

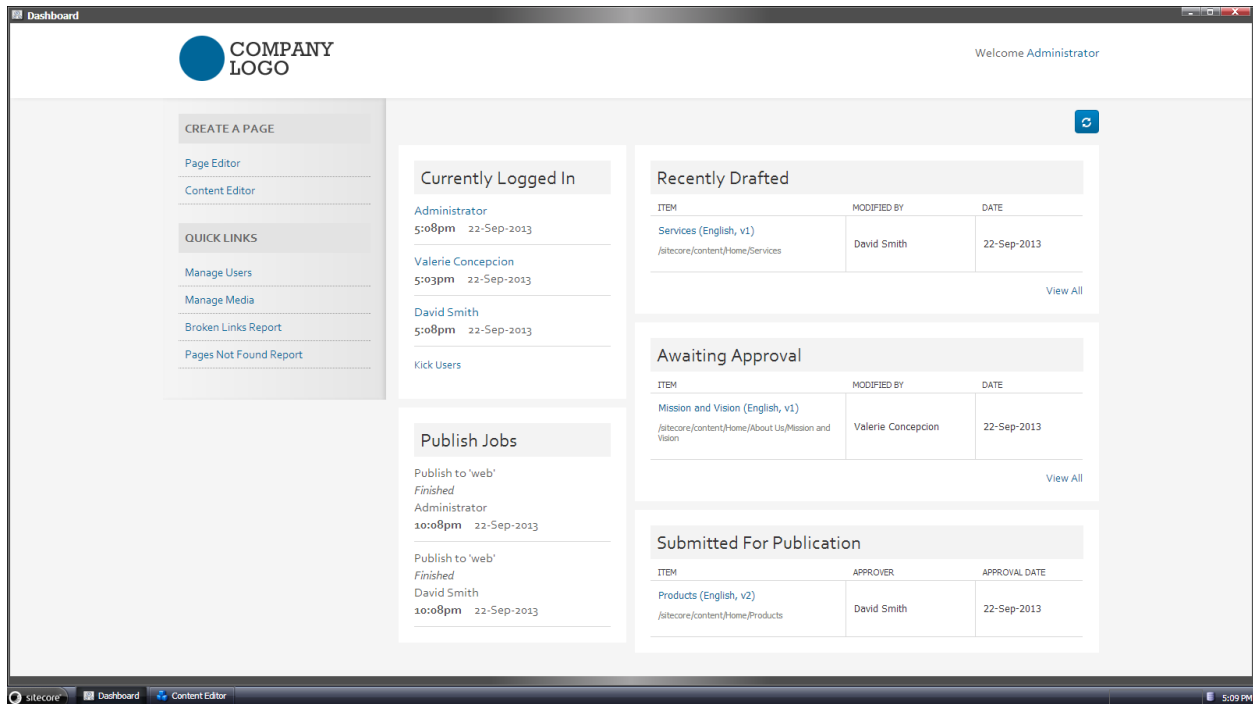
Sitecore Dashboard User Guide

Contents

- Overview 2
- Installation 2
- Getting Started..... 3
 - Sample Widgets 3
 - Logged In 3
 - Job Viewer 3
 - Workflow State 3
 - Publish Queue Viewer 4
 - Quick Links 4
- Configuration 4
 - Adding Quick Links 5
 - Repositioning Widgets 6
 - Common Widget Properties 6
 - Sample Widget Properties 7
 - Selecting a Theme 7
- Customization 7
 - Creating a Custom Theme..... 7
 - Building Custom Widgets..... 8
 - Real-Time Updates..... 8
 - Creating a View 9
 - Creating a Model Class..... 10
- Project Info..... 11

Overview

The Sitecore Dashboard is a Sitecore app that aggregates real-time data about your site into a customizable widget-based view.



Installation

Requirements:

- .NET 4.0+
- IIS 7+
- Json.NET 4.5 (contained in Sitecore package and will overwrite existing version of Newtonsoft.Json.dll)
- Tested on Sitecore 6.4.1, 6.5, and 6.6

Steps:

1. Install Sitecore package
2. Add to <runtime>/<assemblyBinding> in Web.config:

```
<dependentAssembly>
  <assemblyIdentity name="Newtonsoft.Json" publicKeyToken="30ad4fe6b2a6aedd" />
  <bindingRedirect oldVersion="1.0.0.0-3.6.0.0" newVersion="4.5.0.0" />
</dependentAssembly>
```

Getting Started

The module comes preconfigured with a sample dashboard view and set of widgets.

