



SMARTGUIDE[®] 7.0

ASPX TEMPLATE AUTHOR GUIDE

© 2017 Alphinat Inc. All rights reserved.

Alphinat SmartGuide® – ASPX Template Author Guide
November 2017

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Alphinat Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Alphinat Incorporated. Alphinat Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide. Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner. Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization.

Alphinat, SmartGuide, Smartlets and the Alphinat logo are either registered trademarks or trademarks of Alphinat Incorporated in Canada and/or other countries. All other trademarks are the property of their respective owners.

Alphinat Inc., 2000 Peel St. #680, Montreal (Qc), H3A 2W5, Canada.

Content

| | |
|---|-----------|
| Introduction | 5 |
| Purpose | 5 |
| Audience | 5 |
| References..... | 5 |
| Getting started | 7 |
| Understanding how SmartGuide tags work..... | 7 |
| Setting up your environment..... | 9 |
| First steps..... | 10 |
| Overview of the default theme | 18 |
| Integrating SmartGuide in your website or application | 19 |
| Developing themes..... | 21 |
| Developing themes that meet your specific needs | 21 |
| AJAX behavior of default theme | 40 |
| Theme localization..... | 41 |
| Modifying the layout file..... | 43 |
| Exposing CSS classes..... | 46 |
| Assigning themes to Smartlets | 48 |
| Creating custom controls | 50 |
| Adding theme templates | 52 |
| Embedding the runtime server | 54 |
| Multiple Smartlets per page | 55 |
| Cheat sheet..... | 58 |
| Smartlet | 58 |
| Page..... | 62 |
| Field | 64 |
| Value..... | 74 |
| PDF templates..... | 76 |
| ASPX Tags Reference..... | 78 |
| Register directive | 78 |
| <apn:control> | 79 |
| <apn:forEach> | 83 |
| <apn:label/> | 87 |
| <apn:SmartGuide/> | 88 |

| | |
|-------------------------------------|------------|
| <apn:translate/> | 89 |
| <apn:localize/> | 91 |
| <apn:API5/> | 92 |
| <apn:clearAPI5/> | 93 |
| <apn:cssclass/> | 94 |
| <apn:cssstyle/> | 94 |
| <apn:name/> | 95 |
| <apn:value/> | 96 |
| <apn:code/> | 97 |
| <apn:controlattribute/> | 97 |
| <apn:ifcontrol> | 98 |
| <apn:ifnotcontrol> | 100 |
| <apn:ifcontrolattribute> | 101 |
| <apn:ifnotcontrolattribute> | 103 |
| <apn:controllayoutattribute/> | 104 |
| <apn:ifsmartletmultilingual> | 105 |
| <apn:locale> | 106 |
| <apn:choosecontrol> | 107 |
| <apn:whencontrol> | 108 |
| <apn:otherwise> | 109 |
| <apn:ifcontrolrequired> | 111 |
| <apn:ifnotcontrolrequired> | 112 |
| <apn:ifrequiredcontrolexists> | 113 |
| <apn:ifcontrolvalid> | 113 |
| <apn:ifnotcontrolvalid> | 114 |
| <apn:tooltip/> | 115 |
| <apn:help/> | 116 |
| <apn:helpid/> | 117 |
| <apn:ifhelplink> | 117 |
| <apn:ifnohelplink> | 118 |
| <apn:jsfields/> | 119 |
| <apn:metadata> | 120 |
| <apn:metadatatype/> | 121 |
| <apn:hasmetadata> | 122 |
| Attributes..... | 124 |
| The Type attribute | 124 |
| The items attribute..... | 127 |
| List of control types..... | 128 |

Introduction

1

Purpose

Whether you wish to create web or mobile applications that guide your clients, partners and employees through complex procedures, tasks or information silos, SmartGuide allows you to provide user experiences that maximize success and compliance rates.

A big part of that user experience lies in the presentation layer that is covered in this guide. As a .Net template author, you will be using SmartGuide tags to create accessible, responsive themes that can be reused across any number of Smartlets® – thus ensuring consistency across your applications and greatly reducing development time.

Audience

This material is aimed at people with a technical background interested in creating and maintaining .Net-based SmartGuide themes and templates. Using the tags does not require a deep knowledge of .Net which means someone with a good grasp of HTML and/or JavaScript should be able to follow along just fine. Please note however that this document is not a tutorial on .Net, HTML, JavaScript or CSS. You can explore the links provided in the references section below for more information on those subjects. This document also does not cover the use of the SmartGuide JSP tags nor how to create templates using the SmartGuide REST interface which are covered in other user guides.

References

- <http://msdn.microsoft.com/library/default.aspx>
- <http://msdn.microsoft.com/en-us/vstudio/hh388573>
- <http://msdn.microsoft.com/en-us/vstudio/hh341490>
- <https://www.codecademy.com/learn/web>
- <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- <http://www.csstutorial.net/>

- <http://getbootstrap.com/>
- <http://www.materializecss.com>

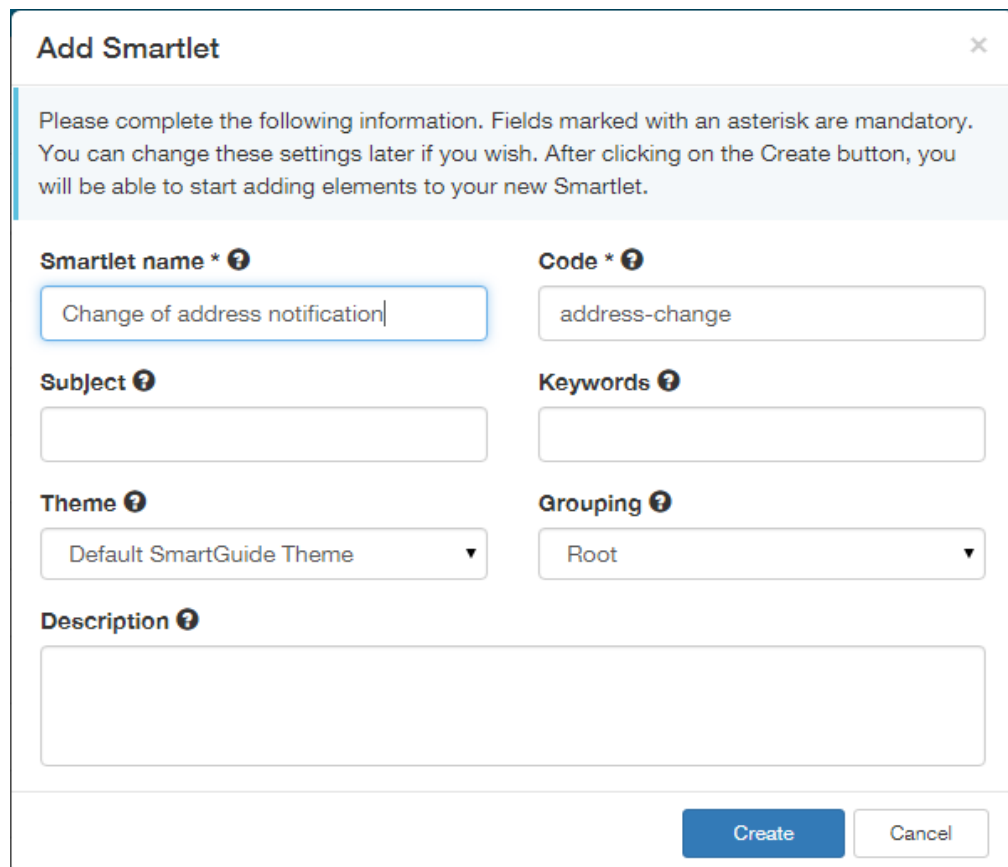
Getting started

2

Understanding how SmartGuide tags work

If SmartGuide Designer allows quickly building the business content and logic of an application, the SmartGuide tags are all about bringing that information to life for your end users with the help of ASPX templates.

In this guide, you will learn how to use the SmartGuide tags to build accessible, responsive presentation themes that can be shared across multiple applications. Indeed, you certainly wouldn't want to unnecessarily create or duplicate ASPXs every time you built a new Smartlet. So instead of inserting your application content directly within your ASPX code, you can refer to it using the SmartGuide tags. For example, to display the Smartlet name that was defined in SmartGuide Designer:



The screenshot shows a dialog box titled "Add Smartlet" with a close button (X) in the top right corner. Below the title bar, there is a light blue informational box containing the text: "Please complete the following information. Fields marked with an asterisk are mandatory. You can change these settings later if you wish. After clicking on the Create button, you will be able to start adding elements to your new Smartlet." Below this box, the form is organized into two columns. The left column contains: "Smartlet name * ?" with a text input field containing "Change of address notification"; "Subject ?" with an empty text input field; "Theme ?" with a dropdown menu showing "Default SmartGuide Theme"; and "Description ?" with a large empty text area. The right column contains: "Code * ?" with a text input field containing "address-change"; "Keywords ?" with an empty text input field; "Grouping ?" with a dropdown menu showing "Root"; and a "Create" button at the bottom right. A "Cancel" button is located to the right of the "Create" button.

You could refer to it in the following manner in your ASPX template:

```

<%@ Register Tagprefix="Apn"
Namespace="Alphinat.SmartGuideServer.Controls"
Assembly="apnsgscontrols" %>
...
<div class="row-fluid">
    <div class="col-md-12">
        <h1>
            <apn:control runat="server" type="smartlet-
name">
                <apn:value runat="server"/>
            </apn:control>
        </h1>
    </div>
</div>
...

```

Which, depending on the rest of the ASPX, HTML and CSS code defined in the theme, might look like this when presented to end users on desktop browsers:

Change of address notification

- ✓ **Exclusive Mover Savings** Get instant access to over \$500 in valuable coupons
- ✓ **Safe and Secure** Safeguard your information with ID verification by a simple \$1.05 charge to your credit or debit card
- ✓ **Speed and Convenience** Save a trip to the post office
- ✓ **Email Confirmation** Receive an immediate email confirmation of your Change of Address
- ✓ **MyMove.com Local Information, Tools and Offers** Make your move complete with catalog forwarding services, neighborhood deals and more at MyMove.com

Note The person who prepares this form states that he or she is the person, executor, guardian, authorized officer, or agent of the person for whom mail would be forwarded under this order. Anyone submitting false or inaccurate information on this form is subject to punishment by fine or imprisonment or both under Sections 2, 1001, 1702 and 1708 of Title 18, United States Code.

Privacy Act Statement

Your information will be used to provide you with mail forwarding and change of address services. Collection is authorized by 39 U.S.C. 401, 403, and 404. Providing the information is voluntary, but if not provided we will not be able to process your request. We do not disclose your information to third parties without your consent, except to facilitate the transaction, to act on your behalf or request, or as legally required. This includes the following limited circumstances: to a congressional office on your behalf; to financial entities regarding financial transaction issues; to a U.S. Postal Service (USPS) auditor; to entities, including law enforcement, as required by law or in legal proceedings; to contractors and other entities asking us to fulfill the service (various providers); to federal, state, local or foreign government agencies regarding personnel matters or for the performance of its duties; for the service of legal process; for voter registration purposes; for jury service duties; to a disaster relief organization if the address has been impacted by a disaster or manmade hazard; to individuals or companies already in possession of your name and old mailing address, as an address correction service. Information will also be provided to licensed service providers of the USPS to perform mailing list correction service of lists containing your name and old address. A list of these licensed service providers can be obtained at the following URL: https://tools.usps.com/go/ToolsAction.do?_afz=guis&CERTIFIED_LICENSEES. For more information regarding our privacy policies visit www.usps.com/privacypolicy.

[Privacy Policy](#)

I understand and acknowledge the statements above.
Continue.

Need to [view, update or cancel a Change Of Address order](#) you already submitted?

Note: A valid credit card and a valid email address are required to complete the Online Change of Address process. If you are unable to use a credit card and a valid email address, you will have the option to print the Change of Address form and then mail or deliver the printed form to your [local post office](#)™.

Browsers USPS Recommends: For the best experience while filing your change of address, we recommend using Internet Explorer 9 or higher, Firefox 31 or higher, or Chrome 38 or higher.

MOVER'S GUIDE® [change of address help](#)

Copyright © 2015 Imagitas, Inc. All Rights Reserved

This clear separation of the content and presentation layers allows you to build a theme once and then use it in multiple applications. Of course, if each of your applications requires a different look and feel, then you could create a custom theme for every one. Or, if only one page needs to be displayed differently, you could create a specific template and add it to your theme. We will cover these different scenarios in this guide and explain the best approach depending on what you wish to achieve.

Let's now get you set up in SmartGuide and have a closer look at how to build a customized presentation theme.

Setting up your environment

Before starting to work on themes in SmartGuide Designer it is strongly recommended that you modify the *Web.config* configuration file of SmartGuide Designer so that the parameter *alphinat.sgs.themes.path* points to your *smartlets/aspx* directory on disk and the parameter *alphinat.sgs.themes.synchronize*

has the value *fromdisk*. This way, you will be able to quickly test the modifications you make to your .Net templates.

Secondly, although SmartGuide Designer allows developing themes through its web interface, you will probably find it more convenient to code the .Net templates using your favorite editor or even within Visual Studio where IntelliSense will speed up coding. Either way, if you configure your environment as described above, you will see your changes reflected upon refreshing your browser window while viewing a Smartlet that is using the theme being edited.

First steps

In order to test your themes in the following sections, we recommend you create a few sample Smartlets. For example, you could create one that contains all field types on a single page to test the look & feel of each one. You should also add field validations, again to see how error messages will appear using your theme.



As a first step of theme customization we will change the logo in the header section. We will then change the font and color scheme and display a validation error message right under a field as opposed to the top of the page. We will be using the Bootstrap-based default theme during the course of the examples.

The first step is to duplicate the default theme. Indeed, you should not make modifications to the default SmartGuide theme itself to ensure you always have a working version you can refer to. You can either make a copy of the default theme on disk using your file explorer, making sure to rename the root folder of the copy, or you can use SmartGuide Designer to do so. For the purpose of this example we will call the new theme *MyTheme*.

➤ To duplicate the default theme:

1. Click on the **Themes** tab in the main SmartGuide Designer navigation bar:



2. Click on the  icon next to *Default SmartGuide Theme* and select the **Duplicate** option or hover over the label and click on the  icon.
3. Modify the **Theme name**.

The theme name will help to identify it in different areas of SmartGuide Designer. Make sure you enter a descriptive and user-friendly name to make it easy to find and use.

4. Modify the Root folder name.

The root folder name is the name of the directory that will be created on disk. If you enter *custom* for example, a *custom* folder will appear under the main *aspx* folder at the same level as *default*, which corresponds to the default SmartGuide theme. The folder name must therefore be unique and should comply with the naming convention of the file system you are using. To ensure maximum portability, we recommend using short names that do not contain any accents or special characters.

5. Optionally enter the theme Description.

If you leave this blank, the root folder name will be displayed when mousing over the theme in the theme list.


6. Click on the Save button.


Once a theme is duplicated you can modify it without impacting the original theme and you will be able to assign it to one or more Smartlets.


Once the default theme is duplicated, make sure you assign it to your test Smartlets under the properties tab:

[Pages](#)
[Services](#)
[Files](#)
[Actions](#)
[Properties](#)

You can edit the Smartlet properties below. Fields marked with an asterisk are mandatory.

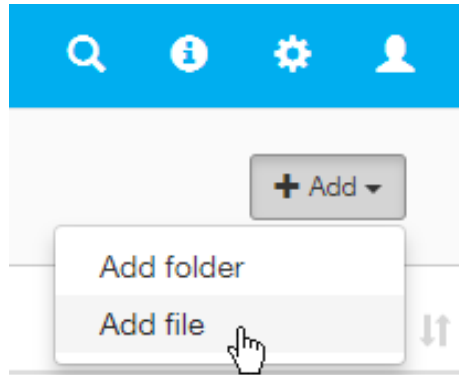
Smartlet name * 

Subject 

Theme 

Changing the logo

To change the logo, first expand your newly created theme, expand the *resources* folder followed by the *img* folder. Then, click on the *Add* button at the top right of the page and select the *Add file* option:



Choose the logo file you wish to use and click the *Save* button. If the file name already exists, remove the old file and try again. The file name is referenced in the *default.aspx* file under the *default* folder. Adjust it as needed on line 100:

```
<img class="logo" src='<% =Page.TemplateSourceDirectory + "/" +  
"../resources/img/logo.jpg"%>' />
```

Alternately, you can reference an external image if you wish, e.g. ``.

The *logo* class is defined in the *custom.css* stylesheet located under the *resources/css* folder. For example, the width of the logo is reduced on smaller devices using the following definitions on lines 41 to 45:

```
@media (max-width: 768px) {  
    img.logo {  
        max-width: 60%;  
    }  
}
```



We highly recommend that you place all your Bootstrap and SmartGuide CSS overwrites in the *custom.css* file to simplify maintenance and to facilitate future upgrades to the latest default theme files.

Once your changes have been made, go back to the *Smartlets* tab to view your Smartlet. You should now see your logo displayed at the top before the page title and progress bar.

Changing the font and color scheme

The next exercise consists in changing the font and color scheme of the theme. We will change the default settings to customize the look as follows:

Make sure that page sections are defined in your test Smartlet to see the breadcrumb trail seen above. Otherwise, a progress bar will be shown.

All of the changes will be made to the *resources/css/custom.css* file and should take around 10 minutes to complete. Go to the *Themes* tab of SmartGuide Designer and navigate to that file of your custom theme to edit it. The first thing we will do is import a different font on line 3:

```
@import
url("https://fonts.googleapis.com/css?family=Barlow+Semi+Con
densed");
```

We can then reference the new font on lines 5 and 13:

```
font-family: 'Barlow Semi Condensed', sans-serif;
```

On to the colors. We'll start by changing the background color of the main navigation bar. We can do this on line 17 by replacing the current value with the following one:

```
background-color: #c0ca33;
```

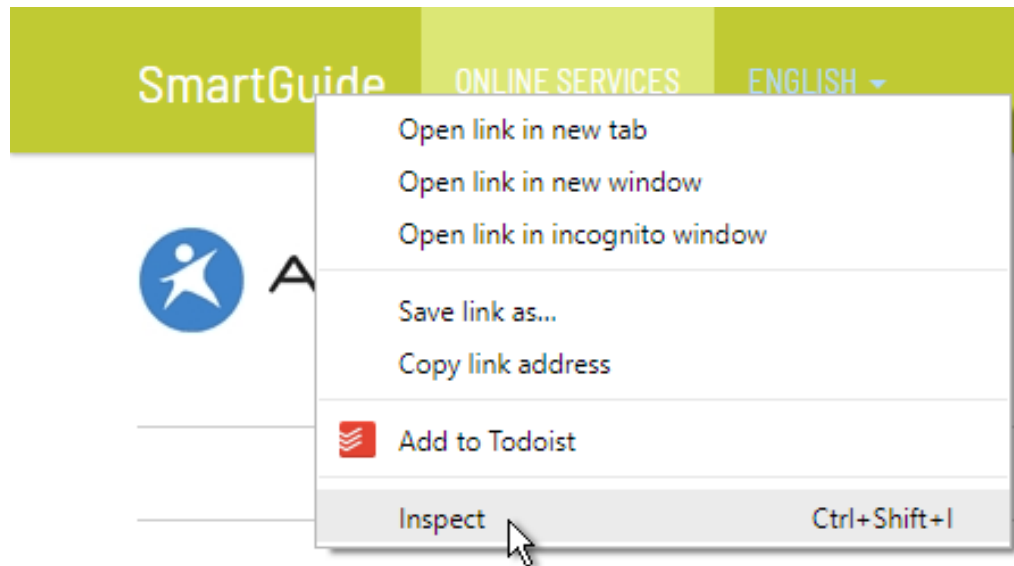
We'll want to use the same color for the page headings as well and for the background of the *Next* navigation button. So, the next step is to replace the color value on lines 52, 77 and 88 with the same Hex color code as above. While we're making the change to the *next page* button, we can add the following definitions after line 88 to change the text color and weight:

```
.btn-primary.next {  
    background-color: #c0ca33;  
    color: #000;  
    font-weight: 400;  
}
```

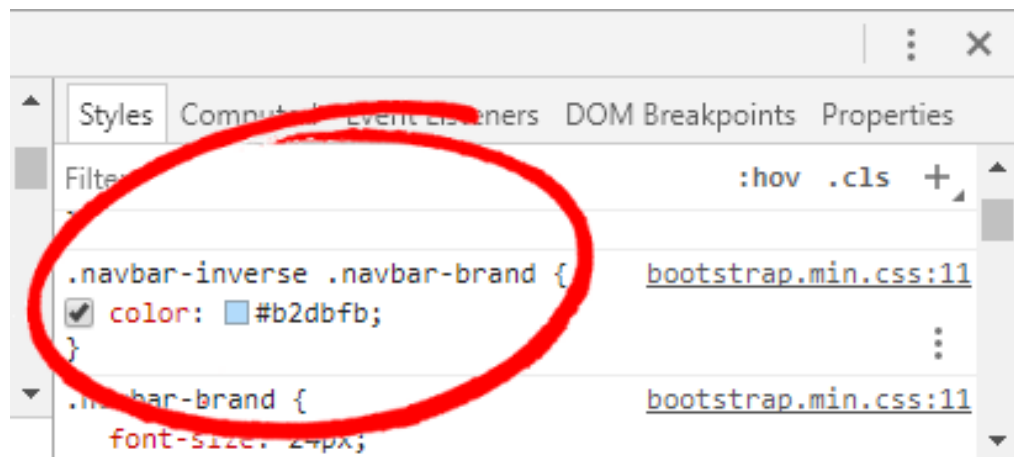
We will now set the background color of the active section in the main navigation bar, the current section in the breadcrumb trail and when hovering over the *Next* navigation button to a lighter shade of lime green. This can be changed by replacing the current Hex color code with #dce775 on lines 23, 73 and 92 (or 94 if you added the definitions to the *next page* button detailed above).

