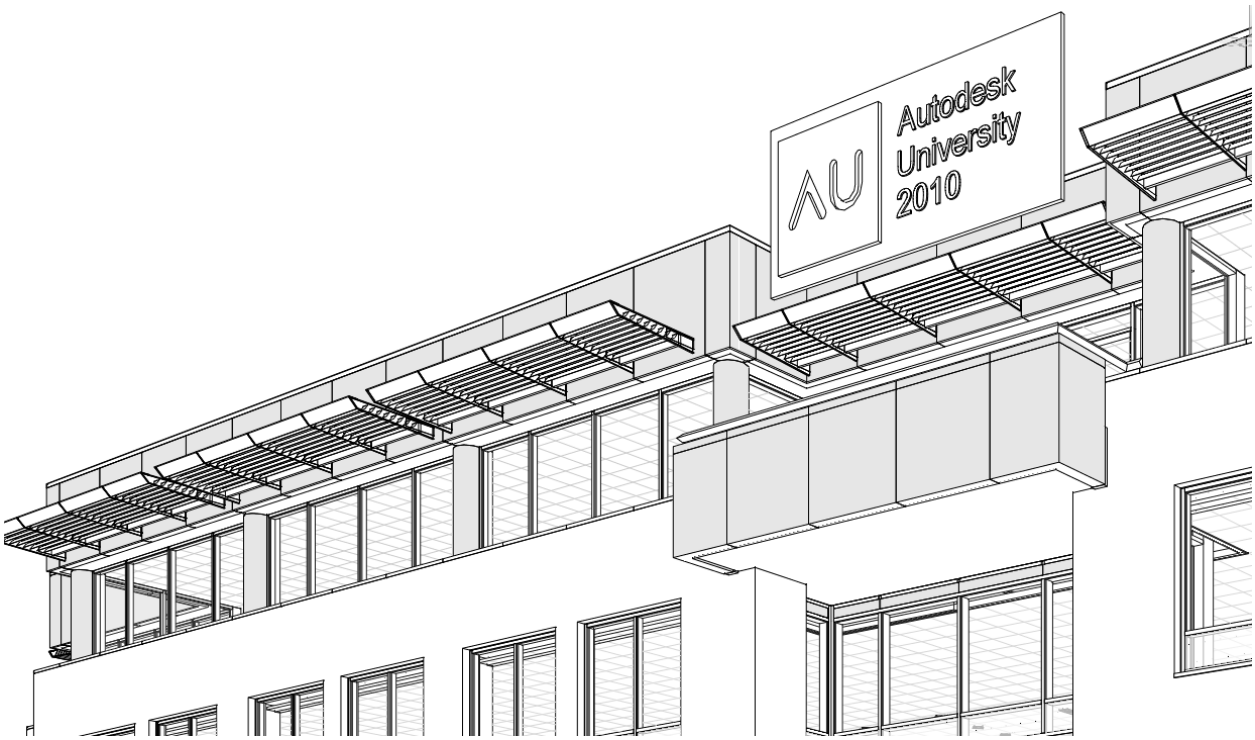
Leveraging the Tail End of the BIM Lifecycle with APIs

Don Rudder – HOK

CP333-1 Building owners constantly express a passionate desire for a BIM product that they can utilize in their day-to-day operations. Shifting the completed Revit® BIM model and data into a light-weight, internet-friendly 3D model and database is the answer. Building owners cannot deny the all-so-familiar web experience when the power and simplicity of the page meets their needs. This class will provide people of various backgrounds an overview on how the tail end of the BIM life cycle can be leveraged using various application programming interfaces (API).



"*You cannot manage what you do not measure*"

About the Speaker:

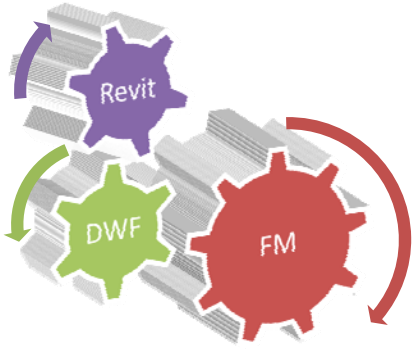
Don is a BIM Manager at HOK San Francisco and is a long time Revit®, AutoCAD®, and Web developer with more than ten years of electrical and mechanical design experience. Development expertise includes C#.NET, VB.NET and F#.NET for Revit Architecture and AutoCAD MEP. He is a key contributor for the API chapter for a new book entitled *Mastering Revit Architecture 2011*. His design experience includes large government projects, power generation facilities, large international resorts, telecom, aviation, education facilities, science and technology, and health care.

don.rudder@hok.com

http://www.revitnet.blogspot.com

Introduction

Facility Management is an incredibly complex and important responsibility, but as with anything, can be better understood when broken down into smaller easier to understand parts.