Sample Widgets

Logged In

Currently Logged In
Administrator 5:08pm 22-Sep-2013
Valerie Concepcion 5:03pm 22-Sep-2013
David Smith 5:08pm 22-Sep-2013
Kick Users

This widget displays the names of users who are currently logged into the CMS and a timestamp of their last activity. The “Kick Users” link opens the /sitecore/shell/Applications/Login/Users/Kick.aspx page which allows admins to terminate user sessions.

Job Viewer

Publish Jobs
Publish to 'web' <i>Finished</i> Administrator 10:08pm 22-Sep-2013
Publish to 'web' <i>Finished</i> David Smith 10:08pm 22-Sep-2013

This widget displays a list of recently queued jobs. Each row contains the name of the job, the job status, the user that started the job, and the time at which the job was queued.

Workflow State

Recently Drafted		
ITEM	MODIFIED BY	DATE
Services (English, v1) /sitecore/content/Home/Services	David Smith	22-Sep-2013

[View All](#)

This widget shows a preview of the items in a particular workflow state (as listed in the Workbox). The “View All” link opens the Workbox application with the workflow pre-selected. Each row includes the item name, version information, path, and the associated user and date of the last event in the item’s workflow history.

Publish Queue Viewer

Submitted For Publication		
ITEM	APPROVER	APPROVAL DATE
Products (English, v2) /sitecore/content/Home/Products	David Smith	22-Sep-2013

This widget displays items that are in the publish queue and therefore would be included in the next incremental site publish. If the item is in a workflow, the associated user and date of the last event in the item’s workflow history is displayed. If the item is not in a workflow, the last modified user and date are displayed. This widget supports pagination since there is no Sitecore application which displays a full list of this data.

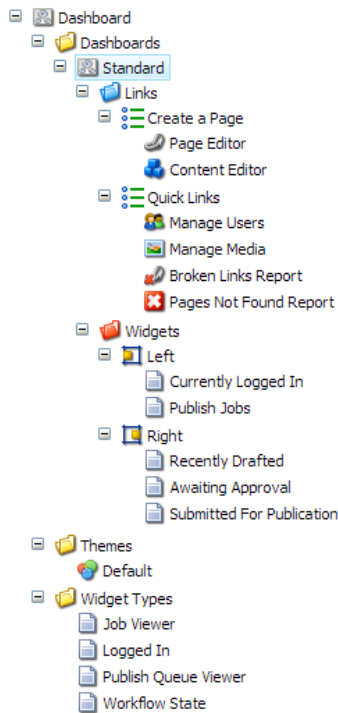
Quick Links

QUICK LINKS
Manage Users
Manage Media
Broken Links Report
Pages Not Found Report

In addition to the widgets that appear in the main area of the Dashboard, the side rail may contain groups of links which may open either a Sitecore application or URL.

Configuration

The default Dashboard view is configured in Sitecore under `/sitecore/content/Applications/Dashboard` in the core database.



The “Dashboards” folder may contain multiple Dashboard configurations (i.e., combinations of widgets, links, theme, etc.). Currently, the only way to change which Dashboard configuration is displayed is to modify the “Dashboard.DefaultDashboard” setting in Sitecore.Dashboard.config. Future enhancements may include allowing different users to view different configurations.

Adding Quick Links

Each Dashboard configuration contains a “Links” folder under which you can add items based on the “Application Shortcut” or “Link” template.

Note that the “Application Shortcut” template is part of the core Sitecore product. Therefore, certain items under the /sitecore/Applications tree may simply be copied to the Quick Links folder rather than creating the item from scratch.

 **Data**

[Insert Link](#) • [Insert Media Link](#) • [Insert External Link](#) • [Insert Anchor Link](#) • [Insert Mail Address Link](#) • [Follow](#) • [Insert JavaScript Link](#) • [Clear](#)

Application [shared]:


Display name:

Icon [shared]:

Parameters [shared]:

Tool tip:

A link may also point to any URL by creating an item based on the “Link” template, which contains a single General Link field.