We also want to change the font color to black when a section is highlighted in the navigation bar and in the breadcrumb trail. To accomplish this, we need to replace #ffffff with #000 on line 22 and white with #000 on line 68.

For the rest of the changes, we will be adding new Bootstrap and SmartGuide overwrites that are not yet present in the *custom.css* file. One of the simplest ways to do so is with the help of your browser's developer tools. Using Google Chrome for example, right-click on **SmartGuide** at the top of the page and select *Inspect*:



Scroll down in the *Styles* panel to see where the color is defined:



Select that definition, copy it and then paste it into the custom.css file, after the *.navbar-inverse* definition for example:

```
/* Navigation bar */
.navbar-inverse {
    background-color: #c0ca33;
}
.navbar-inverse .navbar-brand {
    color: #b2dbfb;
}
```

```
}
```

Then simply change `#b2dbfb` to `#000` in the highlighted portion to change the *SmartGuide* text to black.



For more information about inspecting styles, please visit the official documentation for [Chrome](#), [Firefox](#) and [Edge](#).

We can proceed the same way to change the color of the links in the main navigation bar. Right-click on the **Alphinat.com** link and select *Inspect*. Copy the color definition from the *Styles* panel and paste it after the definition of the first set of links in the navigation bar for example:

```
/* First set of links in top navigation bar */
.navbar-collapse > ul:first-of-type a {
  text-transform: uppercase;
}
.navbar-inverse .navbar-nav>li>a {
  color: #b2dbfb;
}
```

Then replace `#b2dbfb` with `#000` in the highlighted portion.

To change the background color of open links (e.g. when the user opens the Smartlet language selector), you can paste the following code right after the one we just added:

```
.navbar-inverse .navbar-nav>.open>a, .navbar-inverse
.navbar-nav>.open>a:hover, .navbar-inverse .navbar-
nav>.open>a:focus {
  background-color: inherit;
  color: #ffffff;
}
```

To tone down the current page section, we can add the following code after the existing `.section` definitions for example:

```
.section li p.current strong {
  font-weight: 400;
}
```


Finally, to change the color that is displayed under form elements when they are in focus, we can paste the following code right before the `/* Next page button` `*/` definitions for example:

```
/* Form element highlight */
.form-control:focus {
    -webkit-box-shadow: inset 0 -2px 0 #c0ca33 !important;
    -moz-box-shadow: inset 0 -2px 0 #c0ca33 !important;
    box-shadow: inset 0 -2px 0 #c0ca33 !important;
}
```

That's it for now. We can now view the Smartlet to see the end result of all our changes.

The default SmartGuide theme is based on the [Bootswatch Paper](#) example so you may want to head over to their site for more customization ideas. As with the changes above, just remember to place your overwrites in the `custom.css` file to simplify theme maintenance and future upgrades.

As a final exercise we will add a validation message right below a field.

Adding an error message under a field

In this example, we will display a message right under input fields when a validation error occurs. In the default theme, practically each field type has its own ASPX file to allow for full customization and reuse of styling elements. So, to make the change to the input fields, we need to modify the `input.aspx` file located in the `default/controls` folder. Under the *Themes* tab, navigate to the `input.aspx` file of your custom theme to edit it. Then towards the end of the file, right after the line:

```
<apn:ifcontrolattribute runat='server' attr='prefix or
suffix'></div></apn:ifcontrolattribute>
```

Add the following lines:

```
<label>
<span class="small alert-danger">
<%=control.Current.GetAlert()%>
</span>
</label>
```

To avoid duplicating the error messages, we will comment the line that loads the *validation.aspx* file. To do so, you need to edit the *default.aspx* file located in the *default* folder of your theme. Towards the bottom part of the file you will see a comment `<!-- Validation messages -->`. Just comment the line below so it looks like this:

```
<!-- Server.Execute(Page.TemplateSourceDirectory + "/" +  
"controls/validation.aspx"); --%>
```

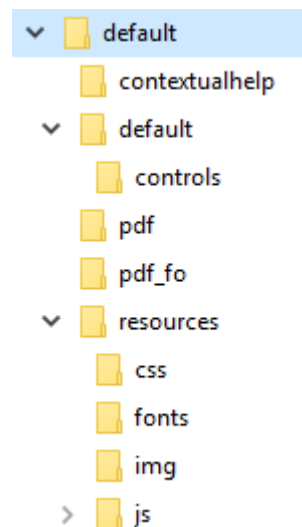
You can now test a Smartlet which has a field validation rule, for example by checking the *Mandatory field* box of a text field (don't forget to set a message). Trigger the validation, for example by clicking on the *Next* button of a multipage Smartlet without entering any values. You should see the message appear in red just under the input field.

Repeat these steps as needed for other field types.

Let's now have a closer look at the default SmartGuide theme and how it uses the SmartGuide tags. To follow along, you can either access the default theme files on disk or through SmartGuide Designer.

Overview of the default theme

The default theme has the following structure:



- **contextualhelp** – In the default theme, detailed help associated to a field is shown in modal windows. If you prefer to display contextual help on distinct pages instead, you would specify the formatting instructions in this folder. If used, its name should not be modified.
- **default** – The *default* folder is the entry point for the theme and determines the general appearance of the Smartlet. It refers to other ASPX and resource files. This folder should not be removed and its name should not be modified.
 - **controls** – This sub folder contains a series of ASPX files that determine the appearance of Smartlet elements. For example, the *input.aspx* file contains display instructions for text fields, including where to place the label, the help icons and tooltips.
- **pdf** – The *pdf* folder holds a series of ASPXs that determine how to format a PDF version of a Smartlet page.
- **pdf_fo** – The *pdf_fo* folder holds a series of ASPXs that determine how to format a PDF version of a Smartlet using XSL-FO statements.
- **resources** – This folder contains the CSS stylesheets, fonts, images, JavaScript and JSON files used by the default theme.



Note: The default theme is a sample implementation using the SmartGuide tags. As we will see later in this guide, with a few exceptions you can structure your ASPX theme as you see fit.

Integrating SmartGuide in your website or application

There are several ways to integrate SmartGuide in your existing website or application. In this section, we look at a number of scenarios and guide you to the appropriate sections for more details.

The most determining factor when integrating SmartGuide is the technology used by your existing website or application and how closely tied you require the SmartGuide applications you will develop to be. The table below covers the various possible cases.

| Technology | Suggested approach |
|---|---|
| .NET application with ASPX pages, Smartlets in separate pages. | <p>You may already have a complete application or website but want to deploy Smartlets independently.</p> <p>You can have SmartGuide Server as a separate application. In that case, you will want to develop a</p> |

theme that reproduces the look of your existing application or website. You can refer to the section [Developing themes](#). Either subsection is applicable as you can either develop a theme in tag mode or in http handler mode, which is the default used by a new SmartGuide installation.

.NET application with ASPX pages, Smartlets embedded in existing pages.

You already have a complete application or website and want to integrate Smartlets in parts of the application. In such a case, you will want to integrate the SmartGuide Server runtime in your application and most likely use the tag mode to render the Smartlet.

You can refer to section [Embedding the runtime server](#) for the first part, and then to the subsection [Creating a theme using external templates](#) for details on the tag mode.

Sharepoint

Please refer to the Sharepoint SmartGuide Integration documentation.

Non .NET based application like PHP

If you use a PHP based website, like Wordpress or Drupal, and want to integrate SmartGuide applications you will need to use a PHP or HTML/JavaScript theme connecting to SmartGuide running under IIS.

To do so, you will need to enable and make use of the AJAX mode of SmartGuide server. Refer to the Developer guide for more information.



Note: There are other possible integration cases, like using an iframe to load a SmartGuide application in an existing site. You can follow the second case in the table above as it applies.

Developing themes

3

Developing themes that meet your specific needs

While one of the default SmartGuide themes can be used to jumpstart your development and immediately view Smartlets that are under construction, you most likely have a target look and feel that you would like to implement before going into production.

If you only have a mockup, you could customize a duplicate of the default theme to achieve the desired look and feel with the help of Bootstrap or Materialize. If you are unfamiliar with these front-end frameworks, you will find documentation and examples on the official [Bootstrap](#) and [Materialize](#) websites. At the time of this writing, version 4 of Bootstrap was in beta so make sure you look up the documentation for Bootstrap 3.3.7 which is the version used by the default SmartGuide theme. During the course of your implementation you can refer to this guide to learn more about the SmartGuide tags you encounter.

If you wish to create a presentation theme based on (or integrated within) an existing web page, the following instructions will show you how.

Creating a theme based on an existing website

Follow these step-by-step instructions if you wish to use an existing web page as the basis for your new theme:

- [Using Bootstrap](#)
- [Using Materialize](#)
- [Using another framework](#)

Using Bootstrap

The following instructions apply if the page that will serve as the basis of the new theme uses the [Bootstrap](#) framework.

➤ To create a new Bootstrap-based theme:

1. Duplicate and rename the default theme (as detailed [above](#)).

2. Save the HTML source code of the web page that will serve as a template in a new file named *default.aspx*. Place this file under the *default* directory of the new theme (therefore overwriting the existing *default.aspx* file).
3. Set the page language at the top of the new *default.aspx* file, e.g.:

```
<%@ Page Language="C#" %>
```

4. Next, add a directive at the top of the *default.aspx* file to include the SmartGuide tag library (this is required on every template that will use the SmartGuide ASPX tags):

```
<%@ Register Tagprefix="Apn"  
Namespace="Alphinat.SmartGuideServer.Controls"  
Assembly="apnsgscontrols" %>
```

5. Add the following code to obtain the current locale:

```
<apn:locale runat="server" id="loc">  
<% Context.Items["currentLocale"] =  
loc.Current.GetValue(); %>  
</apn:locale>
```

You can then use this variable throughout the *default.aspx* file. To reference the currently selected locale to declare the document language for example:

```
<html lang="<%=Context.Items["currentLocale"]%>">
```

6. Place the following meta tags within the *<head>* portion if you wish to populate them with content defined in SmartGuide Designer:

```
<meta name="application-name" content="<apn:control  
runat="server" type="smartlet-name"><apn:value  
runat="server"/></apn:control>">  
<meta name="description" content="<apn:control  
runat="server" type="smartlet-description"><apn:value  
runat="server"/></apn:control>">  
<meta name="keywords" content="<apn:control runat="server"  
type="keyword"><apn:value runat="server"/></apn:control>"  
</>
```

You can check the source code of the default theme for more examples of meta tags you can add.

7. Reference the following files in the *<head>* portion of *default.aspx* (these files are located in the *resources/css* folder of your custom theme):

```
<!-- Bootstrap CSS -->
```

```
<link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/bootstrap.min.css" %>' rel="stylesheet">
<!-- Datatable support -->
<link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/datatables.min.css" %>'
rel="stylesheet">
<!-- Date widget support -->
<link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/bootstrap-datepicker.min.css" %>'
rel="stylesheet">
<!-- Autocomplete support -->
<link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/autocomplete.min.css" %>'
rel="stylesheet">
<!-- SmartGuide CSS -->
<link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/smartguide.min.css" %>'
rel="stylesheet">
<link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/custom.css" %>' rel="stylesheet">
<!-- Load base jQuery --><%-- in case some fields output
html/js requiring jQuery --%>
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/jquery.min.js" %>' ></script>
```

- **bootstrap.min.css** – The default SmartGuide theme uses the [Bootswatch Paper](#) template. To use another Bootstrap template, simply overwrite the bootstrap.min.css file in the *resources/css* folder.
- **datatables.min.css** – To support table sorting and filtering.
- **bootstrap-datepicker.min.css** – This file contains the styling information for the date picker.
- **autocomplete.min.css** – To turn drop-down fields that have an *autocomplete* class assigned to them into autocomplete controls.
- **smartguide.min.css** – This file contains formatting instructions for multiple page and form elements (section breadcrumb, alert messages, datatables, navigation buttons, etc.).
- **custom.css** – Of course, you can reference additional CSS files before or after the ones listed above. If you wish to overwrite certain definitions, we highly recommend placing your Bootstrap and SmartGuide overwrites in the custom.css file to keep the original files intact and facilitate maintenance and upgrades. Due to the nature of cascading style sheets, make sure this file comes last to ensure overwrites are taken into effect.

8. **jquery.min.js** – jQuery is required to bring the dynamic interactions defined in SmartGuide Designer to life. We suggest that you import the jQuery library before the end of the <head> section since certain fields might output HTML or JavaScript code that rely on it.
9. Add the following code to the <body> tag to display the CSS classes and styles defined for a page in SmartGuide Designer:

```
class='<apn:control runat="server"
type="step"><apn:cssclass runat="server"/></apn:control>'
style='<apn:control runat="server"
type="step"><apn:cssstyle runat="server"/></apn:control>'
```

10. Place the following code where the page title should appear:

```
<apn:control runat="server" type="step"><apn:label
runat="server"/></apn:control>
```

11. Adjust the paths to images, JavaScript files and stylesheets:

- Use either an absolute URL to point to external resources or:
- Copy resources to the *resources* sub folders and modify the paths to source them from your new theme, for example:

```
<link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/mytheme.css"%>' rel="stylesheet">
```

12. Place the following code where the SmartGuide form elements should appear:

```
<form id='smartguide_<apn:control runat="server"
type="smartlet-code"><apn:value
runat="server"/></apn:control>' action="do.aspx"
method="post" enctype="multipart/form-data">
<div id="sglib"><%
Server.Execute(Page.TemplateSourceDirectory +
"/sglib.aspx"); %></div>
<!-- form validation -->
<% Server.Execute(Page.TemplateSourceDirectory + "/" +
"../controls/validation.aspx"); %>
<!--loop over page controls -->
<div id="sgControls">
    <% Server.Execute(Page.TemplateSourceDirectory + "/" +
"../controls/controls.aspx"); %>
</div>
<!-- navigation buttons -->
<div class="navigation">
```



```
<% Server.Execute(Page.TemplateSourceDirectory + "/" +
"../controls/navigation.aspx"); %>
</div>
</form>
```

The SmartGuide form and the *sgControls* div are referenced by their ID in the *smartguide.js* file which adds dynamic behavior to the theme. The *sglib* div is required since version 7 of SmartGuide to support JavaScript field actions. More details can be found in the section [AJAX behavior of default theme](#). The *action*, *method* and *enctype* properties should be left as is. You can otherwise remove or overwrite the classes in the above code.

13. Reference the following files before the closing `</html>` tag (these files are located in the *resources/js* folder of your custom theme) to enable the dynamic features of SmartGuide (e.g. datatables, date picker, AJAX form submission, JavaScript actions on fields, etc.):


```
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/jquery.form.min.js" %>' ></script>
<!-- JQuery UI -->
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/ui/jquery-ui-1.10.4.custom.min.js" %>'
></script>
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/bootstrap.min.js" %>' ></script>
<!-- HTML5 shim and Respond.js for IE8 support of HTML5
elements and media queries -->
<!--[if lt IE 9]>
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/html5shiv.min.js" %>' ></script>
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/respond.min.js" %>' ></script>
<![endif]-->
<!-- IE10 viewport hack for Surface/desktop Windows 8 bug
-->
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/ie10-viewport-bug-workaround.js" %>'
></script>
<!-- Date widget support -->
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/moment.min.js" %>' ></script>
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/locale/fr.js" %>' ></script>
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/locale/en-ca.js" %>' ></script>
```

```

<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/bootstrap-datepicker.min.js" %>'
></script>
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/locale/bootstrap-datepicker.fr.min.js"
%>' ></script>
<!-- Datatables support -->
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/datatables.min.js" %>' ></script>
<!-- Autocomplete support -->
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/jquery.autocomplete.min.js" %>'
></script>
<!-- Inputmask support -->
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/inputmask.min.js" %>' ></script>
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/inputmask.extensions.min.js" %>'
></script>
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/inputmask.numeric.extensions.min.js" %>'
></script>
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/jquery.inputmask.min.js" %>' ></script>
<!-- SmartGuide JS -->
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/custom.js" %>' ></script>
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/smartguide.min.js?v=7.0.0.0" %>'
></script>

```

- **jquery.form.min.js** – Unobtrusively upgrades HTML forms to use AJAX.
- **jquery-ui-1.10.4.custom.min.js** – Enables dynamic user interface interactions, effects and widgets.
- **bootstrap.min.js** – Brings Bootstrap's components to life (transitions, modals, dropdowns, tabs, tooltips and more).
- **Polyfills and plugins**– The default theme includes the most popular plugins but you can use different ones if you wish. See the [Adding plugins](#) section for more details.
- **custom.js** – We highly recommend that you place any custom JavaScript code and overwrites here to keep the original files intact and facilitate theme maintenance and upgrades.

- **smartguide.min.js** – This file contains all the bindings that bring the page and field actions defined in SmartGuide Designer to life.
14. If you made the changes listed above using an external editor and if the [alpinat.sgs.themes.SynchronizeMode](#) key in your web.xml is not set to *fromDisk*, you will then need to zip the theme contents and import the archive in SmartGuide Designer:
- Mouse over the theme name or click on the  icon next to it.
 - Select the **Edit theme** option.
 - Click on the **Modify** button next to the hyperlinked theme zip file.
 - Click on the **Choose File or Browse...** button to locate the archive on your hard drive.
 - Click on the **Save** button.

Using Materialize

The following instructions apply if the page that serves as the basis of the new theme uses the [Materialize](#) framework.

➤ To create a new Materialize-based theme:

1. Duplicate and rename the default theme (as detailed [above](#), except you will duplicate the *Materialize CSS Theme* instead).
2. Save the HTML source code of the web page that will serve as a template in a new file named *default.aspx*. Place this file under the *default* directory of the new theme (therefore overwriting the existing *default.aspx* file).
3. Set the page language at the top of the new *default.aspx* file, e.g.:

```
<%@ Page Language="C#" %>
```

4. Next, add a directive at the top of the *default.aspx* file to include the SmartGuide tag library (this is required on every template that will use the SmartGuide ASPX tags):

```
<%@ Register Tagprefix="Apn"
Namespace="Alpinat.SmartGuideServer.Controls"
Assembly="apnsgscontrols" %>
```

5. Add the following code to obtain the current locale:

```
<apn:locale runat="server" id="loc">
<% Context.Items["currentLocale"] =
loc.Current.GetValue(); %>
</apn:locale>
```

You can then use this variable throughout the *default.aspx* file. To reference the currently selected locale to declare the document language for example:

```
<html lang="<%=Context.Items["currentLocale"]%>">
```

6. Place the following meta tags within the *<head>* portion if you wish to populate them with content defined in SmartGuide Designer:

```
<meta name="application-name" content="<apn:control
runat="server" type="smartlet-name"><apn:value
runat="server"/></apn:control>">
<meta name="description" content="<apn:control
runat="server" type="smartlet-description"><apn:value
runat="server"/></apn:control>">
<meta name="keywords" content="<apn:control runat="server"
type="keyword"><apn:value runat="server"/></apn:control>"
/>
```

You can check the source code of the default theme for more examples of meta tags you can add.

