
contacts.ma_mail_type	Contact.Marketingsettings.EmailFormat
contacts.ma_opt_in	Contact.Marketingsettings.Marketingsettings.Optin
contacts.organization_login	Contact.Organization.Login
contacts.organization_name	Contact.Organization.Name
contacts.organization_password	Contact.Organization.NewPassword
contacts.password	Contact.NewPassword
contacts.password_new	Contact.NewPassword
contacts.password_verify	No Version 3 equivalent (Refer to <a href="#">Adding the PasswordInput widget.</a> )
contacts.ph_asst	Contact.Phones.ASST.Number
contacts.ph_fax	Contact.Phones.FAX.Number
contacts.ph_home	Contact.Phones.HOME.Number
contacts.ph_mobile	Contact.Phones.MOBILE.Number
contacts.ph_office	Contact.Phones.OFFICE.Number
contacts.postal_code	Contact.Address.PostalCode
contacts.prov_id	Contact.Address.StateOrProvince
contacts.street	Contact.Address.Street
contacts.survey_opt_in	Contact.Marketingsettings.SurveyOptin
contacts.title	Contact.Title
contacts.c\$contact_custom_field	Contact.CustomFields.c.contact_custom_field
<b>Incidents</b>	
incidents.c_id	Incident.PrimaryContact.ID

Table 6: Version 2 Business Object Equivalents in Version 3 (Continued)

<b>Framework Version 2 Business Object</b>	<b>Framework Version 3 Business Object</b>
incidents.cat	Incident.Category
incidents.closed	Incident.ClosedTime
incidents.contact_email	Incident.PrimaryContact.Emails. PRIMARY.Address.Emails.PRIMARY.Address
incidents.contact_first_name	Incident.PrimaryContact.Name.First
incidents.contact_last_name	Incident.PrimaryContact.Name.Last
incidents.created	Incident.CreatedTime
incidents.fattach	Incident.FileAttachments
incidents.i_id	Incident.ID
incidents.org_id	Incident.Organization.ID
incidents.prod	Incident.Product
incidents.ref_no	Incident.ReferenceNumber
incidents.sla	Incident.SLAInstance
incidents.status_id	Incident.StatusWithType.Status
incidents.subject	Incident.Subject
incidents.thread	Incident.Threads
incidents.updated	Incident.UpdatedTime
incidents.c\$incident_custom_field	Incident.CustomFields.c.incident_custom_field

## Replacing unused widgets

Three widgets that were available in Framework Version 2 are no longer used in Framework Version 3: ChatLaunchFormOpen, MobileEmailAnswerLink, and ContactNameInput. They remain available so as not to break your pages when you migrate to the newer framework, but you will see a warning in the development header for the page. Good practice recommends replacing them with their Framework Version 3 equivalents.

## ChatLaunchFormOpen

The ChatLaunchFormOpen widget has been replaced with non-widget functionality.

*To replace the ChatLaunchFormOpen widget*

1 Open the *chat\_launch.php* file in *customer/development/views/pages/chat*.

2 Locate the following line of code.

```
<rn:widget path="chat/ChatLaunchFormOpen"
  label_form_header="#rn:msg:CHAT_MEMBER_OUR_SUPPORT_TEAM_LBL#" />
```

3 Replace the code you located in step 2 with the following code.

```
<span
  class="rn_ChatLaunchFormHeader">#rn:msg:CHAT_MEMBER_OUR_SUPPORT_TEAM_LBL#</span>
<form id="rn_ChatLaunchForm"
  method="post"
  action="/app/chat/chat_landing">
```

4 Save *chat\_launch.php*.

## MobileEmailAnswerLink

The MobileEmailAnswerLink widget has been replaced with the EmailAnswerLink widget.

*To replace the MobileEmailAnswerLink widget*

1 Open the *detail.php* file in *customer/development/views/pages/mobile/answers*.

2 Locate the following line of code.

```
<rn:widget path="utils/MobileEmailAnswerLink"
  label_link="#rn:msg:EMAIL_THIS_PAGE_ARROW_LBL#" />
```

3 Replace the code you located in step 2 with the following code.

```
<rn:widget path="utils/EmailAnswerLink" />
```

4 Save *detail.php*.

## ContactNameInput

Although the ContactNameInput widget was retained in Framework Version 3 so as not to break any of your pages that use this widget, it is no longer used in the reference implementation. The following Framework Version 2 pages of the reference implementation used the ContactNameDisplay widget.

- `account/profile.php`
- `mobile/account/profile.php`
- `utils/create_account.php`
- `mobile/utils/create_account.php`
- `chat/chat_launch.php`
- `mobile/chat/chat_launch.php`

The following procedure is an example of replacing the `ContactNameDisplay` widget on the Account Settings page with the `FormInput` widget, but the process is similar on the other pages where the `ContactNameDisplay` widget occurs.

### *To replace the `ContactNameDisplay` widget on the Account Settings page*

- 1 Open the `profile.php` file in `customer/development/views/pages/account`.
- 2 Locate the following line of code.

```
<rn:widget path="input/ContactNameInput"
required="true"/>
```

- 3 Replace the code you located in step 2 with the following code.

```
<rn:condition config_check="intl_nameorder == 1">
  <rn:widget path="input/FormInput"
  name="Contact.Name.Last"
  label_input="#rn:msg:LAST_NAME_LBL#"
  required="true"/>
  <rn:widget path="input/FormInput"
  name="Contact.Name.First"
  label_input="#rn:msg:FIRST_NAME_LBL#"
  required="true"/>
</rn:condition_else/>
  <rn:widget path="input/FormInput"
  name="Contact.Name.First"
  label_input="#rn:msg:FIRST_NAME_LBL#"
  required="true"/>
  <rn:widget path="input/FormInput"
  name="Contact.Name.Last"
  label_input="#rn:msg:LAST_NAME_LBL#"
  required="true"/>
</rn:condition>
```

- 4 Save `profile.php`.
-

## Staging and promoting version 3.1

**Important** If you had additional page sets enabled in Framework Version 2 (such as mobile) you must select Copy to Staging in the Action drop-down menu for those page sets in the **Select Configurations** step of the [staging process](#). If you do not copy the page set mappings to staging, those page sets will not be available on your customer portal after you promote the staging environment to production.

- [Overview of the staging and promoting processes](#)
- [Staging the customer portal pages](#)
- [Promoting your customer portal pages](#)