 **Link**

[Insert Link](#) • [Insert Media Link](#) • [Insert External Link](#) • [Insert Anchor](#) • [Insert Email](#) • [Insert JavaScript](#) • [Follow](#) • [Clear](#)

Link [unversioned]:

Links are grouped using the “Link Section” template, which contains a field for header text.

Repositioning Widgets

Widgets are assigned to a “Position”/placeholder in the Dashboard layout. The out-of-the-box “Left” and “Right” positions reference the columns in the main content area of the Standard Dashboard. New positions can be created as children of the “Widgets” folder, but must be added by a developer to the Dashboard layout (ASPX).

To reposition a widget, simply move the Widget item under the desired Position. Re-ordering items under a given position also updates the order in which Widgets are displayed within a given Position.

Common Widget Properties

Each Widget contains the following fields which may be edited in Sitecore:

- **Name** – the title to appear at the top of the Widget
- **Type** – the Widget Type which defines the code used to render the Widget content
- **Css Class** – an optional CSS class to be applied to the widget’s HTML container

- **Parameters** – an optional list of key/value pairs used to configure the Widget. (Note that a Widget Type may be used by multiple Widgets, so this allows each instance to be configured differently.)

Sample Widget Properties

The following table contains a list of parameters that may be used with certain sample widget types.

Widget Type	Parameter Name	Description
Job Viewer	JobCategoryFilters	Use to only display jobs in the given job category (e.g., Publish)
Workflow State	WorkflowID	Sitecore ID of the Workflow item
Workflow State	WorkflowStateID	Sitecore ID of the Workflow state item
Workflow State	MaxItems	Number of items to display before showing the “View All” link
Publish Queue Viewer	PageSize	Number of items to display per page

Selecting a Theme

Themes define the Dashboard look-and-feel and allow for custom branding of the UI. This module only comes with a single “Default” theme. However, if additional themes are developed (see [Creating a Custom Theme](#)), the theme to use may be selected on the Dashboard configuration root (e.g., /sitecore/content/Applications/Dashboard/Dashboards/Standard).

Customization

Creating a Custom Theme

The look-and-feel of the Dashboard may be customized by creating a CSS file and corresponding Theme item that references it under /sitecore/content/Applications/Dashboard/Themes. This file will be inserted after the base stylesheet and thus may be used to override any default styles.

Example:

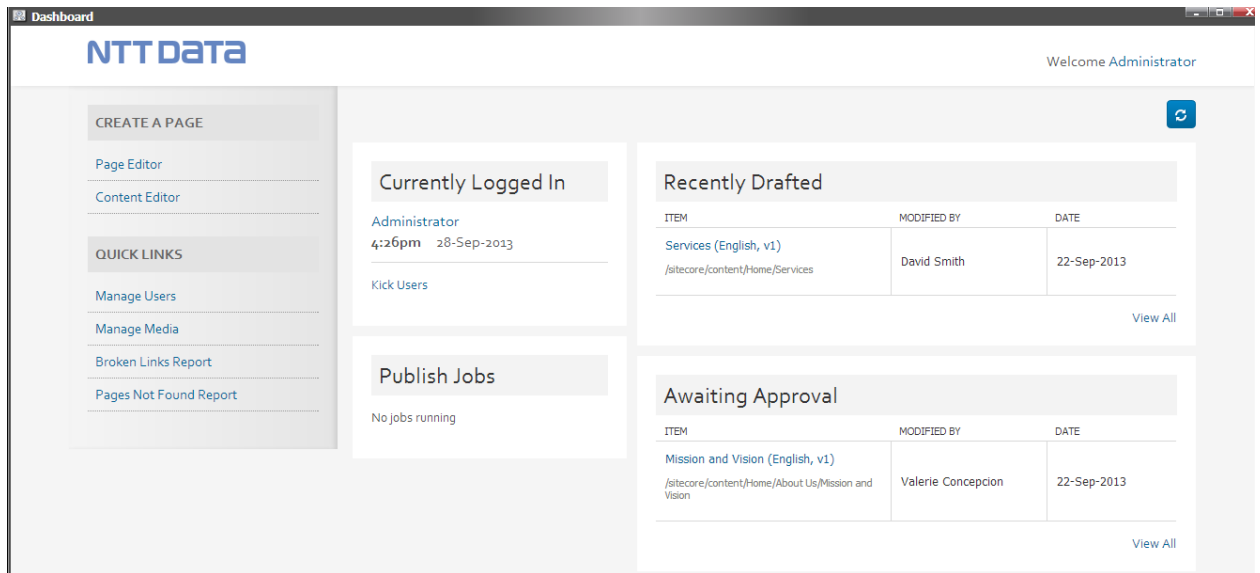
CSS file (/css/dashboard.css):

```
header .brand-logo {
    background: url('../img/ntt_data_logo.png') no-repeat 0 0;
    height:26px;
    width:168px;
}
```