7. Reference the following files in the *<head>* portion of *default.aspx* (these files are located in the *resources/css* folder of your custom theme):

```
<!-- Datable support -->
<link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/datatables.min.css" %>'
rel="stylesheet">
<!-- Autocomplete support -->
<link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/autocomplete.min.css" %>'
rel="stylesheet">
<!-- Materialize classes -->
<link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/materialize.min.css" %>'
rel="stylesheet">
<!-- SmartGuide CSS -->
<link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/smartguide.min.css" %>'
rel="stylesheet">
<link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/custom.css" %>' rel="stylesheet">
<!-- Load base jQuery --><!-- in case some fields output
html/js requiring jQuery -->
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/jquery.min.js" %>' ></script>
```

- **materialize.min.css** – You can either overwrite the default classes in the custom.css file and/or create a new materialize.min.css file by making changes to the source SCSS files and using a Sass compiler such as [Scout-App](#).
- **datatables.min.css** – To support table sorting and filtering.
- **autocomplete.min.css** – To turn drop-down fields that have an *autocomplete* class assigned to them into autocomplete controls.
- **smartguide.min.css** – This file contains formatting instructions for multiple page and form elements (section breadcrumb, alert messages, datatables, navigation buttons, etc.).
- **custom.css** – Of course, you can reference additional CSS files before or after the ones listed above. If you wish to overwrite certain definitions, we highly recommend placing your Bootstrap and SmartGuide overwrites in the custom.css file to keep the original files intact and facilitate maintenance and upgrades. Due to the nature of cascading style sheets, make sure this file comes last to ensure overwrites are taken into effect.
- **jquery.min.js** – jQuery is required to bring the dynamic interactions defined in SmartGuide Designer to life. We suggest that you import the jQuery library before the end of the <head> section since certain fields might output HTML or JavaScript code that rely on it.

8. Add the following code to the <body> tag to display the CSS classes and styles defined for a page in SmartGuide Designer:

```
class='<apn:control runat="server"
type="step"><apn:cssclass runat="server"/></apn:control>'
style='<apn:control runat="server"
type="step"><apn:cssstyle runat="server"/></apn:control>'
```

9. Place the following code where the page title should appear:

```
<apn:control runat="server" type="step"><apn:label
runat="server"/></apn:control>
```

10. Adjust the paths to images, JavaScript files and stylesheets:

- Use either an absolute URL to point to external resources or:
- Copy resources to the *resources* sub folders and modify the paths to source them from your new theme, for example:

```
<link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/mytheme.css"%>' rel="stylesheet">
```

11. Place the following code where the SmartGuide form elements should appear:

```

<form id='smartguide_<apn:control runat="server"
type="smartlet-code"><apn:value
runat="server"/></apn:control>' action="do.aspx"
method="post" enctype="multipart/form-data">
<div id="sglib"><%
Server.Execute(Page.TemplateSourceDirectory +
"/sglib.aspx"); %></div>
<!-- form validation -->
<% Server.Execute(Page.TemplateSourceDirectory + "/" +
"../controls/validation.aspx"); %>
<!--loop over page controls -->
<div id="sgControls">
    <% Server.Execute(Page.TemplateSourceDirectory + "/" +
"../controls/controls.aspx"); %>
</div>
<!-- navigation buttons -->
<div class="navigation">
    <% Server.Execute(Page.TemplateSourceDirectory + "/" +
"../controls/navigation.aspx"); %>
</div>
</form>

```

The SmartGuide form and the *sgControls* div are referenced by their ID in the *smartguide.js* file which adds dynamic behavior to the theme. The *sglib* div is required since version 7 of SmartGuide to support JavaScript field actions. More details can be found in the section [AJAX behavior of default theme](#). The *action*, *method* and *enctype* properties should be left as is. You can otherwise remove or overwrite the classes in the above code.

12. Reference the following files before the closing `</html>` tag (these files are located in the *resources/js* folder of your custom theme) to enable the dynamic features of SmartGuide (e.g. datatables, date picker, AJAX form submission, JavaScript actions on fields, etc.):


```

<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/jquery.form.min.js" %>' ></script>
<!-- JQuery UI -->
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/ui/jquery-ui-1.10.4.custom.min.js" %>'
></script>
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/materialize.min.js" %>' ></script>
<!-- HTML5 shim and Respond.js for IE8 support of HTML5
elements and media queries -->
<!--[if lt IE 9]>

```

```
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/html5shiv.min.js" %>' ></script>
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/respond.min.js" %>' ></script>
<![endif]-->
<!-- IE10 viewport hack for Surface/desktop Windows 8 bug
-->
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/ie10-viewport-bug-workaround.js" %>'
></script>
<!-- Datatables support -->
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/datatables.min.js" %>' ></script>
<!-- Autocomplete support -->
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/jquery.autocomplete.min.js" %>'
></script>
<!-- Inputmask support -->
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/inputmask.min.js" %>' ></script>
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/inputmask.extensions.min.js" %>'
></script>
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/inputmask.numeric.extensions.min.js" %>'
></script>
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/jquery.inputmask.min.js" %>' ></script>
<!-- SmartGuide JS -->
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/custom.js" %>' ></script>
<script src='<%=Page.TemplateSourceDirectory + "/" +
"../resources/js/smartguide.min.js?v=7.0.0.0" %>'
></script>
```

- **jquery.form.min.js** – Unobtrusively upgrades HTML forms to use AJAX.
- **jquery-ui-1.10.4.custom.min.js** – Enables dynamic user interface interactions, effects and widgets.
- **materialize.min.js** – Brings Materialize's components to life (transitions, modals, dropdowns, tabs, tooltips and more).
- **Polyfills and plugins**– The default theme includes the most popular plugins but you can use different ones if you wish. See the [Adding plugins](#) section for more details.

- **custom.js** – We highly recommend that you place any custom JavaScript code and overwrites here to keep the original files intact and facilitate theme maintenance and upgrades.
 - **smartguide.min.js** – This file contains all the bindings that bring the page and field actions defined in SmartGuide Designer to life.
- 13.** If you made the changes listed above using an external editor and if the [alpinat.sgs.themes.SynchronizeMode](#) key in your web.xml is not set to *fromDisk*, you will then need to zip the theme contents and import the archive in SmartGuide Designer:
- Mouse over the theme name or click on the  icon next to it.
 - Select the **Edit theme** option.
 - Click on the **Modify** button next to the hyperlinked theme zip file.
 - Click on the **Choose File or Browse...** button to locate the archive on your hard drive.
 - Click on the **Save** button.

Using another framework

While SmartGuide comes with a Bootstrap and a Materialize theme out-of-the-box, you are not limited to those frameworks when creating your own themes. If you choose this route, you should take the three following considerations into account.

1. Layout considerations

SmartGuide supports the Bootstrap 12-column grid system by default, so you will need to convert the layout attributes SmartGuide produces to match the grid system you wish to use.

For example, when we created the Materialize theme, we converted the default Bootstrap-compatible layout information (e.g. col-md-12) to Materialize-compatible code (e.g. col m12). This conversion had to be done in only one place, namely in the *col.aspx* file:

```
<%@ Page Language="C#" %>
<%@ Register Tagprefix="Apn"
Namespace="Alpinat.SmartGuideServer.Controls"
Assembly="apnsgscontrols" %>
<apn:control runat="server" id="control">
<%
    string layout = control.Current.getLayoutAttribute("all");
    layout = Regex.Replace(layout, "col-(\\D{2})-(offset-|push-|pull-)?(\\d)", "$2$1$3")
```



```

        .Replace("sm", "s")
        .Replace("md", "m")
        .Replace("lg", "l");
    %>
    <div class="col <%= layout %>">
        <% Server.Execute(Page.TemplateSourceDirectory +
"/controls.aspx"); %>
    </div>
</apn:control>

```

If you wish to use the [MUI framework](#) for example, the conversion code would be, again in *col.aspx*:

```

<%@ Page Language="C#" %>
<%@ Register Tagprefix="Apn"
Namespace="Alphinat.SmartGuideServer.Controls"
Assembly="apnsgscontrols" %>
<apn:control runat="server" id="control">
<%
    string layout = control.Current.getLayoutAttribute("all");
    layout = Regex.Replace(layout, "(col-*)", "mui-$1");
%>
<div class="col <%= layout %>">
    <% Server.Execute(Page.TemplateSourceDirectory +
"/controls.aspx"); %>
</div>
</apn:control>

```

2. Page and field actions

To bring the actions defined in SmartGuide Designer to life, be sure to include the following files in your custom theme:

- jquery.min.js
- jquery.form.min.js
- jquery-ui-1.10.4.custom.min.js
- datatables.min.js
- smartguide.min.js

These files can be found in the *resources/js* folder of one of the default themes (remember to only make changes to duplicates, as explained [above](#)).

3. CSS classes

To make it easy for Smartlet designers to assign CSS classes that are present in your theme to pages and fields, we recommend you create a class picker that will be available under the page and field *Appearance* tab. Please refer to the [Exposing CSS classes](#) section for more information.



Important: When creating new themes, please note that at a minimum, they must be composed of a *default* template and a *contextualhelp* template if you wish to display help associated to fields on distinct pages instead of in a modal dialog. Every template folder must contain a *default.aspx* file that specifies how to display the page. This file must at least contain the taglib directive and the form contents. Please refer to the [ASPX Tags Reference](#) section for more information on these and other SmartGuide ASPX tags.

You can have a closer look at the *default.aspx* file of the default themes for other sections of code to embed. For example you might want to add a page section breadcrumb trail (look for <!-- Breadcrumb trail-->), a progress bar (look for the div element with class *progress*), a language selector (look for the <apn:ifsmartletmultilingual> tag), etc. You can also have a look at the [cheat sheet](#) for sample code you can use in your project.

Adding plugins

In the examples above we have reused the plugins that ship with the default SmartGuide themes when creating a new theme. The default themes include the most popular plugins out-of-the-box but you may choose to use different ones if they better fit your needs. For example, if you wanted to replace the [Input Mask plugin](#) with another, you could start by removing the following files and references:

```
<!-- Inputmask support -->
<script src='<% =Page.TemplateSourceDirectory + "/" +
"../resources/js/inputmask.min.js" %>' ></script>
<script src='<% =Page.TemplateSourceDirectory + "/" +
"../resources/js/inputmask.extensions.min.js" %>' ></script>
<script src='<% =Page.TemplateSourceDirectory + "/" +
"../resources/js/inputmask.numeric.extensions.min.js" %>'
></script>
<script src='<% =Page.TemplateSourceDirectory + "/" +
"../resources/js/jquery.inputmask.min.js" %>' ></script>
```

You would then of course replace these files by those required by the plugin of your choosing. You would also need to modify or add any initialization/binding code in the *bindEvents* function of custom.js file to overwrite the default behavior:

```
$( 'input[data-mask]', sgRef.fm ).each( function( index ) {
    var $this = $( this );
    $this.inputmask( $this.attr( 'data-mask' ) );
} );
```

Creating a theme using external templates

Another option when developing themes is to embed the SmartGuide controls within your existing templates. This allows you to use external templates to handle the presentation layer of your Smartlets. Please note that when using this method, you still need to include a *default.aspx* page in your theme.

► To use an external template:

1. Add the **<apn:SmartGuide>** tag to the *default.aspx* file :

```
<%@ Register Tagprefix="Apn"
Namespace="Alphinat.SmartGuideServer.Controls"
Assembly="apnsgscontrols" %>
<html>
<apn:SmartGuide runat="server" config="/config/smartlet-
xmlengine-config.xml" smartletID="Smartlet_Code"
dispatchToTemplates="false"/>
```

Alternately, you could use the following code instead if you wish to insert the Smartlet code dynamically:

```
<%@ Register Tagprefix="Apn"
Namespace="Alphinat.SmartGuideServer.Controls"
Assembly="apnsgscontrols" %>
<html>
<% string smartlet =
(string)Context.Items["smartlet_code"]; %>
<apn:SmartGuide runat="server" config="/config/smartlet-
xmlengine-config.xml" smartletID="<%=smartlet%>"
dispatchToTemplates="false"/>
```

Please refer to the [<apn:SmartGuide/>](#) tag description for more information.

2. Modify the **smartguide** form action as shown below:

```
<form id="smartguide" action="do.aspx" default.aspx"
method="post" enctype="multipart/form-data">
```

3. Modify the paths to JavaScript, CSS and image files. For example:

```
@import url('/smartlets/aspx/default/resources/main.css');
```

4. Copy the default SmartGuide theme to your application, so that you can reference the controls to render the Smartlet pages and fields.
5. Include the references to the SmartGuide theme file you want to render. For example, to have the validations, controls, and navigation in your page include the following lines:

```
<!-- PRESENTATION OF ERROR MESSAGES -->
<% Server.Execute(Page.TemplateSourceDirectory + "/" +
"../controls/validation.aspx"); %>
<!-- MAIN LOOP OVER PAGE CONTROLS -->
<div id="sgControls" class="container-fluid">
    <% Server.Execute(Page.TemplateSourceDirectory + "/" +
"../controls/controls.aspx"); %>
</div>
<!-- NAVIGATION BUTTONS -->
<div class="navigation">
    <% Server.Execute(Page.TemplateSourceDirectory + "/" +
"../controls/navigation.aspx"); %>
</div>
```

6. If you wish to enable the PDF generation feature, add the following code after the *smartguide* form and before the closing `</body>` tag of the *default.aspx* file:

```
<apn:control type="generate" runat="server">
    <form id="pdf" target="_blank" method="post"
action="view.pdf.aspx">
        <apn:forEach id="control" runat="server">
            <% if (!control.Current.getName().Equals("pdf")) {%>
                <input type="hidden" name="<apn:name
runat='server' />" value="<apn:value runat='server' />" />
                <% } %>
            </apn:forEach>
            <input type="hidden" name="pdf" value="" />
        </form>
    </apn:control>
```

7. Modify the path in the *button.aspx* file if you wish to use a hyperlink to generate a PDF:

```
<a
href='/smartlets/default/default/view.pdf.aspx?cache=<%=
System.Guid.NewGuid().ToString() %>&pdf=<apn:name
runat="server"/>&interviewID=<apn:control runat="server"
type="interview-code"><apn:value
```

```
runat="server"/></apn:control>' target="_blank"><apn:value
runat="server"/></a>
```

8. To use a button instead of a hyperlink to generate a PDF, uncomment the pdf control in the *button.aspx* file:

```
<button type="submit" name="<apn:name runat='server'/">"
value="<apn:value runat='server'/">" class="btn
<apn:cssclass runat='server'/">"
style="<apn:controlattribute attr='style' runat='server'/">"
<apn:cssstyle runat='server'/">"
onclick="document.forms['pdf'].elements['pdf'].value='<apn:
name runat='server'/'>'; document.forms['pdf'].submit();
return false;" />
```

9. Comment the code used to generate a PDF via a hyperlink.
10. Repeat steps 6 to 8 to enable the XML generation feature.

For a concrete illustration of the above steps we will use a very simple example of a .NET application shown at

<http://asp.net-tutorials.com/basics/hello-world/>

Copy the sample HTML content shown on that page into a text file, then simply rename this file to "sample.aspx". Now drop this file in the smartlets application folder of SmartGuide. Assuming you are using port 8080 you should be able to view the sample page at:

<http://localhost:8080/smartlets/sample.aspx>

The original code for the *sample.aspx* page is:

```
<%
    HelloWorldLabel.Text = "Hello, world!";
%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
```

```

        <form id="form1" runat="server">
        <div>
            <asp:Label runat="server"
id="HelloWorldLabel"></asp:Label>
        </div>
        </form>
    </body>
</html>

```

The following declaration must now be added at the top of the *sample.aspx* file:

```
<%@ Page Language="c#" %>
```

After integrating the code as detailed at the beginning of the section, your page should look like this:

```

<%@ Page Language="c#" %>
<%@ Register Tagprefix="Apn"
Namespace="Alphinat.SmartGuideServer.Controls"
Assembly="apnsgscontrols" %>
<html>
<apn:SmartGuide runat="server" config="/config/smartlet-
xmlengine-config.xml" smartletID="Smartlet_Code"
dispatchToTemplates="false"/>
<%
    HelloWorldLabel.Text = "Hello, world!";
%>
<% Context.Items["currentLocale"] = ""; %>
<apn:locale runat="server" id="loc">
    <% Context.Items["currentLocale"] =
loc.Current.GetValue(); %>
</apn:locale>

<head runat="server">
    <title>Untitled Page</title>
    <!-- Bootstrap core CSS -->
    <link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/bootstrap.min.css"%>' rel="stylesheet">
    <!-- Bootstrap theme -->
    <link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/bootstrap-theme.min.css"%>'
rel="stylesheet">
    <!-- Datatable support -->

```

```

<link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/dataTables.bootstrap.css">'
rel="stylesheet">
<!-- Date widget support -->
<link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/bootstrap-datetimepicker.min.css">'
rel="stylesheet">
<!-- SmartGuide classes -->
<link href='<%=Page.TemplateSourceDirectory + "/" +
"../resources/css/main.css">' rel="stylesheet">
</head>
<body>
<div id="SmartGuideContent">
<form id="smartguide" action="sample.aspx" method="post"
enctype="multipart/form-data">
<div>
<asp:Label runat="server"
id="HelloWorldLabel"></asp:Label>

<!-- PRESENTATION OF ERROR MESSAGES -->
<% Server.Execute(Page.TemplateSourceDirectory + "/"
+ "../controls/validation.aspx"); %>
<!-- MAIN LOOP OVER PAGE CONTROLS -->
<div id="sgControls" class="container-fluid">
<% Server.Execute(Page.TemplateSourceDirectory
+ "/" + "../controls/controls.aspx"); %>
</div>
<!-- NAVIGATION BUTTONS -->
<div class="navigation">
<% Server.Execute(Page.TemplateSourceDirectory
+ "/" + "../controls/navigation.aspx"); %>
</div>
</div>
</form>
</div>
</body>
</html>

```



Note: The SmartGuide tags in this user guide and in the default SmartGuide theme use the 'apn' prefix, an abbreviation of Alphinat. The prefix can be changed when declaring the [tag library](#) but this will require replacing 'apn' with your custom identifier in all SmartGuide tags defined in the templates.

AJAX behavior of default theme

Modern web applications rely on JavaScript functionality to provide an improved user experience, including dynamic updates of HTML content.

The default theme in SmartGuide provides such functionality out of the box as there will often be dependencies between fields in a Smartlet. For example, one field might control the visibility of a group (show/hide). Or another field might be used in a numeric calculation updated in a “sum” field.

The implementation of this AJAX behavior in the default theme is done through the *resources/js/smartguide.min.js* file (also provided in non-modified format for easier reference) in combination with the jQuery library. A function called *ajaxProcess* can be invoked which will basically post the form data to the server, retrieve the content, and perform live updates in the current page.

The *ajaxProcess* function is invoked by binding events. For example, on a dropdown change or a text field blur event. You can look at the various handlers already defined in *smartguide.js* in the *bindEvents* function.

The decision to update part of the page is done through the use of a *data-eventtarget* attribute on form controls. If a form control has such an attribute (which is added dynamically when field relationships are defined in SmartGuide Designer), it contains an array of element ids which need to be updated when the current element is changed (or selected, etc depending on the binding applied on it).

So, for example, you could add the *data-eventtarget* attribute to an existing form element to enable AJAX dynamic processing of other form elements depending on it.

For the *ajaxProcess* function to work correctly, the following elements must exist in the default.aspx template or in one of the sub templates that it includes (see one of the default themes for an example):

- A <div> with an id of **sgControls**

This div should contain all the SmartGuide form controls.

- A <div> with an id of **alerts**

This div is used to display any error messages.

- A <div> with an id of **sglib**

This div should link to the sglib.aspx file or embed its contents directly (except the taglib declaration as its already defined at the top of default.aspx). In either

case, the name of the *smartletfields* variable should be left intact as it is also referenced in the smartguide.js file.

If you wish to make any changes to the default behavior, we highly recommend that you place them in the custom.js file to keep the original files intact and facilitate theme maintenance and upgrades. Otherwise, if you modify the smartguide.js file remember to either minify it, or modify default.aspx so that it points to the unminified version.

Theme localization

There are often cases where your applications must be available in multiple languages.

At the Smartlet level, translation is handled by exporting/importing an Excel spreadsheet containing all field labels and other information.

There is a similar mechanism at the theme level. You can also export and import an Excel spreadsheet to manage translations.

