DotNetNuke Client API - DragDropAdminModules

Jon Henning

Version 1.0.0
Last Updated: June 20, 2006
Category: Client API

DotNetNuke Client API - DragDropAdminModules

Information in this document, including URL and other Internet Web site references, is subject to change without notice. The entire risk of the use or the results of the use of this document remains with the user.

The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, places, or events is intended or should be inferred.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Perpetual Motion Interactive Systems, Inc. Perpetual Motion Interactive Systems may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Perpetual Motion, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

DotNetNuke® and the DotNetNuke logo are either registered trademarks or trademarks of Perpetual Motion Interactive Systems, Inc. in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Abstract

In order to clarify the intellectual property license granted with contributions of software from any person or entity (the "Contributor"), Perpetual Motion Interactive Systems Inc. must have a Contributor License Agreement on file that has been signed by the Contributor.

Contents

DotNetNuke Client API - DragDropAdminModules

Introduction

Utilizing the DotNetNuke Client API enables the developer to offer a rich client-side feature-set. This document outlines one of those feature-sets; providing drag-N-drop functionality for maintaining module layout.

Goals

Communicate and document the steps involved in creating rich client-side feature-set.

Provide a better understanding of the inner-workings of the Drag-N-Drop feature-set.

Client Code Allowing Containers To Be Dragged

The code behind enabling any container to be dragged is pretty straightforward. Especially since we are going to use the DNN Client API to abstract all of the inconsistencies between browsers. Most moveable containers all have 2 things on common. First, they have an area, usually the Title area, that can be clicked and while holding the mouse button down dragged to a new position. Second, they have a content area that represents the entire "dialog" including the "title" area. With this in mind we have coded the following client-side function found in the dnn.dom.positioning namespace.

enableDragAndDrop(oContainer, oTitle, sDragCompleteEvent)

The first parameter is a reference of the container element, the second is the title element that will respond to the click-and-drag. And the final parameter is an optional parameter to notify the caller when a drag is finally dropped.

Attach Events and Setting Properties
Behind the scenes the function is attaching an event to the body.onmousemove, body.onmouseup, and the title control's onmousedown event. Additionally, the method will set the title's cursor to a hand, set the containers position style to relative (if not already set), assign an ID for the container if one is not provided (see notes below), set the container's ID to a custom attribute (contID) on the title element, and set another custom attribute on the title element for the drag complete event (dragComplete).

MouseDown Event
When the mousedown event is fired a global variable in the positioning namespace is set to a reference of the title control that raised the event.

MouseUp Event
When the mouseup event is fired a check to see if a sDragCompleteEvent is provided, if so then that function is fired where the caller can have access to the control that fired it through the global reference mentioned above (dnn.dom.positioning.dragCtr). Finally, the reference to the moved control is set to null.

MouseMove Event
This event simply detects if the dnn.dom.positioning.dragCtr is set. If so it will call the dnn.dom.positioning.dragContainer method passing in the reference of the "title" element being dragged.

dnn.dom.positioning.dragContainer Method
This method is pretty straightforward. The only trick is assigning the initial offset for the places on the title bar that were clicked. If we did not do this when the control was dragged the mouse cursor would always be in the upper-left corner of the container while dragging. Simply setting the container's top = event.clientY – offset and the left = event.clientX – offset will do. There is one additional thing to worry about, whether the page is scrolled. We can detect this through the dnn.dom.positioning.bodyScrollTop and dnn.dom.positioning.bodyScrollLeft methods. We will need to subtract these values off of the equation mentioned above.

dragComplete – dnncore.js
When the user lets go of the mouse button the __dnn_dragComplete is called where it will initiate a custom PostBack to persist the new module location to the database.

Note: *You may be wondering why we are using the body's onmousemove event instead of the title's event. This is because you can move the mouse faster than the page can render the moved container, thus you will loose some events and the container will not move as smoothly.*

Enabling DNN Containers To Be Hooked Up

The only thing left is to register/reference the appropriate client namespace and call enableDragAndDrop for each container that wants the functionality.

Enabling Drag-N-Drop Module Layout
The DNN core code has been modified to enable the Modules to support Drag-N-Drop when in Layout mode. The two pieces of information that the client needs to know are the Title control's ID and the container's ID. Currently, the code will insert a new control into the collection that will act as the module's container. This is necessary since the core has no control over what the skin's container will look like, therefore it cannot obtain its ID. By having the core a container object that acts as the container object's parent we are guaranteeing that the client-side will have an accessible object to reference. Getting the Title ID currently is making an assumption that the container's Title skin object ID was not renamed from dnnTitle. In the slim chance that the skin author changed this ID, the Drag-N-Drop functionality will not be enabled.