This document illustrates how the Autodesk Revit 2011 and Autodesk Design Review 2011 APIs can be leveraged along with a Microsoft ASP.NET web site to produce a highly visual and data rich Facility Management (FM) and Workplace Management (WM) system.

The information contained in this document is intended to demonstrate general API capabilities relating to FM and not necessarily to suggest or recommend Facility Management (FM) processes.

The Downloadable CP333-1 Class Dataset

This document should be referenced in conjunction with the class dataset available alongside this document on the AU Website. Three distinct data sets have been provided within the main zip file.

Revit Models

All three Revit 2011 models used in the presentation have been provided as well. These models have been stripped down from their original design as a means to remain anonymous and as a result may not represent real life accuracies in terms of design.

Revit API DynamiLink

The provided Revit API sample project was written in Visual Basic using Visual Studio 2010. The project is used to connect and synchronize with a Microsoft Access 2007 database.

ASP.NET Web Site

The provided web site uses Visual Basic as the server side language and JavaScript code as the client side language. The sample web site contains a home page, a data management page, and a floor plan page containing an embedded DWF viewer.

Tips to Improve FM Compatibility with an Autodesk Revit 2011 Model

The tips outlined in this section are intended to improve the quality and usefulness of a DWF file exported from Revit 2011. The Autodesk DWF file format and Autodesk Design Review 2011 provide an easy and very lightweight means for accessing and interacting with 2D and 3D floor plans over the internet.

Use Only "Hidden Line" for All 2D Views (Avoid Using Shaded Views)

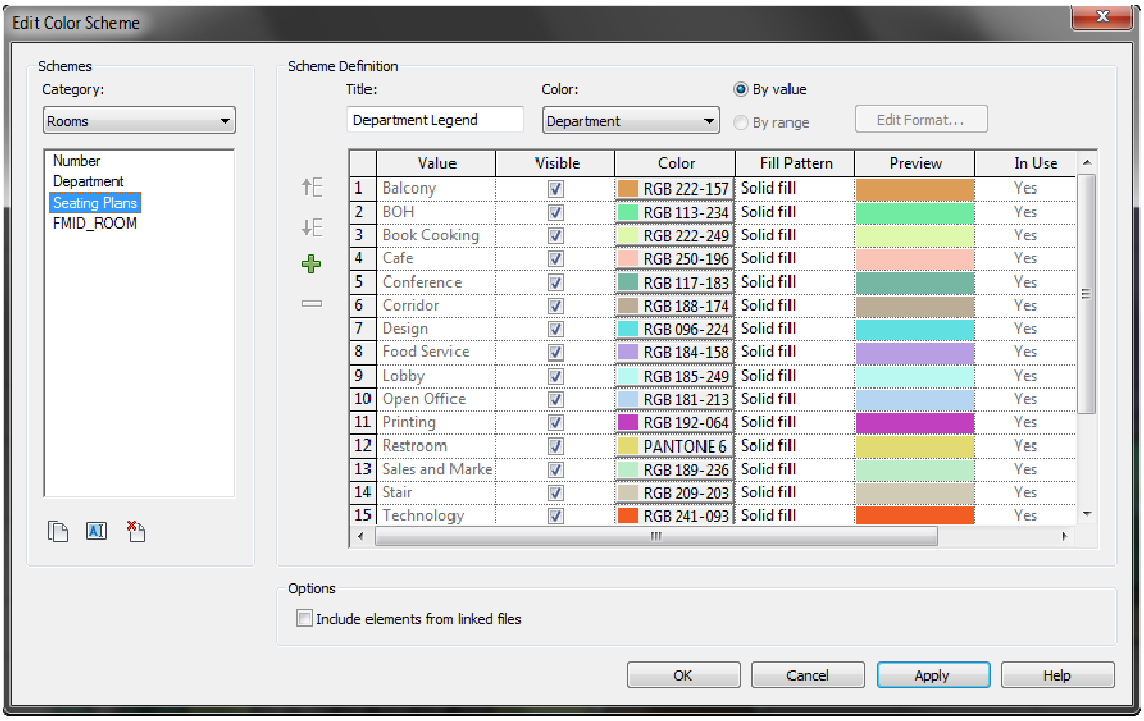
Views configured to use “Shaded with Edges” export as raster when exported to DWF. Raster views are no good for FM use since the elements are not selectable and cannot return any data when selected in the DWF viewer.

