Pre Demo Checklist

* Ensure Custom Errors On
* Set fonts to 14
* Set notepad font size to 16
* Delete c:\temp
* Clear out code templates
* Launch Bank Transfer site
* Drag necessary files into desktop

Ajax Action Link

* Open up project and add Script files in correct order to Site.master
* Add Ajax.ActionLink to Index.aspx
* Implement Controller Action to return string
* Show it

Ajax Form

* Implement Ajax.Form targeting the UL
* Add Text Field
* Implement Controller action to return partial view
* Implement the Partial View

<li><%= Model %></li>

* Demo
* Add Confirm Property, try it again
* Set the HttpMethod
* Show again
* Update AcceptVerbs to also respond to PUT
* Implement Simple OnSuccess method With Alert
* Discuss the context argument that nobody knows about (Show Sys.Mvc.AjaxContext slide).
* Implement OnSuccess with:

var selector = '#' + context.get\_updateTarget().id + ' :last';

$(selector).hide().fadeIn("slow");

One Way Client Template

* Open up project, what we’re going to build is an Image gallery
* Add Markup For Images

<div id="images" class="sys-template">

<img sys:src="{binding Uri}" sys:title="{binding Description}"

width="140" height="104" />

</div>

* Now We Need Data For the Images. Write JSON Projection

public JsonResult Images(string orderBy) {

using (var context = new ImagesDataContext()) {

var jsonProjection = from image in context.Images

select new {

ImageID = image.ImageID,

Name = image.Name,

Description = image.Description,

Contributor = new {

FirstName = image.People.FirstName,

LastName = image.People.LastName

},

Uri = image.Uri

};

return Json(jsonProjection.ToList());

}

}

* Now we add Some Scripts to Site.Master and set the sys XML Namespace.

<script src="../../Scripts/jquery-1.3.2.js" type="text/javascript"></script>

<script src="../../Scripts/MicrosoftAjax.debug.js" type="text/javascript"></script>

<script src="../../Scripts/MicrosoftAjaxTemplates.debug.js" type="text/javascript"></script>

<body xmlns:sys="javascript:Sys">

* Finally, write some JavaScript to bind the JSON data to our template

<script type="text/javascript">

$(document).ready(function() {

var dataContext = $create(Sys.Data.DataContext, {

serviceUri: "/Home",

saveOperation: "null"

});

var imagesView = $create(Sys.UI.DataView,

{

autoFetch: true,

fetchOperation: "Images",

dataProvider: dataContext,

initialSelectedIndex: 0,

selectedItemClass: "selected"

},

null,

null,

$get("images")

);

});

</script>

* Explain
* Run it

Master Detail Client Template

* Implement Master Detail.
* Add Detail HTML

<fieldset id="image-detail" class="sys-template">

<legend>Image Details</legend>

<div class="form">

<div>

<label for="name">Name</label>

<input id="name" type="text" value="{binding Name}" />

</div>

<div>

<label for="description">Description</label>

<input id="description" type="text" value="{binding Description}" />

</div>

<div>

<label for="uri">Uri</label>

<input id="uri" type="text" value="{binding Uri}" />

</div>

</div>

<img alt="{binding Name}" sys:src="{binding Uri}" />

</fieldset>

* Turn the Image List into a Ribbon

<div class="ribbon-container">

<div id="images" class="sys-template ribbon">

<div>

<span>{binding Name}</span>

<img sys:src="{binding Uri}" sys:title="{binding Description}" width="140" height="104" sys:command="select" />

</div>

</div>

</div>

* Bind the Details View to the List View

var imageDetailView = $create(Sys.UI.DataView,

null,

null,

null,

$get("image-detail")

);

$create(Sys.Binding, {

source: imagesView,

path: "selectedData", // selectedData is a property of DataView which is set when an item is selected. In this case, we're binding productView to it.

target: imageDetailView,

targetProperty: "data"

});

* Try Running it Again
* Explain

Two Way Data Binding

* Implement Master Detail.
* Add a Save Button

<input id="save-button" type="button" value="Update" />

* Add Event Handler So Save Button Saves Changes

$("#save-button").live("click", function() { dataContext.saveChanges(); });

* We’re Not Done Yet. What Method Is this going to call?
* Add Reference to

<script src="../../Scripts/AspNetMvcDataContext.js" type="text/javascript"></script>

* Change DataContext to Sys.Data.AspNetMvcDataContext and Show Convention
* Implement Update method

public void Update([Json]Image image) {

using (var context = new ImagesDataContext()) {

context.Images.Attach(image, true);

context.SubmitChanges(); // this works because I turned off concurrency checking for demo purposes. I'm EVIL!

}

}

* Show the JsonModelBinder and why it’s necessary
  + DefaultModelBinder handles form encoded data