Theme item (/sitecore/content/Applications/Dashboard/Themes/NTT DATA):



Result:



Building Custom Widgets

Technically speaking, a Widget is simply a User Control that inherits the `Sitecore.Dashboard.Web.UI.Widgets.Widget` base class. You may develop your own User Control and create a new item under `/sitecore/content/Applications/Dashboard/Widget Types` which references it by path.

Inheriting the Widget base class provides easy access to the “Parameters” field on the Widget item in Sitecore. Referencing the “Parameters” property from your code-behind will return a `NameValueCollection` containing the keys and values on a given Widget item.

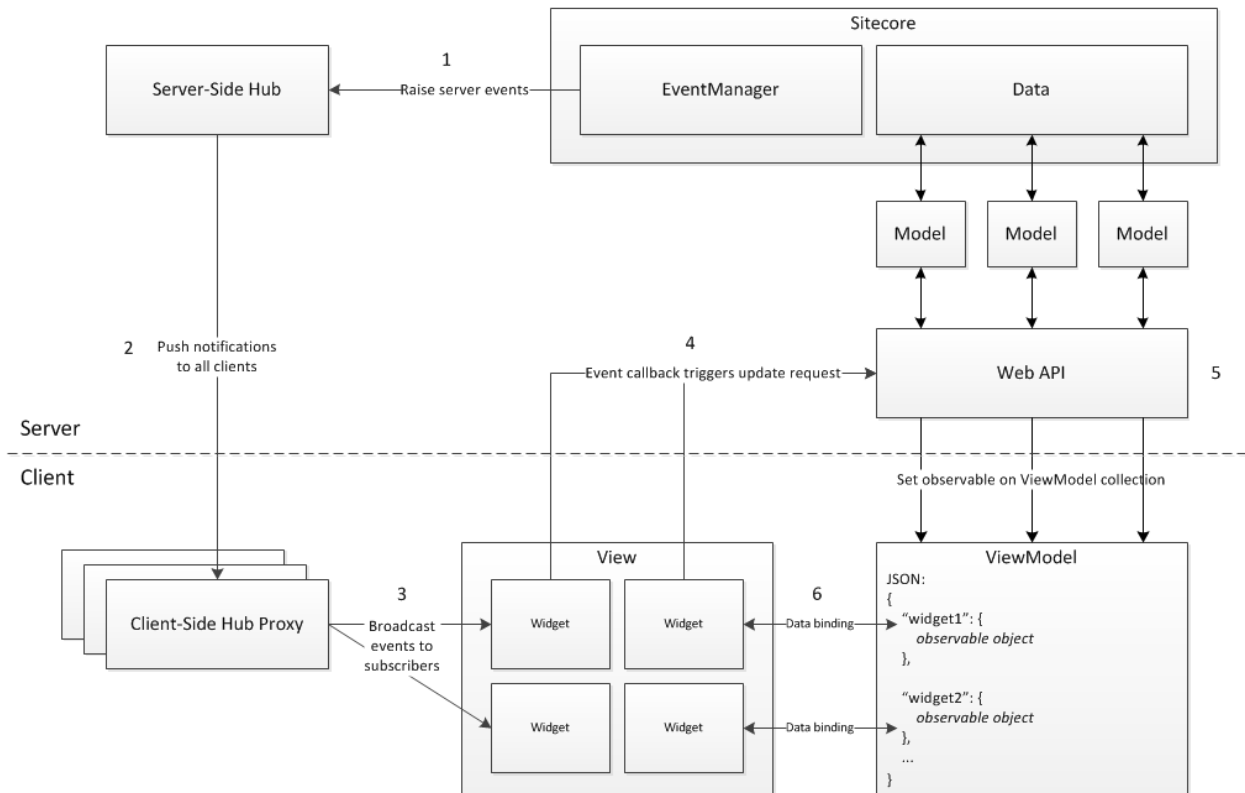
Real-Time Updates

A unique feature of the Sitecore Dashboard module is that it provides a framework for pushing live updates to Widgets as Sitecore events are triggered. This means that updates can be seen in real-time as opposed to a user having to constantly refresh the view.

The Sitecore Dashboard leverages [Microsoft SignalR](#) (included with ASP.NET 4.5), an open source library that supports “server push” functionality through the use of the HTML5 WebSocket transport or a fallback mechanism where necessary (e.g., AJAX long polling, Forever Frame, etc.). The server pushes Sitecore events to the client, which then updates the Widgets subscribed to those events via AJAX.

Below is an overview of the process through which Sitecore events trigger updates to the client UI.

Sitecore Dashboard



1. Sitecore server-side event is raised
2. Dashboard Hub broadcasts message (i.e., name of the event) to all clients
3. Client-side hub proxy raises jQuery event
4. Widgets subscribed to jQuery event make AJAX call to get latest data from Web API
5. Web API gets updated model from Sitecore and returns as JSON
6. Widget view model is updated based on response and view is automatically refreshed through declarative data binding ([Knockout JS](#))

The LoggedIn and JobViewer Widgets are good examples to start from. However, if you wish to create a real-time Widget from scratch, the steps are as follows:

Creating a View

1. Create a User Control (ASCX) that inherits the Sitecore.Dashboard.Web.UI.Widgets.Widget class.