Looking at the default SmartGuide themes, many elements are available in both English and French, like the copyright message in the page footer for example. We could have checked the locale and provided different texts accordingly in the template itself, but that would have quickly added a lot of code to our pages. For example:

```
<%
    string copyrightMessage = "All rights reserved";
    if (currentLocale.Equals("fr")) {
        copyrightMessage = "Tous droits réservés" ;
    }
%>
<%= copyrightMessage %>
```

Instead, we made use of the SmartGuide *apn:localize* tag which simply takes the provided key and returns the corresponding text from the theme translation file:


```
<apn:localize runat="server" key="theme.text.copyright"/>
```

And placed the translations in the Excel file:

| | A | B | C | D |
|----|----------------------|----------------------|---------------------|----------------------|
| 1 | Identifier | Text | en | fr |
| 21 | theme.text.copyright | theme.text.copyright | All rights reserved | Tous droits réservés |

It makes the template code cleaner, provides a better separation of concerns and allows easily managing translations and adding any number of languages without having to know SmartGuide or having to modify your presentation templates. The exception to this rule is if you add new identifiers in the Excel spreadsheet. For them to be displayed, you will of course need to reference them in your template file(s). You can find more information about *apn:localize* in the [tags reference](#) section.

➤ **To export the Theme translation file:**

1. Mouse over the Theme you wish to localize or click the  icon next to its name.
2. Select the **Export the translation file** option.
3. Save the file to your hard drive.

The default language of the Smartlet is defined in column C of the *Text* sheet. Use columns D and further for each additional language or locale you wish to make your Smartlet available in. For example, to make your theme available in French in addition to English, write **fr** in cell D1 and provide translations in column D next to each entry, as in the example show above for the copyright.


You can also localize text to account for differences in distinct markets. For example, you could display a different text to users based on whether they are located in the US or in the UK. Locales are defined by concatenating the language code followed by a hyphen or underscore and the country code (e.g. EN-US, EN-GB...).



Important

- When you add a language, make sure you use the 2-letter ISO 639-1 code.
- When you add a locale, be sure to use the 2-letter ISO 3166-1 alpha-2 code.
- If you add identifiers to the spreadsheet, make sure that you also add references to them in your template file(s).

➤ **To import the Theme translation file:**

1. Mouse over the Theme you wish to localize or click the  icon next to its name.
2. Select the **Import translation file** option.
3. Locate the theme translation file on your hard drive.

4. Click the **Upload** button.

In addition to text translation, two more tags are available to manage localization. They are:

```
<apn:locale runat="server"/>
```

and

```
<apn:ifsmartletmultilingual runat="server"/>
```

The first one provides the current locale in use, for example "en". It can also provide the [full language](#), like "English".

The [second tag](#) allows you to perform specific processing, like showing a dropdown menu of available languages, when it is detected that a Smartlet has more than one language. You can refer to the default SmartGuide themes for examples of how language selectors can be implemented.

If a Smartlet is viewed in a language or locale that is not defined in the translation file, the text defined in the file's default language will be used to replace the apn:localize keys.

Modifying the layout file

The default SmartGuide themes are shipped with a JSON file that defines the options that are shown under the *Layout* tab when editing Smartlet fields. You will find the *bootstrap_layout_info.json* file under the *resources* folder.

The file contains the following attributes:

| Attribute | Description |
|-------------------------|--|
| name required | The name of the layout. Typically named after the framework being used. Example Values: bootstrap, angular material In the current version of SmartGuide, only the Bootstrap framework is fully supported. It is recommended to leave the default value until |

additional layout types are supported.

| | | | | | |
|----------------------------------|--|-------------------------|---|--------------------------|---|
| description optional | You can provide an optional textual description of the layout. | | | | |
| defaultdevice required | Must be one of the sizes listed under devices . For a mobile-first design, you would specify the smallest device size here, i.e. <i>xs</i> . By default, it is set to <i>md</i> meaning the page layout defined in SmartGuide Designer will be applied to medium-sized devices. | | | | |
| numberofcols required | <p>The number of columns the grid system is based on (12 for Bootstrap).</p> <p>This attribute will be used in upcoming releases when other layout types are supported. For now, you should leave the default value.</p> | | | | |
| devices required | <p>Each device size specified here will appear as a row under the field <i>Layout</i> tab. For example:</p> <p>Device type</p> <div><p>X-small devices (e.g. smartphones)</p><p>Small devices (e.g. tablets)</p><p>Medium devices (e.g. screens ≥ 992px)</p><p>Large devices (e.g. screens ≥ 1200px)</p></div> <p>Attributes</p> <table><tr><td>size required</td><td>Will be used to build the CSS classes that will be assigned to Smartlet fields based on selections made in the <i>Layout</i> tab.</td></tr><tr><td>label required</td><td>The device label will be displayed as shown in the image above. Since SmartGuide Designer is available in English and French, it is recommended to provide labels in both languages. If you only provide a label in English and a Designer user is working in French,</td></tr></table> | size required | Will be used to build the CSS classes that will be assigned to Smartlet fields based on selections made in the <i>Layout</i> tab. | label required | The device label will be displayed as shown in the image above. Since SmartGuide Designer is available in English and French, it is recommended to provide labels in both languages. If you only provide a label in English and a Designer user is working in French, |
| size required | Will be used to build the CSS classes that will be assigned to Smartlet fields based on selections made in the <i>Layout</i> tab. | | | | |
| label required | The device label will be displayed as shown in the image above. Since SmartGuide Designer is available in English and French, it is recommended to provide labels in both languages. If you only provide a label in English and a Designer user is working in French, | | | | |

no label will then be shown. Please note that if you provide a label in a single language, you still need to indicate the abbreviation of the language being used. E.g.:
`"label":{"en":"Small devices (e.g. tablets)"} }`

The *size* and *label* attributes are required for each device.

The device *size* value is used when rendering classes in the theme for the layout. For example, if an element is 6 columns wide for medium devices, the layout attribute returned for that element will be `"col-md-6"`.

attributes
semi-optional

Each attribute will appear as a column. E.g.:

| Hide ? | Cols ? | Offset ? |
|-------------------------------------|--------------------------------|----------------------|
| <input checked="" type="checkbox"/> | <input type="text"/> | <input type="text"/> |
| <input type="checkbox"/> | <input type="text"/> | <input type="text"/> |
| <input type="checkbox"/> | <input type="text" value="6"/> | <input type="text"/> |
| <input type="checkbox"/> | <input type="text"/> | <input type="text"/> |
| <input type="checkbox"/> | <input type="text"/> | <input type="text"/> |

Attributes

attribute
required

The *hidden*, *col*, *offset*, *push* and *pull* attributes are used by the default Bootstrap-based layout and should be left intact until additional layout types are supported in upcoming releases.

internalattr
required

Internal attribute.
The default value is *grid_width* and

| | |
|--|---|
| | should not be changed. |
| label required | The label is shown as the column header in the Designer interface. |
| tooltip optional | The provided tooltip will be displayed when mousing over the attribute's help icon. |
| values required | Must be set to one of the <i>attributesvalues</i> defined later in the file. |
| default optional | Allowed values: <ul style="list-style-type: none"> - true - false |
| | If <i>true</i> is specified, the corresponding box will be checked by default. |
| type required | Specifies how the attribute will be displayed in the interface. |
| | Allowed values: <ul style="list-style-type: none"> - checkbox - dropdown |
| attributesvalues semi-optional | Case-sensitive. Must correspond to the <i>values</i> previously defined in the file for <i>attributes</i> . Contains a number of values allowed for the specified type. |

Exposing CSS classes

To facilitate the use of CSS classes by the person developing Smartlets in SmartGuide Designer, it is possible to expose these classes through a JSON file in the theme.

The file is located under the *resources* folder of the theme and is named *theme_classes.json*. While the contents of the file can be changed, its name should not.

It is organized in sections and has the following structure:

```
{
  "Text formatting options": {
```

```

    "Label": {
      "en": "Text formatting options",
      "fr": "Options de formatage de texte"
    },
    "Description": {
      "en": "Apply one or more of the following text
formatting options",
      "fr": "Appliquer une ou plusieurs des options de
formatage de texte suivantes"
    },
    "List" : [
      {
        "class": "lead",
        "en": "Makes a paragraph stand out",
        "fr": "Fait ressortir un paragraphe"
      },
      {
        "class": "text-left",
        "en": "Left aligns text",
        "fr": "Aligne le texte à gauche"
      },
      ...
    ]
  },
  ...
}


```

In this example, we have a section called “Text formatting options”. The label and description follow. Then a list of classes exposed is presented as an array of JSON objects. Each class contains three properties, one for the class name and two for the label in English and French that will provide an explanation when presented in the Designer.

So in the example above, taken from the default theme, you will see an icon representing the class picker under the *Appearance* tab when editing a field:

[Properties](#)
[Value](#)
[Help](#)
[Validation](#)
[Appearance](#)
[Layout](#)

CSS class ?



CSS style ?

Value placement ?

- ☒ Default
- ☐ Place horizontally
- ☐ Place vertically

Field placement ?

- ☐ Default placement
- ☐ Place next to the previous field
- ☒ Place under the previous field

Field visibility ?

- ☒ Always display the field
- ☐ Never display the field
- ☐ Conditionally display the field

Clicking on it brings up the class picker:

Please choose below the CSS classes associated with the field.

Show
Text formatting options ▼

Filter classes:

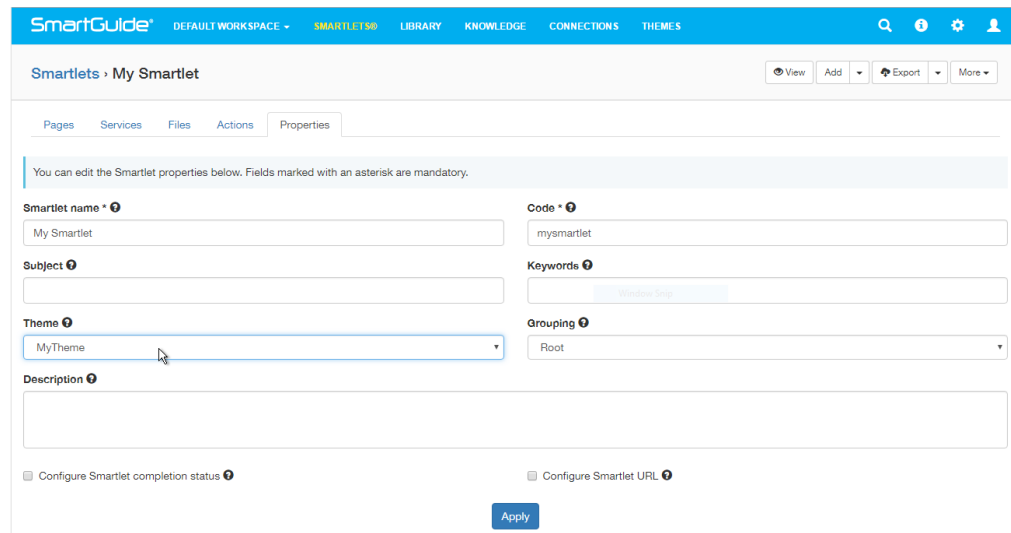
Apply one or more of the following text formatting options

| Class name | Description |
|--|-----------------------------|
| <input type="checkbox"/> <code>.lead</code> | Makes a paragraph stand out |
| <input type="checkbox"/> <code>.text-capitalize</code> | Capitalizes text |

Where we can see the two classes shown in the JSON file excerpt above.

Assigning themes to Smartlets

Once you've created a theme, you can associate it to a Smartlet under its *Properties* tab:



SmartGuide® DEFAULT WORKSPACE ▾ SMARTLET™ LIBRARY KNOWLEDGE CONNECTIONS THEMES

Smartlets: My Smartlet View Add Export More

Pages Services Files Actions Properties

You can edit the Smartlet properties below. Fields marked with an asterisk are mandatory.

Smartlet name * My Smartlet

Code * mysmartlet

Subject

Keywords Windows Soap

Theme MyTheme

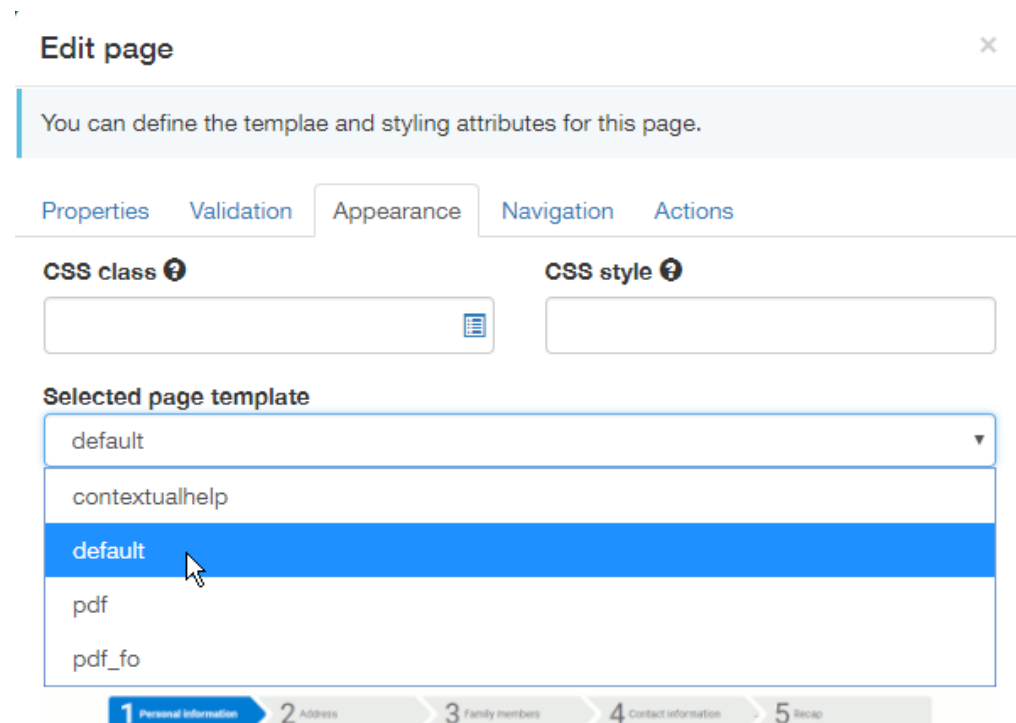
Grouping Root

Description

☐ Configure Smartlet completion status ☐ Configure Smartlet URL

Apply

When assigning a theme to a Smartlet, its *default* template is used to display every page unless you make another selection. To use a different template for a specific Smartlet page, simply select it under the *Appearance* tab when editing the page in SmartGuide Designer:



Edit page

You can define the template and styling attributes for this page.

Properties Validation Appearance Navigation Actions

CSS class

CSS style

Selected page template

default

contextualhelp

default

pdf

pdf_fo

1 Personal information 2 Address 3 Family members 4 Contact information 5 Recap

Creating custom controls

As we've seen so far, practically every field type that SmartGuide supports out-of-the-box has its own ASPX file (e.g. `input.aspx`, `select.aspx`...). So, to modify the look and feel of every text input field, you would only need to change a single file, namely the `input.aspx` file (and/or perhaps one of the CSS stylesheets depending on the type of changes you wish to make).

In addition, you can also extend the existing field types to create custom controls. The same basic steps can be followed to create any type of custom control:

- Create a custom control template based on the closest existing field type.
- Assign a specific class name to the custom control.
- Modify the control loop to use the custom template when a control contains the class name specified above.
- To make it easy for Smartlet designers to use these custom controls, you can create a sample control and place it in the library. You can also make the custom class available via the [class picker](#).

The following example shows how to create an autocomplete field for city names and is based on the *Remote JSONP datasource* sample shown at <http://jqueryui.com/autocomplete/#remote-jsonp>.

➤ To create a city autocomplete control:

1. Navigate to the **default/controls/** folder of your theme.
2. Duplicate the **input.aspx** file and name it `city.aspx`.
3. Add the following code right before the closing `</div>` tag of the new file:

```
<script>
    window.addEventListener('load', function() {
        $( "#<apn:name runat="server"/>"
    ).autocomplete({
        source: function( request, response ) {
            $.ajax({
                url:
                "http://gd.geobytes.com/AutoCompleteCity",
                dataType: "jsonp",
                data: {
                    q: request.term
                },
                success: function( data ) {
```

```

        response( data );
    }
    });
},
    minLength: 3
});
});
</script>

```

4. Retrieve a copy of the **jQuery UI** CSS file, place it in the *resources/css* folder and refer to it from the *default.aspx* page:

```

<!-- JQuery UI CSS -->
<link href=" <%=Page.TemplateSourceDirectory + "/" +
    "../resources/css/jquery-ui.css" rel="stylesheet">

```

5. Retrieve a copy of the **jQuery UI** JavaScript file, place it in the *resources/js* folder and refer to it from the *default.aspx* page:

```

<script src=" <%=Page.TemplateSourceDirectory + "/" +
    "../resources/js/jquery-ui.js"></script>

```

6. Declare an ID in the **controls.aspx** file to enable referring to each field's properties when looping through controls:

```

<apn:forEach id="ctrl" runat="server">

```

7. Modify the control loop to conditionally use the custom template when an input field has a CSS class of *city*:

```

<apn:whencontrol type="INPUT" runat="server">
    <% if (ctrl.Current.getCSSClass().IndexOf("city") != -1)
    {
        Server.Execute(Page.TemplateSourceDirectory + "/" +
            "../controls/city.aspx"); %>
    } else {
        Server.Execute(Page.TemplateSourceDirectory + "/" +
            "../controls/input.aspx"); %>    } %>
</apn:whencontrol>

```

Where **ctrl** refers to the variable name defined in step 6.

8. To test out the new control, add a text field to a Smartlet page and assign it a class of **city**.

Adding theme templates

If one or more of your Smartlet pages require a unique layout, special formatting or functionality, you should create a new template to keep your default template as generic as possible. You can then apply the new template to a specific Smartlet page or use it across multiple pages. The *default* template, which is automatically assigned to new Smartlet pages by default, contains the following files for instance:


- **default.gif** – This image is a screenshot of how a Smartlet page looks like using the *default* template. It will be shown under the page *Appearance* tab when selecting the template to use.
- **default.aspx** – This file contains the general formatting instructions for Smartlet pages that use the default template.
- **controls** – This sub folder contains a series of ASPXs that determine the appearance of Smartlet elements. For example, the *input.aspx* file contains display instructions for text fields, including where to place the label, the help icons and tooltips.

While new templates do not need to strictly follow this default structure, it is important to note that they must contain a *default.aspx* page at a minimum. In addition, it is recommended to include a screenshot of the template to facilitate its selection within SmartGuide Designer. And if you will be customizing multiple controls, you might want to group them in a folder in order to keep the *default.aspx* file as clean as possible.

Custom template example

In the following example, we will create a new template to display a Smartlet's first page differently than the rest of the pages. Since it's faster to start from an existing example, we will duplicate the *default* template of a copy of the default SmartGuide theme.

➤ To duplicate the default template:

1. First, make sure you're working on a copy of the default SmartGuide theme (as detailed [above](#)).
2. Click on the theme name to access its contents.
3. Mouse over the *default* template or click on the  icon next to it.
4. Select the **Duplicate** option.
5. Modify the **Template name**.

The name you enter will be used to create the directory on disk. If you enter *mytemplate* for example, a *mytemplate* folder will appear under the theme folder at the same level as the *default* template. The folder name must therefore be unique and should comply with the naming convention of the file system you are using. To ensure maximum portability, we recommend using short names that do not contain any accents or special characters.

6. Click on the **Upload** button if you wish to upload a screenshot for the template.

Although optional, screenshots are recommended to help Smartlet designers recognize templates.

7. Click on the **Browse...** button to locate the screenshot on your hard drive.

Screenshots can be in .gif, .jpg or .png format.

8. Optionally enter the theme **Description**.

We also recommend you provide a template description to facilitate its selection at the Smartlet page level.

9. Click on the **Save** button.

The new template, since it's based on the default template, contains its own *default.aspx* file and set of *controls* with instructions on how to display the Smartlet page and its elements. In our example, we wish to change how static images and static texts are displayed on the first page while displaying the other controls in the same manner as on every other page.

To do so, we simply need to modify the *image.aspx* and *statictext.aspx* files located under the *controls* folder of our custom template.

For the rest of the controls, we'll then reference the ones located in the default template instead of the custom one by adjusting the path in the *controls.aspx* file. For example:

```
<Apn:WhenControl runat="server" type="SUMMARY-SECTION">
    <% Server.Execute(Page.TemplateSourceDirectory + "/" +
    "../..../default/controls/summary.aspx"); %>
</Apn:WhenControl>
```

To keep our template folder as clean as possible, we can then delete any unused files (e.g. *summary.aspx*) located under the *controls* folder of our custom template.