ClientAPI.EnableContainerDragAndDrop
The MoveableContainers code will search for the supplied Control IDs. If it determines that they exist then a call is made to the ClientAPI.EnableContainerDragAndDrop method. This method will register the dnn.dom.positioning namespace, add a client side body load event handler, and register client variables (__dnn_dragDrop) for each container that will be moveable.

dnncore.js - __dnn_enableDragDrop()
This client-side function will look for 1 or more client variables (__dnn_dragDrop) and call the dnn.dom.positioning.enableDragAndDrop function for each.

Docking, Visual Indicators, and PostBacks

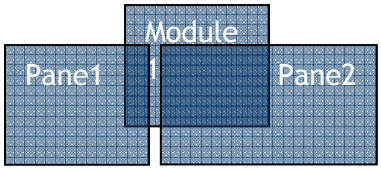
Having never written any code to provide docking functionality I decided to ask the community for some pointers on where to start. Armand (nokiko) was kind enough to share some script with me that got me on the right track. From that script I determined that in order to doc a component we need to have the ability to score the modules overlapping sections with each pane.

Getting List of Pane IDs and Names

In order to determine the overlap scores I first needed to get a list of the panes from the server side. This is accomplished inside the ClientAPI.EnableContainerDragAndDrop method. Inside this function the

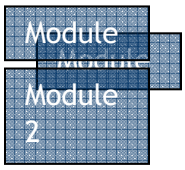
PortalSettings.Panes collection is looped and concatenated into a string and sent down to the client via the __dnn_Panes and __dnn_PaneNames variables.

Scoring



In viewing the example above it is obvious that Pane2 should be scored higher than Pane1. But to programmatically score this we need to apply some simple geometry. By multiplying the Length x Width we will get the Area of overlap. The logic actually is a little more involved since we have no way of knowing how the overlap is occurring (whose on 1st … um, I mean Left, and which one (pane or module) to use for the right coordinate, but you should get the basic idea. If you care to know the inner details then simply look at the elementOverlapScore function found in the dnn.positioning.js file.

Which Position/Order?



To determine which position/order to place the module in we simply need to compare the top properties of modules found in the selected Pane. Which ever module is the first one that has a larger top property dnn.dom.positioning.elementTop(oMod) is the index that we will use.

Indicating Where Module Will Be Dropped

In order to visually show where the module will be dropped the dnn.dom.positioning.enableDragAndDrop function allows for a parameter to be passed in to be called when the container is being dragged. Since this method will be called quite often and the Scoring and Index calculation is not exactly cheap, we want to only do the calculations once in X number of calls. For now this number is set at 75. The dnncore.js __dnn_dragOver function is used for this. It will Score and determine the Module Order for the current position of the dragged container. It will then apply a border to the appropriate Pane and will place a border on the module that indicates where it will fall in the order.

Dropping Module

When the module is finally released the dnncore.js __dnn_dragComplete function is called (a parameter into the enableDragAndDrop function assigns this). It will go through the same exercise that the _dnn_dragOver function did to determine where to place the module. It will then utilize the ClientAPI to remove the module from the pane's control collection and place it in the new position. The _dnn_MoveToPane function is called to handle this.

Once the module is in its proper place a few things could happen. Eventually the ClientAPI will enable a call to be made to the server without a postback, but for now the only option available to us is the use of the ClientAPI's PostBack functionality. This is accomplished through the following code.

```
dnn.callPostBack('MoveToPane', 'moduleid=' + sModuleID, 'pane=' + oPane.paneName, 'order=' + iIndex * 2);
```

See DotNetNuke_ClientAPI.doc for more details on how this works.

Additional Information

The DotNetNuke Portal Application Framework is constantly being revised and improved. To ensure that you have the most recent version of the software and this document, please visit the DotNetNuke website at:

http://www.dotnetnuke.com

The following additional websites provide helpful information about technologies and concepts related to DotNetNuke:

DotNetNuke Community Forums
http://www.dotnetnuke.com/tabid/795/Default.aspx

Microsoft® ASP.Net
http://www.asp.net

Open Source
http://www.opensource.org/

W3C Cascading Style Sheets, level 1
http://www.w3.org/TR/CSS1

Errors and Omissions

If you discover any errors or omissions in this document, please email marketing@dotnetnuke.com. Please provide the title of the document, the page number of the error and the corrected content along with any additional information that will help us in correcting the error.

Appendix A: Document History

Version	Last Update	Author(s)	Changes
1.0.0	Sep 18, 2004	Jon Henning	• Created
1.0.1	Aug 16, 2005	Shaun Walker	• Applied new template