2. Using Knockout JS, setup a ["with" binding](#) around your widget HTML, whose binding context is the User Control's ClientID. Add declarative bindings to the contained HTML to bind to properties of the view model.

```
<!-- ko with: <%= ClientID %> -->
<!-- Add HTML elements with data-bind attribute -->
<!-- /ko -->
```

3. Add JavaScript to create a Widget object whose constructor takes the value of the ClientID and ApiRoute property (inherited from the Widget base class). Note: the ApiRoute is based on the GUID of the Widget item which determines the URL of the Web API call.

```
<script>
    $j(document).ready(function() {
        // Constructor takes ID of widget to auto-refresh and URL of Web
API call
        var widget = new Widget('<%= ClientID %>', '<%= ApiRoute %>');
        ...
    });
</script>
```

4. Using jQuery.bind, add a callback function for the events that should trigger an update to this Widget. The callback function should call widget.update().

```
<script>
    $j(document).ready(function() {
        // Constructor takes ID of widget to auto-refresh and URL of Web
API call
        var widget = new Widget('<%= ClientID %>', '<%= ApiRoute %>');
        // Bind update event handler to Sitecore security events
        $j(document).bind('job:started job:ended', function() {
            widget.update();
        });
    });
</script>
```

Creating a Model Class

1. Create a Model class that inherits the Sitecore.Dashboard.Models.WidgetModel class.
2. Override the Initialize() method that populates the Model. This will be converted to JSON by the Web API WidgetsController.

```
public class JobViewer : WidgetModel
{
    public override void Initialize()
    {
        // Populate model
    }
}
```

3. On the Widget Type item in Sitecore, specify the class signature (including assembly) in the "Model Type" field.

Widget Type	
Control Path [shared]:	/sitecore modules/Web/Sitecore.Dashboard/UI/Widgets/JobViewer.ascx
Model Type - for real-time data binding (optional) [shared]:	Sitecore.Dashboard.Models.JobViewer, Sitecore.Dashboard

Project Info

To report issues or contribute to the project, check out the project page at <https://github.com/NTTDATA/Sitecore-Dashboard>.

For general questions and comments, contact Valerie Concepcion (valerie.concepcion@nttdata.com).