To view the template in action, we simply have to [assign it](#) to the first page of a Smartlet.

Embedding the runtime server

If you have an existing .NET application consisting of a set of ASPX pages, libraries and tags, you can embed SmartGuide in your application.

You will first need to copy several files from SmartGuide Server. First, copy the required libraries for SmartGuide Server in *smartletsbin* to your *bin* directory. Finally copy the *config* and *data* directories located under *smartlets* to your application's *root* directory.

The last step is to bring all required configuration parameters and servlet from the *Web.config* file of SmartGuide Server to the *Web.config* file of your application. The minimum set of parameters is listed below:

```
<add key="apn_parameter_encoding" value="UTF-8" />
<add key="alpinat.sg.license.path" value="
C:/data/licence/smartguide.lic"/>
<add key="alpinat.sg.log4j.properties.path" value="apn-
log4j.xml" />
<!--if workspaces are enabled, set true below -->
<add key="apn.workspace.enabled" value="true" />
<add key="alpinat.sgs.page.autorefresh" value="true" />
<add key="com.alpinat.sgs.repository.Path"
value="D:\smartlets" />
<!-- This section is to allow integrated pipeline mode in
IIS 7.0 and later -->
  <system.webServer>
    <handlers>
      <add name="do.aspx" path="do.aspx" verb="*"
type="com.alpinat.interview.si.xml.servlet.XMLHttpHandler,
apn-sgs" />
    </handlers>
    <validation
validateIntegratedModeConfiguration="false" />
  </system.webServer>
```

The important parameters that must be customized above are the license path, the workspace flag, and the repository path for your Smartlets.

SmartGuide Server should now be integrated in your application directory structure. The next step will be to integrate the SmartGuide tags in your existing ASPX pages.

You can refer to the [Creating a theme using external templates](#) sub section for more details.

Multiple Smartlets per page

SmartGuide allows the possibility of presenting several Smartlets on a single page, making it possible to create a portal type application.

As a prerequisite, you must enable the multiple Smartlets mode in your *Web.config* configuration file:

```
<add key="com.alphinat.sgs.MultipleSmartletsEnabled"
value="true" />
```

Once this is done you will need to make use of the tag mode of SmartGuide to render multiple Smartlets on the page. Essentially each Smartlet on the page will have its own *apn:SmartGuide* tag.

One critical element that is required for multiple Smartlets mode is to post the Smartlet code with the form such that SmartGuide knows to which Smartlet the posted data belongs. If you fail to do so no processing will occur and an error message will appear in the SmartGuide logs.

Below is a typical example of what the body of the page would look like (we omit here the JavaScript includes at the bottom which you must keep):

```
<div class="topmenu">
  <apn:SmartGuide runat="server" config="/config/smartlet-
xmlengine-config.xml" smartletID="TopMenu"
dispatchToTemplates="false"/>
  <% Server.Execute(Page.TemplateSourceDirectory + "/" +
"../default/controls/controls.aspx"); %>
</div>

<div class="content">
  <apn:SmartGuide runat="server" config="/config/smartlet-
xmlengine-config.xml" smartletID="ListOfClients"
dispatchToTemplates="false"/>
  <form id='smartguide_<apn:control runat="server"
type="smartlet-code"><apn:value
```

```

runat="server"/></apn:control>' action="do.aspx"
method="post" enctype="multipart/form-data">
  <apn:API5 runat="server" id="sg5"/>
  <input type="hidden" name="com.alphinat.sgs.smartletcode"
value="<%
ISmartlet rootSmartlet = sg5.Smartlet;
while(rootSmartlet.getParentSubSmartletField()!=null){
  ISmartletField field =
  (ISmartletField)rootSmartlet.getParentSubSmartletField();
  rootSmartlet = field.getSmartlet();
}
Response.Write(rootSmartlet.getCode());
%>" />
<!-- PRESENTATION OF ERROR MESSAGES -->
<% Server.Execute(Page.TemplateSourceDirectory + "/" +
"../default/controls/validation.aspx"); %>

<!-- MAIN LOOP OVER PAGE CONTROLS -->
<div style="margin-right: 120px; margin-left: 120px; ">
<% Server.Execute(Page.TemplateSourceDirectory + "/" +
"../default/controls/controls.aspx"); %>
</div>
<div id="sgNavButtons">
  <% Server.Execute(Page.TemplateSourceDirectory + "/" +
"../default/controls/navigation.aspx"); %>
</div>
</form>
</div>

<div class="footer">
  <apn:SmartGuide runat="server" config="/config/smartlet-
xmlengine-config.xml" smartletID="Footer"
dispatchToTemplates="false"/>
  <% Server.Execute(Page.TemplateSourceDirectory + "/" +
"../default/controls/controls.aspx"); %>
</div>

```

In the example above we have 3 Smartlets. One to present some data as part of a menu header, one to present the main content, and one for the footer.

You will notice that the first and third Smartlets don't have an associated HTML form. We assume here that for these two Smartlets there are no input fields.

For the Smartlet rendered in the main content of the page we have a form, along with a piece of code to fetch the current Smartlet code. The loop is necessary because we might be within a sub-Smartlet and need to proceed up the hierarchy to get back to the parent Smartlet and it is the original parent Smartlet code that needs to be passed to execute and render that Smartlet (even when in a sub Smartlet context).

This chapter provides you with a quick reference guide on using the SmartGuide tags to display information regarding Smartlets, pages, fields and values.

Smartlet

Page

Field

Value

58

62

64

74

Smartlet

| To display the Smartlet... | Write this... |
|----------------------------|--|
| Name | <pre><apn:control runat="server" type="smartlet-name"> <apn:value runat="server"/> </apn:control></pre> <p>or</p> <pre><apn:control runat="server" type="smartlet-name" id="name"> <%= name.getValue() %> </apn:control></pre> |
| Code | <pre><apn:control runat="server" type="interview-code"> <apn:value runat="server"/> </apn:control></pre> <p>or</p> <pre><apn:control runat="server" type="interview-code" id="code"> <%= code.Current.getValue() %> </apn:control></pre> |
| Subject | <pre><apn:control runat="server"</pre> |

| | |
|--------------------|--|
| | <pre> type="title"> <apn:value runat="server"/> </apn:control> or <apn:control runat="server" type="title" id="title"> <%= title.Current.getValue() %> </apn:control> </pre> |
| Keywords | <pre> <apn:control runat="server" type="keyword"> <apn:value runat="server"/> </apn:control> or <apn:control runat="server" type="keyword" id="key"> <%= key.Current.getValue() %> </apn:control> </pre> |
| Theme | <pre> <apn:control runat="server" type="smartlet-theme"> <apn:value runat="server"/> </apn:control> or <apn:control runat="server" type="smartlet-theme" id="theme"> <%= theme.Current.getValue() %> </apn:control> </pre> |
| Grouping | n/a |
| Description | <pre> <apn:control runat="server" type="smartlet-description"> <apn:value runat="server"/> </apn:control> or <apn:control runat="server" </pre> |

| | |
|-------------------------------|--|
| | <pre> type="smartlet-description" id="description"> <%= description.Current.getValue() %> </apn:control> </pre> |
| Author | <pre> <apn:control runat="server" type="smartlet-author"> <apn:value runat="server"/> </apn:control> </pre> <p>or</p> <pre> <apn:control runat="server" type="smartlet-author" id="author"> <%= author.Current.getValue() %> </apn:control> </pre> |
| Last modification date | <pre> <apn:control type="smartlet- lastmodification"> <apn:value runat="server"/> </apn:control> </pre> <p>or</p> <pre> <apn:control runat="server" type="smartlet-lastmodification" id="lastmod"> <%= lastmod.Current.getValue() %> </apn:control> </pre> |
| Progress percentage | <pre> <apn:control runat="server" type="progress"> <apn:value runat="server"/> </apn:control> </pre> <p>or</p> <pre> <apn:control runat="server" type="progress" id="bar"> <%= bar.Current.getValue() %> </apn:control> </pre> |
| Page sections | <pre> <apn:forEach runat="server" </pre> |

```

items="sections" id="section">
<li><%=section.Current.getLabel()%><
/li>
</apn:forEach>
</ol>

```

Will display a numbered list containing the names of each Smartlet page section.

Page list

```

<apn:forEach runat="server"
items="global-navigation">
<a href='do.aspx?<apn:name
runat="server"/>=<apn:name
runat="server"/>'><apn:label
runat="server"/><a/> &nbsp;
</apn:forEach>

```

Will display a hyperlink to each Smartlet page using its title as defined in SmartGuide Designer followed by a space. To create a hyperlink back to the first page of a Smartlet for example, you could write:

```

<apn:forEach runat="server"
items="global-navigation"
id="indexNav">
    <% if(indexNav.getCount() == 1) {
%>
        <a href='do.aspx?<apn:name
runat="server"/>=<apn:name
runat="server"/>'>Start again<a/>
    <% } %>
</apn:forEach>

```

or

```

<apn:forEach runat="server"
items="global-navigation" begin="0"
end="0">
    <a href="do.aspx?<apn:name
runat="server"/>=<apn:name
runat="server"/>">Start again<a/>
</apn:forEach>

```

| | |
|--------------------------------|--|
| Current language | <pre><apn:locale runat="server" id="loc"> <h1><%= loc.getLabel() + " (" + loc.getValue() + ")" %></h1> </apn:locale></pre> |
| All available languages | <pre><select> <apn:forEach runat="server" id="locale" items="languages"> <option <%= (locale.Current.getValue().Equals (currentLocale) ? "selected" : "") %> value="<%=locale.Current.getValue()% "><%=locale.Current.getLabel()%></o ption> </apn:forEach> </select></pre> |

Page

| To display the page... | Write this... |
|------------------------|--|
| Name | <pre><apn:control runat="server" type="step"> <apn:code runat="server"/> </apn:control></pre> <p>or</p> <pre><apn:control runat="server" type="step" id="step"> <%= step.Current.getCode() %> </apn:control></pre> |
| Title | <pre><apn:control runat="server" type="step"> <apn:label runat="server"/> </apn:control></pre> <p>or</p> <pre><apn:control runat="server" type="step" id="step"> <%= step.Current.getLabel() %> </apn:control></pre> |

| | |
|---|--|
| Section | <pre> <apn:control runat="server" type="section"> <apn:label runat="server"/> </apn:control> or <apn:control runat="server" type="section" id="section"> <%= section.Current.getLabel() %> </apn:control> </pre> |
| CSS classes | <pre> <apn:control runat="server" type="step"> <apn:cssclass runat="server"/> </apn:control> </pre> |
| CSS styles | <pre> <apn:control runat="server" type="step"> <apn:cssstyle runat="server"/> </apn:control> </pre> |
| Description | n/a |
| Template | n/a |
| List of elements | <pre> <form id="smartguide" action="do.aspx" method="post" enctype="multipart/form- data"> <apn:forEach runat="server"> See the list of properties under elements below </apn:forEach> </form> </pre> |
| Elements as a JavaScript associative array | <pre> <script> var smartletfields = <apn:jsfields runat="server"/>; </script> </pre> <p>You can find an example of this in the default themes (see the <code>sglib.aspx</code> file which is included in the default template). The <code>smartletfields</code> variable is referenced in the <code>smartguide.js</code> file to bring the field actions defined in SmartGuide Designer to life.</p> |
| Element error messages | <pre> <apn:forEach runat="server" items="alert-controls" id="ctrl"> <%= </pre> |

```
ctrl.Current.getAlert() %></span><br />
</apn:forEach>
```

Will return the message entered in SmartGuide Designer under the *Validation* tab when the element validation fails. Each message would be displayed in red on a new line.

Navigation buttons

```
<apn:control runat="server"
type="previous">
  <input type="submit" name='<apn:name
runat="server"/>' value='<apn:label
runat="server"/>'>
</apn:control>
```

Will display a *Previous page* button on the page if that navigation option is set in SmartGuide Designer.

```
<apn:control runat="server" type="next">
  <input type="submit" name='<apn:name
runat="server"/>' value='<apn:label
runat="server"/>'>
</apn:control>
```

Will display a *Next page* button on the page if that navigation option is set in SmartGuide Designer.

Field

| To display the field... | Write this... |
|-------------------------|--|
| ID | <pre><apn:name runat="server"/></pre> <p>or</p> <pre><apn:control runat="server" id="ctrl"> <%= ctrl.Current.getName() %> </apn:control></pre> |
| Name | <pre><apn:code runat="server"/></pre> <p>or</p> |

| | |
|---------------------------|--|
| | <pre><apn:control runat="server" id="ctrl"> <%= ctrl.Current.getCode() %> </apn:control></pre> |
| Type | <pre><apn:control runat="server" id="ctrl"> <%= ctrl.Current.getType() %> </apn:control></pre> <p>The field type is represented by a numerical ID. Refer to the List of control types for a complete list.</p> <p>The field type is represented by a numerical ID. Refer to the List of control types for a complete list.</p> |
| Label | <pre><apn:label runat="server"/></pre> <p>or</p> <pre><apn:control runat="server" id="ctrl"> <%= ctrl.Current.getLabel() %> </apn:control></pre> |
| Encryption status | Use the boolean <i>isEncrypted()</i> with the SmartGuide API. |
| Persistence status | Use the boolean <i>isPersistent()</i> with the SmartGuide API. |
| Read-only status | <pre><apn:controlattribute runat="server" attr="readonly"/></pre> <p>or</p> <pre><apn:control runat="server" id="ctrl"> <%= ctrl.Current.getAttribute("readonly") %> </apn:control></pre> <p>Returns <i>readonly</i> if the element has been marked as such in SmartGuide Designer. To check whether an element is editable or not, you can use the following code for example:</p> <pre><% if (ctrl.getAttribute("readonly").Equals("r</pre> |

```
eadonly")) {%>
  <%-- Do something --%>
<% } %>
```

Alternatively, you can use the boolean *isReadOnly()* with the SmartGuide API.

Custom attributes

```
<apn:metadata runat="server"/>
```

Will display the attributes and values defined under the field *Data* tab.

Value

```
<apn:value runat="server"/>
```

or

```
<apn:control runat="server" id="ctrl">
  <%= ctrl.Current.GetValue() %>
</apn:control>
```

To display the value without interpreting any HTML code that it might contain, you can use:

```
<apn:value runat="server"
tohtml='true' />
```

Tooltip

```
<apn:controlattribute runat="server"
attr="title"/>
```

or

```
<apn:control runat="server" id="ctrl">
  <%= ctrl.Current.GetAttribute("title")
%>
</apn:control>
```

Detailed help

```
<apn:help runat="server"/>
```

or

```
<apn:control runat="server" id="ctrl">
  <%= ctrl.Current.GetHelp() %>
</apn:control>
```

The detailed help can either be a URL or text. The

`<apn:ifhelplink>` and `<apn:ifnohelplink>` helps distinguish between both types.

To access the detailed help of a field in a new browser window by clicking a hyperlink for example, you could use the following code:

```
<apn:ifhelplink runat="server">
  <a href='<apn:help runat="server"/>'
target="_blank">[Help]</a>
</apn:ifhelplink>
<apn:ifnohelplink runat="server">
  <a href='?<apn:helpid
runat="server"/>=<apn:helpid
runat="server"/>'
target="_blank">[Help]</a>
</apn:ifnohelplink>
```

Or to access the detailed help by clicking on a help icon, you could write:

```
<apn:ifhelplink runat="server">
  <a href='<apn:help runat="server"/>'
target="_blank"></a>
</apn:ifhelplink>
<apn:ifnohelplink runat="server">
  <input type="image" value='<apn:helpid
runat="server"/>' name='<apn:helpid
runat="server"/>' src="img/help.gif" />
</apn:ifnohelplink>
```

Mandatory status

```
<%= ctrl.Current.isRequired() %>
```

Returns *true* or *false*. You can also use the `apn:ifcontrolrequired` tag. To display a bold asterisk next to mandatory fields for example, you could write:

```
<apn:ifcontrolrequired runat="server">
  <span style="font-
weight:bold">*</span>
</apn:ifcontrolrequired>
```

You can also use:

```
<apn:control runat="server" id="ctrl">
  <%= ctrl.Current.getFieldType() %>
</apn:control>
```

or

```
<apn:control runat="server" id="ctrl">
  <%= ctrl.Current.getStyle() %>
</apn:control>
```

which will return either *required* or an empty string.

Validation

```
<apn:ifcontrolvalid runat="server">
  // content goes here
</apn:ifcontrolvalid>

<apn:ifnotcontrolvalid runat="server">
  <span style="color:red"><%=
ctrl.Current.getAlert() %></span><br>
</apn:ifnotcontrolvalid>
```

These tags allow you to display different content whether the entered field value is valid or not. In this example, if there is a validation error, the message that was entered in SmartGuide Designer will be displayed in red.

Validation format

```
<apn:controlattribute runat="server"
attr="format"/>

or

<apn:control runat="server" id="ctrl">
  <%=
ctrl.Current.getAttribute("format") %>
</apn:control>
```

Returns the validation format of the current field if defined (e.g. yyyy-mm-dd for a date field).

CSS Class

```
<apn:cssclass runat="server"/>
```

or

```
<apn:control runat="server" id="ctrl">
  <%= ctrl.Current.getCSSClass() %>
</apn:control>
```

Will return the value of the *CSS class* entered under the field *Appearance* tab.

CSS Style

```
<apn:cssstyle runat="server"/>
```

or

```
<apn:control runat="server" id="ctrl">
  <%= ctrl.Current.getCSSStyle() %>
</apn:control>
```

Will return the value of the *CSS style* entered under the field *Appearance* tab.

Width

(applies only to image fields)

```
<apn:controlattribute runat="server"
attr="width"/>
```

or

```
<apn:control runat="server" id="ctrl">
  <%= ctrl.Current.getAttribute("width")
%>
</apn:control>
```

Will return the field width in pixels.

Height

(applies only to image fields)

```
<apn:controlattribute runat="server"
attr="height"/>
```

or

```
<apn:control runat="server" id="ctrl">
  <%=
ctrl.Current.getAttribute("height") %>
</apn:control>
```

Will return the field height in pixels.

| | |
|--|--|
| Alt (applies only to image fields) | <pre><apn:controlattribute runat="server" attr="alt"/></pre> <p>or</p> <pre><apn:control runat="server" id="ctrl"> <%= ctrl.Current.getAttribute("alt") %> </apn:control></pre> |
| Size | <pre><apn:controlattribute runat="server" attr="size"/></pre> <p>or</p> <pre><apn:control runat="server" id="ctrl"> <%= ctrl.Current.getAttribute("size") %> </apn:control></pre> <p>Will return a value if the field width is specified in characters. See Style below if the width is specified in pixels.</p> |
| Prefix | <pre><apn:controlattribute runat="server" attr="prefix"/></pre> <p>or</p> <pre><apn:control runat="server" id="ctrl"> <%= ctrl.Current.getAttribute("prefix") %> </apn:control></pre> |
| Suffix | <pre><apn:controlattribute runat="server" attr="suffix"/></pre> <p>or</p> <pre><apn:control runat="server" id="ctrl"> <%= ctrl.Current.getAttribute("suffix") %> </apn:control></pre> |
| Style | <pre><apn:controlattribute runat="server"</pre> |

```
attr="style"/>
```

or

```
<apn:control runat="server" id="ctrl">
  <%= ctrl.Current.GetAttribute("style")
%>
</apn:control>
```

Will return *width:150px* if the field width is entered in pixels (150 in this example). See *Size* above if the width is entered in characters.

Will return *height:50px* if the field height is entered in pixels (50 in this example).

Will return *width:150px;height:50px* if both the width and height are entered in pixels.

Will return *visibility:hidden* if the field is a hidden field.

Class
(applies only to buttons)

```
<apn:controlattribute runat="server"
attr="class"/>
```

or

```
<apn:control runat="server" id="ctrl">
  <%= ctrl.Current.GetAttribute("class")
%>
</apn:control>
```

Maxlength

```
<apn:controlattribute runat="server"
attr="maxlength"/>
```

or

```
<apn:control runat="server" id="ctrl">
  <%=
ctrl.Current.GetAttribute("maxlength")
%>
</apn:control>
```

Will return the value of the maximum length entered under the field *Validation* tab.

| | |
|--|--|
| Rows (applies only to textareas) | <pre><apn:controlattribute runat="server" attr="rows"/></pre> <p>or</p> <pre><apn:control runat="server" id="ctrl"> <%= ctrl.Current.GetAttribute("rows") %></pre> <pre></apn:control></pre> <p>Will return a value if the field height is specified in rows.</p> |
| Cols (applies only to textareas) | <pre><apn:controlattribute runat="server" attr="cols"/></pre> <p>or</p> <pre><apn:control runat="server" id="ctrl"> <%= ctrl.Current.GetAttribute("cols") %></pre> <pre></apn:control></pre> <p>Will return a value if the field height is specified in columns.</p> |
| Visibility | See <i>Style</i> above. |
| Placement | <pre><apn:control runat="server" id="ctrl"> <% = ctrl.Current.GetPlacement() %></pre> <pre></apn:control></pre> <p>Returns <i>next</i> if the option <i>Place next to the previous field</i> is selected as the field placement under the <i>Appearance</i> tab.</p> <p>Returns <i>under</i> if the option <i>Place under the previous field</i> is selected as the field placement under the <i>Appearance</i> tab.</p> |
| Visible | <pre><apn:controlattribute attr="visible" runat="server"/></pre> <p>or</p> |


```
<apn:control var="ctrl" runat="server">
  <%=
ctrl.Current.GetAttribute("visible") %>
</apn:control>
```

Returns *true* if the field is visible and *false* if it is not. This functionality is new as of version 5.5 and allows to put a placeholder in the page (and empty div for example) for non-visible fields.

Eventtarget

```
<apn:controlattribute attr="eventtarget"
runat="server"/>
```

or

```
<apn:control var="ctrl" runat="server">
  <%=
ctrl.Current.GetAttribute("eventtarget")
%>
</apn:control>
```

Returns a javascript array of field ids which are affected by the current field. Affected means that these other fields have a value, condition, visibility or validation involving the current field.

Eventsource

```
<apn:controlattribute attr="eventsource"
runat="server"/>
```

or

```
<apn:control var="ctrl" runat="server">
  <%=
ctrl.Current.GetAttribute("eventsource")
%>
</apn:control>
```

Returns a javascript array of field ids which affect the current field. Affect means that the current field has a value, condition, visibility or validation involving these other fields.

Render mode for repeat groups

```
<apn:controlattribute runat="server"
attr="rendermode"/>
```

or

```
<apn:control runat="server" id="ctrl">
  <%=
ctrl.Current.getAttribute("rendermode")
%>
</apn:control>
```

Returns the computed render mode for a repeat group by SmartGuide. There are two return values possible: "table" and "block".

The table mode is detected when all fields are on a single row and none of the fields is a group. We assume the required way to display data is like a table.

The block mode is detected when fields span more than one row or a group is involved. In such a case the add/remove buttons appear and the end user can add/remove instance of the repeat group.

Value

| To display the value... | Write this... |
|-------------------------|---|
| ID | <pre><apn:controlattribute runat="server" attr='id' /></pre> <p>To display the list of values for a checkbox type field, you can use the following code in an <i><apn:forEach></i> loop:</p> <pre><input type="checkbox" name='<apn:name runat="server"/>' id='<apn:controlattribute runat="server" attr="id"/>' value='<apn:value runat="server"/>' /></pre> |

Which would result in the following sample output when the page is viewed:

```
<input type="checkbox"
name="d_1222415202965" id="l_466"
value="Value 1" />
```

```
<input type="checkbox"
name="d_1222415202965" id="l_467"
value="Value 2" />
```

Since we can uniquely identify field values, it's then possible to make use of the html <label> tag to allow selecting a radio button or check box by also clicking on its label. For example:

```
<label for='<apn:controlattribute
runat="server" attr="id"/>'><apn:label
runat="server"/></label>
```

```
<input type="checkbox" name= '<apn:name
runat="server"/>' id=
'<apn:controlattribute runat="server"
attr="id"/>' value= '<apn:value
runat="server"/>' />
```

Label

```
<apn:label runat="server"/>
```

or

```
<apn:control runat="server" id="ctrl2">
  <%= ctrl2.Current.GetLabel() %>
</apn:control>
```

Value

```
<apn:value runat="server"/>
```

or

```
<apn:control runat="server" id="ctrl2">
  <%= ctrl2.Current.GetValue() %>
</apn:control>
```

Tooltip

```
<apn:controlattribute      runat="server"
attr="title"/>
```

or

```
<apn:control runat="server" id="ctrl2">
```

| | |
|----------------------|--|
| | <pre> <%= ctrl2.Current.getAttribute("title") %> </apn:control> </pre> |
| Detailed help | <pre> <apn:help runat="server"/> </pre> <p>or</p> <pre> <apn:control runat="server" id="ctrl2"> <%= ctrl2.Current.getHelp () %> </apn:control> </pre> |
| Layout | <pre> <apn:control runat="server" id="ctrl"> <%= ctrl.Current.getChoiceLayout() %> </apn:control> </pre> <p>Returns either <i>horizontally</i> or <i>vertically</i>.</p> <p>Returns <i>horizontally</i> if the option <i>Place horizontally</i> is selected as the value placement under the <i>Appearance</i> tab of check box and radio button type fields.</p> <p>Returns <i>vertically</i> if the option <i>Place vertically</i> is selected as the value placement under the <i>Appearance</i> tab of check box and radio button type fields.</p> |

PDF templates

You can also design templates that will be used to create a PDF output of any number of Smartlet pages.

Guidelines

- Start your template with the following declaration if you wish to support accents and other special characters:

```
<?xml version="1.0" encoding="UTF-8"?>
```
- Use inline css.
- Do not use the *display* property.
- Use tables for layout purposes (positioning through the *float* property is not supported while the *position* property has limited support).
- Use the *background-color* property to set the background color.

ASPX Tags Reference

5

The ASPX tags described within this document may be placed within a standalone ASPX page or within ASPX pages included within the primary ASPX page, with the exception of the `<apn:whencontrol>` and `<apn:otherwise>` tags, which must be nested within a `<apn:choosecontrol>` tag located within the same ASPX page. This allows you to reuse components throughout the various templates that may be required for a given Smartlet project.

Register directive

In order to use customized ASPX tags, a register directive must be included at the beginning of each ASPX template.

The register directive to use the SmartGuide tags follows the following syntax:

```
<%@ Register Tagprefix="Apn"
    Namespace="Alphinat.SmartGuideServer.Controls"
    Assembly="apnsgscontrols" %>
```

All attributes of the register directive above are required. The Namespace and assembly specify which assembly to load. The Tagprefix attribute defines the string which will qualify the tag name (separated by a colon).

In the above example, the prefix *apn* is the tag name prefix that must be used in front of each of the tags appearing within the SmartGuide ASPX tag library. For instance, if the tag defined is *tagname*, and the prefix attribute is *apn*, the actual tag name would be *apn:tagname*. An ASPX tag can be written using one of two different formats independently of whether it has a body or not.

Syntax 1

The tag contains a body:

```
<apn:tagname runat="server" >
    // body content
</apn:tagname>
```

Syntax 2

The tag does not contain a body:

```
<apn:tagname runat="server" />
```

<apn:control>

The control tag is used to create a context, if a context does not already exist. A context is used to define a section within the ASPX page where information about the current Smartlet control is accessible via the specified variable for scripting purposes.

Syntax

```
<apn:control runat="server" [id="varName"] [type="type"]
[name="name"]>
    // body content
</apn:control>
```

Body Content

ASPX.

Attributes

| <i>Name</i> | <i>Type</i> | <i>Description</i> |
|--------------------|--------------------|---|
| id | String | Is the name of the exported tag body scoped variable for the current item of the collection. The variable's type is: com.alphinat.xmlengine.interview.aspx.Control. |
| type | String | Allows obtaining a control with a specific type. Please refer to The Type attribute section for a list of valid type attribute values. |
| name | String | Allows obtaining a control with a specific name. Control names are defined using SmartGuide Designer. |

Description

Establishes a context within which specific control information can be obtained.

- Specifying a value for the *name* attribute allows obtaining a named control. Control names are defined using SmartGuide Designer.
- Specifying a value for the *type* attribute allows to obtain a control with a specific type as well as to access specialized controls.

- Defining a scoped variable using the *id* attribute will allow for the usage of ASPX scripting.

Example

In the following example, the value of the *title* control found in the Smartlet page will be displayed. Please refer to [The Type attribute](#) section for a list of all valid type attribute values.

```
<apn:control runat="server" type="title" id="titleVar">
  <h1><%= titleVar.Current.getValue() %></h1>
</apn:control>
```

Observe that we cannot directly use the "id" defined to get the current SmartGuide object. It is necessary to use the "Current" member of the control id. This is true for all SmartGuide tags.

Methods

The control's methods are accessed through the *id* attribute name as in the example above.

| Name | Description |
|--|--|
| boolean containsValue(String) | Returns <i>true</i> if the specified text matches the controls current value. |
| String getAlert() | Returns the error messages associated with the control when validation fails. Can also be set with <code>setAlert(String)</code> . |
| String getAttribute(String) | Returns the specified control's attribute. Possible values for the attribute are: alt class code cols cssclass cssstyle eventsources eventtarget format help |

helpid
 height
 id
 label
 maxlength
 name
 prefix
 readonly
 rendermode
 rows
 selected
 size
 src
 style
 suffix
 title
 tooltip
 value
 visible
 width

If any other attribute is passed as a parameter to this function, then its associated value will be returned assuming it was previously set with the `.data(key,value)` API function.

| | |
|---|--|
| String getChoiceLayout() | Returns the choice layout. Can also be set with <code>setChoiceLayout(String)</code> . |
| Class getClass() | Returns the control class. |
| String getCode() | Returns the control name. Can also be set with <code>setCode(String)</code> . |
| String getCSSClass() | Returns the CSS class. Can also be set with <code>setCSSClass(String)</code> . |
| String getCSSStyle() | Returns the CSS style set under the control <i>Appearance</i> tab in SmartGuide Designer. It can also be set with <code>setCSSStyle(String)</code> . |
| String getFieldType() | Returns <i>required</i> when the field is required. Can also be set with <code>setFieldType(String)</code> . |
| String getHelp() | Returns the detailed help. Can also be set with |

`setHelp(String)`.

| | |
|-----------------------------------|--|
| String getHelpId() | Returns an internal identifier for the help text. Can also be set with <code>setHelpId(String)</code> . |
| String getHTMLValue() | Returns HTML encoded form of the control value. |
| String getLabel() | Returns the label. Can also be set with <code>setLabel(String)</code> . |
| String getLabelId() | Returns an internal identifier for the label. Can also be set with <code>setLabelId(String)</code> . |
| String getMetaData(String) | Returns the value associated to the attribute name passed in parameter. Can also be set with <code>setMetaData(String attr, String val)</code> |
| String getName() | Returns the name of the generated HTML form element. Can also be set with <code>setName(String)</code> . |
| ControlInfo getNext() | Returns the control following the current one in the page. |
| ControlInfo getParent() | Returns the parent control to the current one. |
| String getPlacement() | Returns the placement of the control as defined under the <i>Appearance</i> tab in SmartGuide Designer. Can also be set with <code>setPlacement(String)</code> . |
| ControlInfo getPrevious() | Returns the control preceding the current one in the page. |
| String getSelectedLabel() | Returns the label of the currently selected element. Applicable to dropdown list and radio button fields only. |
| String getStyle() | Returns <i>required</i> when the field is required. Can also be set with <code>setStyle(String)</code> . |
| String getTooltip() | Returns the tooltip associated with the control. Can also be set with <code>setTooltip(String)</code> . |
| int getType() | Returns the field type (please refer to the List of control types for more information). Can also be set with <code>setType(int)</code> . |
| String getValue() | Returns the control value. Can also be set with <code>setValue(String)</code> . |
| boolean isHelpLink() | Returns <i>true</i> if the help is a url (instead of text). |

| | |
|-----------------------------|--|
| boolean isRequired() | Returns <i>true</i> if the control is mandatory. |
| boolean isValid() | Returns <i>true</i> if the control is validated. |
| setHelpLink(boolean) | Specifies whether the help is a url or not. |
| setRequired(boolean) | Specifies whether the control is mandatory or not. |
| setValid(boolean) | Specifies whether the control is validated or not. |

<apn:forEach>

Repeats its nested body content over a collection of Smartlet controls.

Syntax

```
<apn:forEach runat="server" [id="varName"]
[items="collection"] [begin="begin"] [end="end"]
[step="step"]>
    // body content
</apn:forEach>
```

Body Content

ASPX. As long as there are items to iterate over, the tag's body content will be processed by the ASPX container and written to the current `HtmlTextWriter`.

Attributes

| <i>Name</i> | <i>Type</i> | <i>Description</i> |
|--------------------|--------------------|--|
| id | String | Name of the exported scoped variable for the current item of the iteration. The variable's type is: <code>com.alphinat.xmlengine.interview.aspx.Control</code> . |

| | | |
|-------|--------|--|
| items | String | <p>Specifies the collection of items to be iterated over. Currently, the three supported values are:</p> <ul style="list-style-type: none"> ▪ alert-controls – allows to iterate over all the controls that contain alert-type messages that appear in the event that an error has occurred. ▪ global-navigation – allows iterating over the navigation links to all the pages of a Smartlet. ▪ sections – allows iterating over the page sections of a Smartlet. ▪ locales – allows iterating over the supported languages of the current Smartlet. |
| begin | int | The <i>forEach</i> tag will begin the iteration at the Smartlet control located at the index specified by the <i>begin</i> attribute. The first control of the collection has an index of 0 (zero). |
| end | int | The <i>forEach</i> tag will end the iteration at the index specified by the <i>end</i> attribute. |
| step | int | The <i>forEach</i> tag will iterate over every <i>step</i> item of the Smartlet control collection. Example: if step has a value of 2, the <i>forEach</i> tag will iterate over every 2, 4, 6... control in the collection. |

Constraints

- If specified, the index set by the *begin* attribute must be ≥ 0 .
- If specified, the index set by the *end* attribute must be $\geq \textit{begin}$.
- If specified, the index set by the *step* attribute must be ≥ 1 .

Description

The element over which the *forEach* iterates is based on the context within which the *forEach* tag is placed. For instance:

- If the *forEach* tag is placed inside a page, it will iterate over all the first-level controls of the Smartlet page.
- If the *forEach* tag is placed inside a group, it will iterate over all the first-level controls found in the group.
- If the *forEach* tag is placed inside a repeat, it will iterate over all the first-level controls inside the repeat.
- If the *forEach* tag is placed inside a select element, it will iterate over all the items inside the select.

Example 1

The *forEach* tag iterates over the controls on the page and displays the value of each control.

```
<apn:forEach runat="server" id="ctrl">
  <input type="text" value="<%= ctrl.Current.getValue() %>" >
</apn:forEach>
```

Example 2

The *forEach* tag iterates over the alerts on the page and displays these alerts in red.

```
<apn:forEach runat="server" items="alert-controls"
id="ctrl">
  <font color="red"> <%= ctrl.Current.getAlert() %>
</font><br>
</apn:forEach>
```

Note

You can get the HTML id of the control with `ctrl.Current.getName()`. This allows you to add HTML markup as required directly on the control in the page.

Example 3

The *forEach* tag iterates over the navigation controls of a Smartlet and renders these controls as hyperlinks separated by a space.

```
<apn:forEach runat="server" items="global-navigation">
  <a href='do.aspx?<apn:name runat="server"/>=<apn:name
runat="server"/>'><apn:label runat="server"/></a>
</apn:forEach>
```

Note

`<apn:name/>` extracts the control's identifier.
`<apn:label/>` extracts the page's title.

Example 4

The *forEach* tag iterates over the navigation controls of a Smartlet and renders these controls as buttons.

```
<apn:forEach runat="server" items="global-navigation">
```

```

    <apn:control>
        <input type="submit" name='<apn:name runat="server"/>'
value='<apn:label runat="server"/>'>
    </apn:control>
</apn:forEach>

```

Example 5

The *forEach* tag iterates over the locales supported by the Smartlet and renders these locales as options in a dropdown menu:

```

<select>
<apn:forEach runat="server" id="locale" items="languages">
    <option <%= (locale.Current.getValue().Equals(currentLocale)
? "selected" : "") %>
value="<%= locale.Current.getValue() %>"><%= locale.Current.get
Label() %></option>
</apn:forEach>
</select>

```

Note

You can also use tags to retrieve information about the available languages:

<apn:value/> the two letter locale identifier.
 <apn:label/> the current language.

Methods

The *forEach* control exposes a number of methods when accessed through the *id* attribute name as follows (without the Current member):

```

<apn:forEach runat="server" id="ctrl2">
    <%= ctrl2.Count %>
</apn:forEach>

```

| Name | Description |
|----------------------|--|
| boolean First | Returns <i>true</i> if the current iteration is the first one. |
| boolean Last | Returns <i>true</i> if the current iteration is the last one. |
| int Count | Returns the number of iterations performed so far. |

| | |
|------------------|---|
| | Depends on the <i>begin</i> , <i>end</i> , and <i>step</i> attributes above. |
| int Index | Returns the index (zero-based) of the current iteration. |
| int Begin | Returns the value of the <i>begin</i> attribute if specified, otherwise <i>null</i> . |
| int End | Returns the value of the <i>end</i> attribute if specified, otherwise <i>null</i> . |
| int Step | Returns the value of the <i>step</i> attribute if specified, otherwise <i>null</i> . |

<apn:label/>

Displays the label associated with the current control. This tag is functionally equivalent to invoking the `com.alphinat.xmlengine.interview.aspx.Control's getLabel()` method when using scripting variables.

Syntax

```
<apn:label runat="server"/>
```

Body Content

Empty

Attributes

None

Constraints

None

Description

The control's label is written to the current `HtmlTextWriter`.

Example

```
<apn:control runat="server">
  <apn:label runat="server"/>:    <input type="text"
value="" name="">
</apn:control>
```

<apn:SmartGuide/>

This tag is used when running SGS in tag mode (as opposed to using it as a servlet). It allows using external templates when viewing Smartlets. A number of parameters can be configured, including the name of the Smartlet to execute.

Syntax

```
<apn:SmartGuide runat="server" [smartletID="smartlet code"]  
[path=""] [dispatchToTemplate="true/false"]  
[resetSmartlet="true/false"]  
[configurationScopeAttribute="attrName"] [roles="path"]/>
```

Body Content

Empty

Attributes

| <i>Name</i> | <i>Type</i> | <i>Description</i> |
|-----------------------------|--------------------|---|
| config | String | Specifies the location of the smartlet-xmlengine-config.xml configuration file when not available in its default location (under config of smartlets). |
| smartletID | String | Specifies the code of the Smartlet that will be executed. |
| dispatchToTemplate | boolean | If set to <i>true</i> , the specified template to use in the Smartlet will be used. By default it is <i>false</i> , meaning that the current page where the tag appears will do the processing. |
| resetSmartlet | boolean | Set to <i>true</i> to reinitialise the Smartlet to its initial state. |
| configurationScopeAttribute | String | Specifies the name of a page variable containing a dictionary of configuration parameters, bypassing the need to specify them in the web.config. See the examples below. |

Constraints

Must be declared before any other SmartGuide control in the aspx page.

Description

The main SmartGuide control is executed, allowing any other control on the page to execute.

Example 1

Here we simply specify a Smartlet code to use:

```
<apn:SmartGuide runat="server" smartletID="mysmartlet"
dispatchToTemplates="false" />
```

Example 2

Here we define a Map in the aspx page and store it in the page context:

```
<% System.Collections.Hashtable sgsConfig = new
System.Collections.Hashtable();
sgsConfig["config"] = "/config/Smartlet-xmlengine-
config.xml";
sgsConfig["dispatchToTemplates"] = "false";
Context.Items["sgdConfig"] = sgdConfig;
%>
<apn:smartguide runat="server" smartletID="mySmartlet"
configurationScopeAttribute="sgsConfig" />
```

<apn:translate/>

Displays the dictionary value corresponding to the provided key in the body, or if not specified by the key provided by the enclosing control value, label or name.

Note

This tag is provided for backward compatibility with older themes, and is now considered deprecated. Users should use the new <apn:localize/> tag for translation purposes.

Syntax

```
<apn:translate runat="server" resource="resName"
[scope="session/application/request"]
[defaultValue="defaultVal"]>
// key
</apn:translate>
```

Body Content

String that will serve as a key for lookup in a dictionary. If not specified, a lookup is performed on the enclosing tag properties.

Attributes

| <i>Name</i> | <i>Type</i> | <i>Description</i> |
|--------------|-------------|---|
| scope | String | Provide the scope where the dictionary is made available. Value can be <i>session</i> (default value), <i>application</i> , or <i>request</i> . |
| resource | String | Name of the dictionary to look up in the specified scope. |
| defaultValue | String | Default value to display in case the key lookup doesn't return anything. |

Constraints

None.

Description

Provides a simple way to get multilingual support for a Smartlet by providing a translation mechanism using a dictionary to get values corresponding to a key given in the body of the tag. If the body is empty then the enclosing tag properties are used as keys in the following order depending on the tag:

- **Static text** – Value, label, and name. If none of these provided a translation then the value and the label will be directly used as the translated value.
- **Group** – Label, name. If none of these provided a translation then the label will be directly used as the translated value.
- **Option** – Value, label. If none of these provided a translation then the label will be directly used as the translated value.
- **All other** – Name, label. If none of these provided a translation then the label will be directly used as the translated value.

Example

Here we assume the existence of a session dictionary called *resDict*. We also assume that a control's label has been populated with a key:

```
<apn:translate runat="server" scope="session"
resource="resDict">
  <apn:label runat="server"/>
</apn:translate>
```

<apn:localize/>

Displays the dictionary value corresponding to the key provided in the body, or through the key attribute.

Syntax

```
<apn:localize runat="server" [defaultValue="defaultVal"]  
[id="varName"] [key="customkey"] [keyPrefix="customprefix"]  
[keySuffix="customsuffix"]>  
// key  
</apn:translate>
```

Body Content

String that will serve as a key for lookup in a dictionary. If not specified, a lookup is performed on the enclosing tag properties.

Attributes

| <i>Name</i> | <i>Type</i> | <i>Description</i> |
|--------------------|--------------------|--|
| defaultValue | String | Default value to display in case the key lookup doesn't return anything. |
| key | String | The key on which to perform a lookup. |
| keyPrefix | String | Prefix that will be prepended to the key before the lookup is performed. |
| keySuffix | String | Suffix that will be appended to the key before the lookup is performed. |

Constraints

None.

Description

Provides a way to access localized custom resources, like the ones found in theme files.

Example

To get the copyright notice for the default theme:

```
<apn:localize runat="server" key="theme.text.copyright"/>
```

<apn:API5/>

Makes a number of objects available in the page giving access to the full API.

Syntax

```
<apn:API5 runat="server" [id="varName"] />
```

Body Content

Empty

Attributes

| <i>Name</i> | <i>Type</i> | <i>Description</i> |
|--------------------|--------------------|--------------------------------------|
| id | String | Name of the exported scoped variable |

Constraints

Since these objects live in the context of a page execution, some actions, like adding a field, will have no effect.

Description

Provides a point of entry to the full API, bypassing the limitations of the tag set.

Methods

The *API5* control exposes a number of methods when accessed through the *id* attribute name as follows:

```
<apn:API5 runat="server" id="ctrl">  
  <%= ctrl.getSmartlet().getName() %>  
</apn:forEach>
```

| Name | Description |
|---------------------------------------|---|
| ISmartlet getSmartlet() | Returns the <i>ISmartlet</i> object that can be used in scriptlets. |
| ISmartletPage getCurrentPage() | Returns the <i>ISmartletPage</i> object <i>that can be used in scriptlets</i> . |
| IServiceContext | Returns the <i>IServiceContext</i> object that can be used in |

| | |
|-------------------------|--|
| getContext() | scriptlets. |
| IEnvironment | Returns the <i>IEnvironment</i> object that can be used in |
| getEnvironment() | scriptlets. |

Example

Here we fetch a field's string value to display the current logged on user. Notice that, unlike making use of the « apn:control » tag, we don't need to be on a page that has the « hidUserId » field to get access to it. We can simply use the findFieldByName API function. For example:

```
<apn:api5 runat="server" id="sg"/>
<%
String userId =
sg.getSmartlet().findFieldByName("hidUserId").getString();
out.print("Current logged on user is: "+userId);
%>
```

<apn:clearAPI5/>

Clears the variables set by the "apn:API5" tag.

Syntax

```
<apn:clearAPI5 runat="server"/>
```

Body Content

Empty

Attributes

None

Constraints

None

Description

The current context and associated variables are nullified. This statement should always be used whenever the "apn:API5" tag is used to avoid leaks. And if used, then it should always be at the end of the page to avoid problems rendering elements on the page.

Example

```
<apn:clearAPI5 runat="server">
```

<apn:cssclass/>

Displays the class associated with the current control. This tag is functionally equivalent to invoking the `com.alphinat.xmlengine.interview.aspx.Control's getCSSClass()` method when using scripting variables.

Syntax

```
<apn:cssclass runat="server"/>
```

Body Content

Empty

Attributes

None

Constraints

None

Description

The control's CSS Class is written to the current `HtmlTextWriter`.

Example

```
<apn:control runat="server">
  <input type="text" value="" name="" class='<apn:cssclass
runat="server"/>'>
</apn:control>
```

<apn:cssstyle/>

Displays the style associated with the current control. This tag is functionally equivalent to invoking the `com.alphinat.xmlengine.interview.aspx.Control's getCSSStyle()` method when using scripting variables.

Syntax

```
<apn:cssstyle runat="server"/>
```

Body Content

Empty

Attributes

None

Constraints

None

Description

The control's CSS Style is written to the current HtmlTextWriter.

Example

```
<apn:control runat="server">
  <input type="text" value="" name="" style='<apn:cssstyle
runat="server"/>'>
</apn:control>
```

<apn:name/>

Displays the id associated with the current control. This tag is functionally equivalent to invoking the `com.alphinat.xmlengine.interview.aspx.Control`'s *getName()* method when using scripting variables.

Syntax

```
<apn:name runat="server"/>
```

Body Content

Empty

Attributes

None

Constraints

None

Description

The control's name is written to the current `HtmlTextWriter`.

Example

```
<apn:control runat="server">
  <input type="text" value="" name='<apn:name
runat="server"/>'>
</apn:control>
```

<apn:value/>

Displays the value associated with the current control. This tag is functionally equivalent to invoking the `com.alphinat.xmlengine.interview.aspx.Control.GetValue()` method when using scripting variables.

Syntax

```
<apn:value runat="server" [tohtml="true/false"]/>
```

Body Content

Empty

Attributes

| <i>Name</i> | <i>Type</i> | <i>Description</i> |
|--------------------|--------------------|---|
| tohtml | String | Determines whether the value must be html encoded. This prevents javascript code injection for example. If not specified <i>false</i> is assumed. |

Constraints

None

Description

The control's value is written to the current `HtmlTextWriter`.

Example

```
<apn:control runat="server">
  <input type="text" value='<apn:value runat="server"/>'
name='<apn:name runat="server"/>'>
</apn:control>
```

<apn:code/>

Displays the code associated with the current control. The code represents the control's name in SmartGuide Designer. This tag is functionally equivalent to invoking the `com.alphinat.xmlengine.interview.aspx.Control`'s *getCode()* method when using scripting variables.

Syntax

```
<apn:code runat="server"/>
```

Body Content

Empty

Attributes

None

Constraints

None

Description

The control's code is written to the current `HtmlTextWriter`.

Example

In this example, we iterate over all the controls on the page. For each control, we wish to display its code and label separated by a dash. Both represent the control's name and label as defined in SmartGuide Designer.

```
<apn:forEach runat="server">
  <apn:code runat="server"/> - <apn:label runat="server"/>
</br>
</apn:forEach>
```

<apn:controlattribute/>

Displays the value of the attribute associated with the current control if the value specified within the *attr* attribute exists. This tag is functionally equivalent to invoking the `com.alphinat.xmlengine.interview.aspx.Control`'s *getAttribute(String)* method when using scripting variables.

Syntax

```
<apn:controlattribute runat="server" attr="attrName"/>
```

Body Content

Empty

Attributes

| <i>Name</i> | <i>Type</i> | <i>Description</i> |
|--------------------|--------------------|--|
| attr | String | Allows obtaining the value of a specific attribute within the current control. |

Constraints

The *attr* attribute must be specified.

Description

The attribute value of the control is written to the current `HtmlTextWriter`.

Example

The following highlighted code retrieves the value of the current control's size attribute:

```
<apn:control runat="server">
  <apn:label runat="server"/>: <input type="text"
value='<apn:value runat="server"/>' name='<apn:name
runat="server"/>' size='<apn:controlattribute attr='size'
runat="server"/>'>
</apn:control>
```

<apn:ifcontrol>

Evaluates its body content if the expression specified within the *name* or *type* attribute refers to a valid control.

Syntax1

The body content will be evaluated if a control with the specified name exists within the current Smartlet page. The *name* attribute is associated with the current control's name as defined in SmartGuide Designer.

```
<apn:ifcontrol runat="server" name="nameCondition">
    // body content
</apn:ifcontrol>
```

Syntax2

The body content will be evaluated if the current control with the specified type exists within the current Smartlet page. Please refer to [The Type attribute](#) section for a list of all supported type values.

```
<apn:ifcontrol runat="server" type="typeCondition">
    // body content
</apn:ifcontrol>
```

Body Content

ASPX.

Attributes

| <i>Name</i> | <i>Type</i> | <i>Description</i> |
|--------------------|--------------------|---|
| name | String | Allows referring to a specific control by name. Control names are defined using SmartGuide Designer. |
| type | String | Allows referring to a specific control by type. Please refer to The Type attribute section for a list of valid type attribute values. |

Constraints

At least one of the two attributes (*name* or *type*) must be specified.

Description

If the *name* or *type* attributes refer to an existing control, the body content is evaluated by the ASPX container and the result is outputted to the current `HtmlTextWriter`:

- The *name* attribute specifies that the tag's body content will be evaluated if a control with the specified name exists within the current Smartlet page.
- The *type* attribute specifies that the tag's body content will be evaluated if a control with the specified type exists within the current Smartlet page.

Example 1

The following code displays the help and label associated with the current control if this control is of type *staticText*:

```
<apn:ifcontrol runat="server" type="staticText">
<% Server.Execute(Page.TemplateSourceDirectory +
"/help.aspx"); %>
    <h2><apn:label runat="server"/></h2>
</apn:ifcontrol>
```

Example 2

In this example, we iterate over all the controls defined in the context. For each control of type *select1*, the label and value associated with the current control as defined in SmartGuide Designer are displayed:

```
<apn:forEach runat="server">
    <apn:ifcontrol runat="server" type="select1">
        <apn:label runat="server"/>: <apn:value
runat="server"/><br/>
    </apn:ifcontrol>
</apn:forEach>
```

<apn:ifnotcontrol>

Evaluates its body content if the expression specified within the *name* or *type* attribute does not refer to a valid control or control type. This tag produces a result that is exactly opposite to that of the *apn:ifcontrol* tag.

Syntax 1

The body content will be evaluated if a control with the specified name does not exist within the current Smartlet page.

```
<apn:ifnotcontrol runat="server" name="nameCondition">
    // body content
</apn:ifnotcontrol>
```

Syntax 2

The body content will be evaluated if the current control with the specified type does not exist within the current Smartlet page.

```
<apn:ifnotcontrol runat="server" type="typeCondition">
    // body content
```

```
</apn:ifnotcontrol>
```

Body Content

ASPX.

Attributes

| <i>Name</i> | <i>Type</i> | <i>Description</i> |
|--------------------|--------------------|---|
| name | String | Allows referring to a specific control by name. Control names are defined using SmartGuide Designer. |
| type | String | Allows referring to a specific control by type. Please refer to The Type attribute section for a list of valid type attribute values. |

Constraints

At least one of the two attributes (*name* or *type*) must be specified.

Description

If the *name* or *type* attributes do not refer to an existing control, the body content is evaluated by the ASPX container and the result is outputted to the current `HtmlTextWriter`:

- The *name* attribute specifies that the tag's body content will be evaluated if a control with the specified name does not exist within the current Smartlet page.
- The *type* attribute specifies that the tag's body content will be evaluated if a control with the specified type does not exist within the current Smartlet page.

Example

The following code displays the help and label associated with the current control if this control is not of type *staticText*:

```
<apn:ifnotcontrol runat="server" type="staticText">
  <% Server.Execute(Page.TemplateSourceDirectory +
"/help.aspx"); %>
  <h4><apn:label runat="server"/></h4>
</apn:ifnotcontrol>
```

<apn:ifcontrolattribute>

Evaluates its body content if the control attributes specified within the *attr* attribute are defined.

Syntax1

The body content will be evaluated if the specified attribute exists within the current control. The *attr* attribute is associated with any attribute that could be assigned to the current control.

```
<apn:control runat="server">
  <apn:ifcontrolattribute runat="server" attr="attribute">
    // body content
  </apn:ifcontrolattribute>
</apn:control>
```

Syntax2

The body content will be evaluated if the specified attributes in the current control, evaluated as an expression, return true. The AND, OR, NOT operators are supported as well as parenthesis and wilcards for attribute names.

```
<apn:control runat="server">
  <apn:ifcontrolattribute runat="server" attr="expression">
    // body content
  </apn:ifcontrolattribute>
</apn:control>
```

Body Content

ASPX.

Attributes

| <i>Name</i> | <i>Type</i> | <i>Description</i> |
|--------------------|--------------------|--|
| attr | String | Allows referring to a specific attribute, or an expression composed of attributes. |

Constraints

The attr attribute must be specified.

Description

If the *attr* attribute is defined for the current control, the body content is evaluated by the ASPX container and the result is outputted to the current `HtmlTextWriter`:

- The *attr* attribute specifies that the tag's body content will be evaluated if the control attribute is defined for the current control. In the case of an expression

composed of many attribute, the tag's body content will be evaluated if the expression evaluates to true.

Example 1

The body content will be evaluated if the prefix has been assigned a non-empty value for the control.

```
<apn:control runat="server">
  <apn:ifcontrolattribute runat="server" attr="prefix">
    // body content
  </apn:ifcontrolattribute>
</apn:control>
```

Example 2

The body content will be evaluated if either the prefix or the suffix have a non empty value assigned.

```
<apn:control runat="server">
  <apn:ifcontrolattribute runat="server" attr="prefix OR
suffix">
    // body content
  </apn:ifcontrolattribute>
</apn:control>
```

<apn:ifnotcontrolattribute>

Evaluates its body content if the control attribute specified within the *attr* attribute is not defined.

Syntax1

The body content will be evaluated if the specified attribute doesn't exist within the current control. The *attr* attribute is associated with any attribute that could be assigned to the current control.

```
<apn:control runat="server">
  <apn:ifnotcontrolattribute runat="server"
attr="attribute">
    // body content
  </apn:ifnotcontrolattribute>
</apn:control>
```

Body Content

ASPX.

Attributes

| <i>Name</i> | <i>Type</i> | <i>Description</i> |
|--------------------|--------------------|---|
| attr | String | Allows referring to a specific attribute. |

Constraints

The attr attribute must be specified.

Description

If the *attr* attribute is not defined for the current control, the body content is evaluated by the ASPX container and the result is outputted to the current `HtmlTextWriter`:

- The *attr* attribute specifies that the tag's body content will be evaluated if the control attribute is not defined for the current control.

Example 1

The body content will be evaluated if the prefix has not been assigned a value for the control.

```
<apn:control runat="server">
  <apn:ifnotcontrolattribute runat="server" attr="prefix">
    // body content
  </apn:ifnotcontrolattribute>
</apn:control>
```

This tag is functionally equivalent to `<apn:ifcontrolattribute runat="server" attr="NOT prefix"/>`.

<apn:controllayoutattribute/>

Displays the value of the layout attribute associated with the current control if the value specified within the *attr* attribute exists.

Syntax

```
<apn:controllayoutattribute runat="server" attr="attrName"/>
```


Body Content

Empty

Attributes

| <i>Name</i> | <i>Type</i> | <i>Description</i> |
|--------------------|--------------------|---|
| attr | String | Allows obtaining the value of a specific layout attribute within the current control. |

Constraints

The *attr* attribute must be specified.

Description

The attribute value of the control is written to the current `HtmlTextWriter`.

Example

The following highlighted code retrieves all layout attributes of the current control:

```
<apn:control runat="server">
  <div class="<apn:controllayoutattribute attr='all'/'>">
    ...
  </div>
</apn:control>
```

This control is used to get layout information like "col-md-6", or "xs-hidden". In addition to "all", one or many specific attributes can be specified, like "col,push" which could result in "col-md-4 push-md-2" if such layout attributes were specified for the control.

<apn:ifsmartletmultilingual>

Evaluates its body content if the Smartlet contains more than one language.

Syntax

```
<apn:ifsmartletmultilingual runat="server">
  // body content
</apn:ifsmartletmultilingual>
```

Body Content

ASPX. The body content is processed by the ASPX container and the result is written to the current `HtmlTextWriter`.

Attributes

None.

Constraints

None.

Description

If the Smartlet has more than one language defined, the body content is evaluated by the ASPX container and the result is output to the current HtmlTextWriter. This is useful when you want to display an element like a dropdown menu with choice of language only when required.

<apn:locale>

The locale tag is used to get the current locale in use by the Smartlet.

Syntax

```
<apn:locale runat="server" [var="varName"]>
    // body content
</apn:locale>
```

Body Content

ASPX.

Attributes

| <i>Name</i> | <i>Type</i> | <i>Description</i> |
|-------------|-------------|---|
| var | String | Is the name of the exported tag body scoped variable for the current locale of the Smartlet. The variable's type is: com.alphinat.xmlengine.interview.aspx.Control. |

Description

Establishes a context within which locale information can be obtained.

- Defining a scoped variable using the *var* attribute will allow for the usage of ASPX scripting.

Example

In the following example, the locale and language name are returned:

```
<apn:locale runat="server" id="loc">
  <h1><%= loc.Current.getLabel() + " (" +
loc.Current.getValue() + ")" %></h1>
</apn:locale>
```

The above could return:

```
<h1>English (en)</h1>
```

<apn:choosecontrol>

Provides the context of mutually exclusive conditional execution.

Syntax

```
<apn:choosecontrol runat="server">
  // body content
  // <apn:whencontrol> and <apn:otherwise> sub-tags
</apn:choosecontrol>
```

Body Content

ASPX. The body content is processed by the ASPX container (at most one of the nested actions will be processed) and written to the current `HtmlTextWriter`.

Attributes

None

Constraints

The body of the `<apn:choosecontrol>` can only contain the following:

- 1 or more `<apn:whencontrol>` sub-tags.
- 0 or 1 `<apn:otherwise>` sub-tag.
- If used, the `<apn:otherwise>` sub-tag must be the last action nested within the `<apn:choosecontrol>` sub-tags.
- White spaces may appear anywhere around the `<apn:whencontrol>` and `<apn:otherwise>` sub-tags.

Description

The `<apn:choosecontrol>` tag evaluates the body of the first `<apn:whencontrol>` tag whose test attributes (*name* or *type*) refer to an existing control according to the mechanisms described within the `<apn:ifcontrol>` section of the document. If none of the `<apn:whencontrol>` actions' conditions evaluate to true, the `<apn:otherwise>` action's body content is evaluated.

Example

```
<apn:choosecontrol runat="server">
  <apn:whencontrol runat="server" type="INPUT">
    <% Server.Execute(Page.TemplateSourceDirectory +
"/input.aspx"); %>
  </apn:whencontrol>
  <apn:whencontrol runat="server" type="TEXTAREA">
    <% Server.Execute(Page.TemplateSourceDirectory +
"/textarea.aspx"); %>
  </apn:whencontrol>
</apn:choosecontrol>
```

<apn:whencontrol>

Represents an alternative within the `<apn:choosecontrol>` action. Apart from the mutually-exclusive nature of the tag, its behaviour is identical to that of the `<apn:ifcontrol>` tag.

Syntax1

The body content will be evaluated if a control with the specified name exists within the current Smartlet page. The *name* attribute is associated with the current control's name as defined in SmartGuide Designer.

```
<apn:whencontrol runat="server" name="nameCondition">
  // body content
</apn:whencontrol>
```

Syntax2

The body content will be evaluated if the current control with the specified type exists within the current Smartlet page. Please refer to [The Type attribute](#) section for a list of all supported type values.

```
<apn:whencontrol runat="server" type="typeCondition">
```

```
// body content
</apn:whencontrol>
```

Body Content

ASPX. The first `<apn:whencontrol>` action within the `<apn:choosecontrol>` action that evaluates to true will have the ASPX container process its body content and then write it to the current `HtmlTextWriter`.

Attributes

| Name | Type | Description |
|------|--------|---|
| name | String | Allows referring to a specific control by name. Control names are defined using SmartGuide Designer. |
| type | String | Allows referring to a specific control by type. Please refer to The Type attribute section for a list of valid type attribute values. |

Constraints

- One of the two attributes (*name* or *type*) must be specified.
- Must have the `<apn:choosecontrol>` as the immediate parent.
- Must appear before the `<apn:otherwise>` action that belongs to the same `<apn:choosecontrol>`.

Example

```
<apn:choosecontrol runat="server">
  <apn:whencontrol runat="server" type="INPUT">
    <% Server.Execute(Page.TemplateSourceDirectory +
"/input.aspx"); %>
  </apn:whencontrol>
  <apn:whencontrol type="TEXTAREA">
    <% Server.Execute(Page.TemplateSourceDirectory +
"/textarea.aspx"); %>
  </apn:whencontrol>
</apn:choosecontrol>
```

<apn:otherwise>

Represents the last alternative within the `<apn:choose>` action. If none of the conditions defined by the `<apn:whencontrol>` actions are met, the `<apn:otherwise>` action is executed.

Syntax

```
<apn:otherwise runat="server">
    // body content
</apn:otherwise>
```

Body Content

ASPX. After all the *<apn:whencontrol>* actions are evaluated and none is set to true, the ASPX container processes the body content of the *<apn:otherwise>* action and writes its content to the current `HtmlTextWriter`.

Attributes

None.

Constraints

- Must be inside the *<apn:choosecontrol>* as the immediate parent.
- Must be the last nested action within the *<apn:choosecontrol>* action.

Description

The *<apn:otherwise>* action belongs to the *<apn:choosecontrol>* action and is the last child to be evaluated (after all the *<apn:whencontrol>* actions). If none of the *<apn:whencontrol>* actions are set to true, the ASPX container processes the *<apn:otherwise>* tag's body content and then writes it to the current `HtmlTextWriter`.

Example

```
<apn:choosecontrol runat="server">
    <apn:whencontrol runat="server" type="INPUT">
    <% Server.Execute(Page.TemplateSourceDirectory +
"/input.aspx"); %>
    </apn:whencontrol runat="server">
    <apn:whencontrol runat="server" type="TEXTAREA">
    <% Server.Execute(Page.TemplateSourceDirectory +
"/textarea.aspx"); %>
    </apn:whencontrol>
    <apn:otherwise>
    <% Server.Execute(Page.TemplateSourceDirectory +
"/othercontrols.aspx"); %>
    </apn:otherwise>
</apn:choosecontrol>
```

<apn:ifcontrolrequired>

Evaluates its body content if the current control is defined as being required in SmartGuide Designer. This tag is functionally equivalent to testing the value returned by the `com.alphinat.xmlengine.interview.aspx.Control`'s *isRequired()* method when using scripting variables.

Syntax

```
<apn:ifcontrolrequired runat="server">
    // body content
</apn:ifcontrolrequired>
```

Body Content

ASPX. The body content is processed by the ASPX container and the result is written to the current `HtmlTextWriter`.

Attributes

None.

Constraints

None.

Description

If the current control is defined as being mandatory in SmartGuide Designer, the body content is evaluated by the ASPX container and the result is output to the current `HtmlTextWriter`.

Example

The following code displays a red asterisk if the current control is required:

```
<input type="text" runat="server" name='<apn:name
runat="server"/>' value='<apn:value runat="server"/>'>
<apn:ifcontrolrequired runat="server"><span style="
color:red">*</span> </apn:ifcontrolrequired>
```

<apn:ifnotcontrolrequired>

Evaluates its body content if the current control is not defined as being mandatory in SmartGuide Designer. This tag is functionally equivalent to negating the test of the value returned by the `com.alphinat.xmlengine.interview.aspx.Control`'s *isRequired()* method when using scripting variables.

Syntax

```
<apn:ifnotcontrolrequired runat="server">
    // body content
</apn:ifnotcontrolrequired>
```

Body Content

ASPX. The body content is processed by the ASPX container and the result is written to the current `HtmlTextWriter`.

Attributes

None.

Constraints

None.

Description

If the current control is not defined as being mandatory in SmartGuide Designer, the body content is evaluated by the ASPX container and the result is output to the current `HtmlTextWriter`.

Example

The following code will display the string " (optional) " if the current control is not required:

```
<input type="text" runat="server" name='<apn:name
runat="server"/>' value='<apn:value runat="server"/>'>
<apn:ifnotcontrolrequired runat="server"> (optional)
</apn:ifnotcontrolrequired>
```

<apn:ifrequiredcontrolexists>

Processes the tag's body if at least one control on the Smartlet page has been defined as being required.

Syntax

```
<apn:ifrequiredcontrolexists runat="server">
    // body content
</apn:ifrequiredcontrolexists>
```

Body Content

ASPX. If a control has been defined as being required exists on the Smartlet page, the body content of the *<apn:ifrequiredcontrolexists>* action is evaluated by the ASPX container and the result is output to the current `HtmlTextWriter`.

Attributes

None.

Constraints

None.

Description

If the current page of the Smartlet contains a control that has been defined as mandatory, then the *<apn:ifrequiredcontrolexists>* body content is evaluated by the ASPX container and the result is outputted to the current `HtmlTextWriter`.

Example

The following code displays the message "* indicates a mandatory field" if the Smartlet page contains at least one control that has been defined as being required:

```
<apn:ifrequiredcontrolexists runat="server">* indicates a
mandatory field </apn:ifrequiredcontrolexists>
```

<apn:ifcontrolvalid>

Evaluates its body content if the current control is valid. This tag is functionally equivalent to testing the value returned by the `com.alphinat.xmlengine.interview.aspx.Control`'s *IsValid()* method when using scripting variables.

Syntax

```
<apn:ifcontrolvalid runat="server">
    // body content
</apn:ifcontrolvalid>
```

Body Content

ASPX. The body content is processed by the ASPX container and the result is written to the current `HtmlTextWriter`.

Attributes

None.

Constraints

None.

Description

If the current control is valid, the body content is evaluated by the ASPX container and the result is outputted to the current `HtmlTextWriter`.

<apn:ifnotcontrolvalid>

Evaluates its body content if the current control is not valid. This tag is functionally equivalent to negating the test of the value returned by the `com.alphinat.xmlengine.interview.aspx.Control`'s *`IsValid()`* method when using scripting variables.

Syntax

```
<apn:ifnotcontrolvalid runat="server">
    // body content
</apn:ifnotcontrolvalid>
```

Body Content

ASPX. The body content is processed by the ASPX container and the result is written to the current `HtmlTextWriter`.

Attributes

None.

Constraints

None.

Description

If the current control is not valid, the body content is evaluated by the ASPX container and the result is outputted to the current `HtmlTextWriter`.

Example

The following code displays the alert associated with the current control in red if the value entered in the field is not valid.

```
<apn:control runat="server" id="ctrl">
  <apn:ifnotcontrolvalid runat="server">
    <span style="color:red"><%= ctrl.Current.Alert()
%></span>
  </apn:ifnotcontrolvalid>
</apn:control>
```

<apn:tooltip/>

Displays the tooltip text associated with the current control. The tooltip represents the current control's tooltip in SmartGuide Designer. This tag is functionally equivalent to invoking the `com.alphinat.xmlengine.interview.aspx.Control`'s *getTooltip()* method when using scripting variables.

Syntax

```
<apn:tooltip runat="server"/>
```

Body Content

Empty.

Attributes

None.

Constraints

None.

Description

The control's tooltip text is written to the `currentHtmlTextWriter`.

Example

The `<apn:tooltip/>` tag extracts the current control's tooltip as defined in SmartGuide Designer.

```
<input id='<apn:name runat="server"/>' name='<apn:name  
runat="server"/>' type="text" title='<apn:tooltip  
runat="server"/>' />
```

<apn:help/>

Displays the help associated with the current control. The help represents the control's detailed help in SmartGuide Designer. This tag is functionally equivalent to invoking the `com.alphinat.xmlengine.interview.aspx.Control's getHelp()` method when using scripting variables.

Syntax

```
<apn:help runat="server"/>
```

Body Content

Empty.

Attributes

None.

Constraints

None.

Description

The control's help is written to the current `HtmlTextWriter`. Help can be of type *link* or *text*. Please see the `<apn:ifhelplink>` and the `<apn:ifnohelplink>` tags for more information.

Example

The `<apn:help/>` tag extracts the current control's detailed help as defined in SmartGuide Designer:

```
<apn:control runat="server">  
  <a href='<apn:help runat="server"/>'></a>
```

```
<apn:label runat="server"/>: &nbsp;
<input type="text" value='<apn:value runat="server"/>'
name='<apn:name runat="server"/>'>
</apn:control>
```

<apn:helpid/>

Displays the id associated with the current control's detailed help. It allows referring to the help by its id when defining help icons for example (see below). This tag is functionally equivalent to invoking the `com.alphinat.xmlengine.interview.aspx.Control's getHelpId()` method when using scripting variables.

Syntax

```
<apn:helpid runat="server"/>
```

Body Content

Empty.

Attributes

None.

Constraints

None.

Description

The control's help id is written to the current `HtmlTextWriter`.

Example

```
<input type="image" value='<apn:helpid runat="server"/>'
name='<apn:helpid runat="server"/>' src="help.gif" />
```

<apn:ifhelplink>

Evaluates its body content if the current control's help is identified as a URL-type link. This tag is functionally equivalent to testing the value returned by the `com.alphinat.xmlengine.interview.aspx.Control's isHelpLink()` method when using scripting variables.

Syntax

```
<apn:ifhelplink runat="server">
    // body content
</apn:ifhelplink>
```

Body Content

ASPX. The body content is processed by the ASPX container and the result is written to the current `HtmlTextWriter`.

Attributes

None.

Constraints

None.

Description

If the current control's help is a URL-type link, the body content is evaluated by the ASPX container and the result is outputted to the current `HtmlTextWriter`.

Example

If the control's help is of type *link*, the example below will display a clickable image that opens a new browser window displaying the contents of the site identified by the URL:

```
<apn:ifhelplink runat="server">
    <a href='<apn:help runat="server"/>' target="_blank">
        </a>
</apn:ifhelplink>
```

<apn:ifnohelplink>

Evaluates its body content if the current control's detailed help is not a URL-type link (i.e. the *Hyperlink* box is not checked under the *Detailed help* input box in SmartGuide Designer). This tag is functionally equivalent to negating the test of the value returned by the `com.alphinat.xmlengine.interview.aspx.Control.isHelpLink()` method when using scripting variables.

Syntax

```
<apn:ifnohelplink runat="server">
```

```
// body content
</apn:ifnothelpink>
```

Body Content

ASPX. The body content is processed by the ASPX container and the result is written to the current `HtmlTextWriter`.

Attributes

None.

Constraints

None.

Description

If the current control's help is not a URL-type link, the body content is evaluated by the ASPX container and the result is outputted to the current `HtmlTextWriter`.

Example

If the control's help is not of type *link*, the example below will display the help content in bold, followed by a new line, the control's label, and the control's text field:

```
<apn:ifnothelpink runat="server"><b><apn:help
runat="server"/></b> </apn:ifnothelpink><br/>
<apn:label runat="server"/> <input type="text"
name='<apn:name runat="server"/>'>
```

<apn:jsfields/>

Displays a JavaScript associative array of all fields present on the page where the tag is placed, whether they are visible or not. The output is used to bring to life the page and field actions defined in SmartGuide Designer.

Syntax

```
<apn:jsfields runat="server"/>
```

Body Content

Empty

Attributes

None.

Constraints

None.

Description

The output is written to the current `HtmlTextWriter`.

Example

The following sample places the associative array of all page fields in a variable:

```
<script>
    var smartletfields = <apn:jsfields runat="server"/>;
</script>
```

A JavaScript function can then be used to loop through the page controls. E.g.:

```
for (var key in smartletfields) {
    var lastRadio;
    var fieldType = smartletfields[key].type;
    if (fieldType === 'radio') {
        lastRadio = '#div_' + key;
    }
}
$("<hr style='border-top:double' />").appendTo($(lastRadio));
```

<apn:metadata>

Displays all or specific custom attributes and values associated with the current control as HTML attributes. The values are escaped to handle quotes, less than and greater than characters.

Syntax

```
<apn:metadata runat="server" [name="attributeName"]
[match="expression"] />
```

Body Content

Empty

Attributes

| <i>Name</i> | | <i>Type</i> | <i>Description</i> |
|--------------------|--|--------------------|--|
| name | | String | Allows obtaining a specific attribute and its value for the current control. |
| match | | String | Allows using an expression to obtain specific attributes and their values for the current control. Expressions support the use of wildcards for attribute names. |

Constraints

The optional attributes cannot be used in combination.

Description

The custom attributes and values of the control are written to the current `HtmlTextWriter`.

Example 1

The following code retrieves all the custom attributes and associated values defined for the current control:

```
<apn:metadata runat="server"/>
```

Example 2

The following code retrieves the custom attribute named *placeholder* and its associated value:

```
<apn:metadata runat="server" name="placeholder"/>
```

Example 3

The following code retrieves all the data attributes and associated values for the current control:

```
<apn:metadata runat="server" match="data-*" />
```

<apn:metadatavalue/>

Displays the value of a specific custom attribute associated with the current control.

Syntax

```
<apn:metadatatype runat="server" name="attrName"
[tohtml="true|false"] />
```

Body Content

Empty

Attributes

| <i>Name</i> | | <i>Type</i> | <i>Description</i> |
|--------------------|--|--------------------|--|
| name | | String | Allows obtaining a specific attribute associated to the current control. |
| tohtml | | String | Determines whether the attribute value must be HTML encoded. If not specified <i>false</i> is assumed. |

Constraints

The *name* attribute must be specified. The value of *tohtml* can only be "true" or "false".

Description

The attribute value is written to the current `HtmlTextWriter`.

Example

The following code retrieves the value of a custom attribute named *data-mask*:

```
<apn:metadatatype runat="server" name="data-mask"/>
```

<apn:hasmetadata>

Evaluates its body content if the control attribute specified within the *name* attribute is defined.

Syntax

The body content will be evaluated if the specified attribute exists within the current control. The *name* attribute is associated with any custom attribute that could be assigned to the current control.

```
<apn:hasmetadata runat="server" name="attributeName">
    // body content
```

```
</apn:hasmetadata>
```

Body Content

ASPX.

Attributes

| <i>Name</i> | | <i>Type</i> | <i>Description</i> |
|--------------------|--|--------------------|--|
| name | | String | Allows referring to a specific custom attribute. |

Constraints

The *name* attribute must be specified.

Description

If the custom attribute is defined for the current control, the body content is evaluated by the ASPX container and the result is outputted to the current `HtmlTextWriter`.

Example

The body content will be evaluated if the *pattern* attribute has been assigned a non-empty value for the control.

```
<apn:hasmetadata runat="server" name="pattern">  
  This control expects the following pattern:  
  <apn:metadatatype runat="server" name="pattern"/>  
</apn:hasmetadata>
```

The *Type* attribute

The *type* attribute accepts the following values, depending on where it is used:

The screenshot shows a web form with several elements labeled with attribute names in boxes:

- 'generate'**: Points to a button labeled "Visualiser le formulaire avec les réponses".
- 'progress'**: Points to a progress bar showing "8 % du questionnaire complété".
- 'title'**: Points to the main heading "DESCRIPTION SOMMAIRE DU PROJET DE COOPÉRATIVE".
- 'keyword'**: Points to the text "Ministère du Développement économique, de l'Innovation et de l'Exportation".
- 'repeat-index'**: Points to the section header "1. Promoteur du projet".
- 'insert'**: Points to an "Ajouter" button.
- 'delete'**: Points to a "Supprimer" button.
- 'previous'**: Points to a "Précédent" button.
- 'next'**: Points to a "Suivant" button.
- 'summary'**: Points to a "Sommaire" button.

At the Smartlet level

| Attribute value | Description |
|--------------------------|--|
| smartlet-title | <p>Refers to the Smartlet subject as defined in SmartGuide Designer. To display the subject of the Smartlet, you would write:</p> <pre><apn:control runat="server" type="smartlet-title"> <apn:value runat="server"/> </apn:control></pre> <p>Note: The short name form "title" is also supported.</p> |
| smartlet-keywords | <p>Refers to the Smartlet keywords as defined in SmartGuide Designer.</p> <p>Note: The alternate names "smartlet-keyword",</p> |

“keyword”, and “keywords” are also supported.

| | |
|----------------------------------|--|
| smartlet-code | Refers to the Smartlet code as defined in SmartGuide Designer. Note: The alternate name “interview-code” is also supported. |
| smartlet-name | Refers to the Smartlet name as defined in SmartGuide Designer. |
| smartlet-description | Refers to the Smartlet description as defined in SmartGuide Designer. |
| smartlet-theme | Returns the current theme used by the Smartlet. |
| smartlet-author | Returns the author of the Smartlet. |
| smartlet-lastmodification | Returns the last modification date of the Smartlet. |
| progress | Refers to the Smartlet progress bar. |
| step | Refers to the current Smartlet page title. |
| section | Refers the the current Smartlet page section, if defined. |

On navigation buttons

| Attribute value | Description |
|-----------------|---|
| next | Refers to the button that allows navigating from a Smartlet page to the next. |
| previous | Refers to the button that allows navigating from a Smartlet page to the previous. |
| modify | Refers to the buttons shown on a data recap page that allow navigating to the different Smartlet pages. |
| summary | Refers to the button that allows navigating back to a data recap page. |

In repeated groups

| Attribute value | Description |
|-----------------|---|
| insert | Refers to the repeated group’s button that allows adding an instance. |

| | |
|------------------------------|---|
| delete | Refers to the repeated group's button that allows deleting an instance. |
| repeat-index | Refers to the repeated group's field containing the index of the item to be deleted. |
| cancel_add_instance | Represents a button to cancel adding an instance to a repeat group when using grid view mode. |
| save_add_instance | Represents a button to save the added instance to a repeat group when using grid view mode. |
| cancel_edit_instance | Represents a button to cancel edition of an instance of a repeat group when using grid view mode. |
| save_edit_instance | Represents a button to save an instance of a repeat group currently being edited when using grid view mode. |
| delete | Represents a button to delete a repeat group instance. |
| prepare_add_instance | Represents a button to prepare the addition of a new repeat group instance when using grid view mode. |
| prepare_edit_instance | Represents a button to prepare the edition of an existing repeat group instance when using grid view mode. |
| default-instance | Represents the default instance of a repeat group. |

At the PDF level

| Attribute value | Description |
|-----------------|--|
| generate | Refers to the button that allows generating a PDF. |

In controls

| Attribute value | Description |
|-------------------|---|
| input | An input control of type <i>text</i> . |
| statictext | A field of type <i>static text</i> . |
| textarea | A <i>text area</i> control which accepts entering multiple lines of text. |
| secret | An input control of type <i>password</i> . |
| select1 | A select control of type <i>check box</i> or <i>list box</i> . |
| optgroup | Represents a group of choices within a select or |

| | |
|------------------------|--|
| | <i>select1</i> control. |
| date | Represents a date value. |
| image | A field of type <i>image</i> . |
| trigger | A control of type <i>button</i> . |
| upload | A control of type <i>upload</i> . |
| sub-smartlet | Represents an embedded Smartlet. |
| repeat | A set of repeated groups. |
| group | A group of controls and other groups. |
| result | Represents a list of knowledge base entries. |
| row | Represents a row of controls when using a layout enabled theme. |
| col | Represents a column of controls when using a layout enabled theme. |
| summary-section | A summary element. The type "recap" is also supported. |

At the embedded Smartlet level

| Attribute value | Description |
|---------------------------|--|
| edit-sub-interview | Refers to the button that allows accessing an embedded Smartlet. |
| return | Refers to the button that allows returning to the parent Smartlet. |

The *items* attribute

The *items* attribute is supported by the *forEach* tag and accepts the following values:

| Attribute value | Description |
|--------------------------|---|
| alert-controls | Allows iterating over all the controls that contain alert messages which appear if an error has occurred. |
| global-navigation | Allows iterating over the navigation links to all the pages of a Smartlet. |

List of control types

The *getType()* method of a control returns a numeric ID representing the type of that control. Here is the complete list of IDs and their corresponding control:

| Type ID | Control |
|-------------|--------------------------------|
| -1 | Undefined |
| 1000 | group |
| 1001 | input |
| 1004 | repeat |
| 1005 | password |
| 1006 | checkbox or listbox |
| 1007 | radio button or drop-down menu |
| 1008 | textarea |
| 1009 | button |
| 1010 | upload |
| 1011 | static text |
| 1012 | image |
| 1013 | Result/knowledge |
| 1014 | date |
| 1015 | Sub-Smartlet |