Using the “Hidden Lines” option for 2D views will result in an exported DWF file comprised of selectable elements with all parameters and associated values accessible when selected in the DWF viewer.

Selectable Room Elements in an Exported DWF View

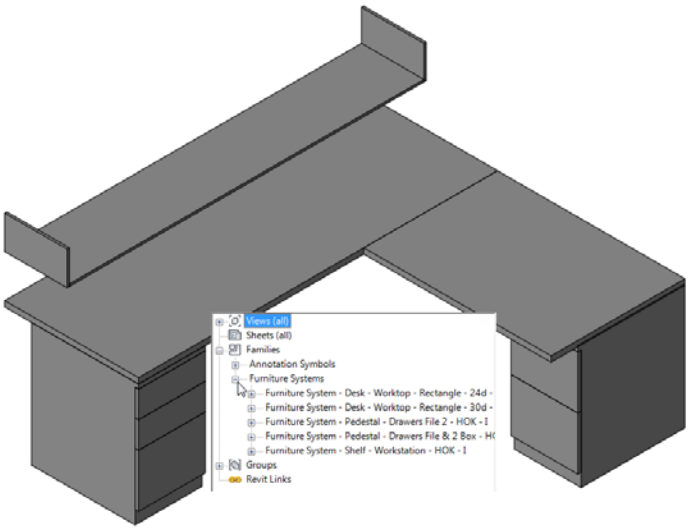
Since rooms are not true geometric elements in Revit, it can be tricky to export a DWF file where rooms are selectable and all of the room’s parameters and data are accessible from the element. I am aware of only one way to export a view from Revit 2011 where the room elements are selectable with all parameters and values.

The trick is to first create a "Color Scheme" for the Rooms category and to apply it to the views you wish to export for FM purposes. After a color fill has been applied and values have been entered into the parameter configured to drive the color scheme, an exported DWF file will contain selectable rooms.



Use Caution When Using Nested Families

Families that contain nested families can present a problem when exported to DWF format from Revit. Data parameters and all associated values entered into a parent family will be lost when exported to DWF. Only data entered into route families are accessible in the exported DWF.



The image on the left shows a family comprised of several nested families assembled together representing a workstation. The worktops are separate families as well as the shelf and cabinets. Each of these families are then inserted into a new family and placed throughout the project.

Any parameter data entered into this parent family would be lost when exported as DWF from Autodesk Revit.

FM Feature Based Parameters (FMID)

A nice trick for handling element or category specific FM/WM features from within a web environment when an element is selected within an embedded DWF file is to implement what I refer to as “FMID Parameters.” FMID parameters are parameters named using a highly organized and structured nomenclature that you listen for and react to as elements are selected in the DWF viewer pane in your web environment.

As elements are selected and contain one of your FMID parameters, you can adjust the web interface accordingly or trigger the launch of a web page using the parameter and value as an argument to perform a task.

FMID Naming Strategies

Having clean, structured, and predictable data to work with definitely improves programmatic efficiency. For instance if each of my FM task specific parameter names start with “FMID_” and end with a four character descriptor it becomes easier to write code to work with.

I've added a few basic FMID parameter examples in the sample Revit models to illustrate how these parameters can be used to trigger specific reactions in the web environment.

FMID_SEAT

This parameter is used to identify and query employees by seating assignment. Data entered into this parameter for chair elements coincides with the tenant's workplace seating assignment addresses or network port ID's.

FMID_STRG

This parameter triggers a query of all associated records stored in an object as inventory. Storage cabinets, lockers, and giant warehouse shelving are all perfect candidates for an FMID_STRG parameter implementation.

FMID_ROOM

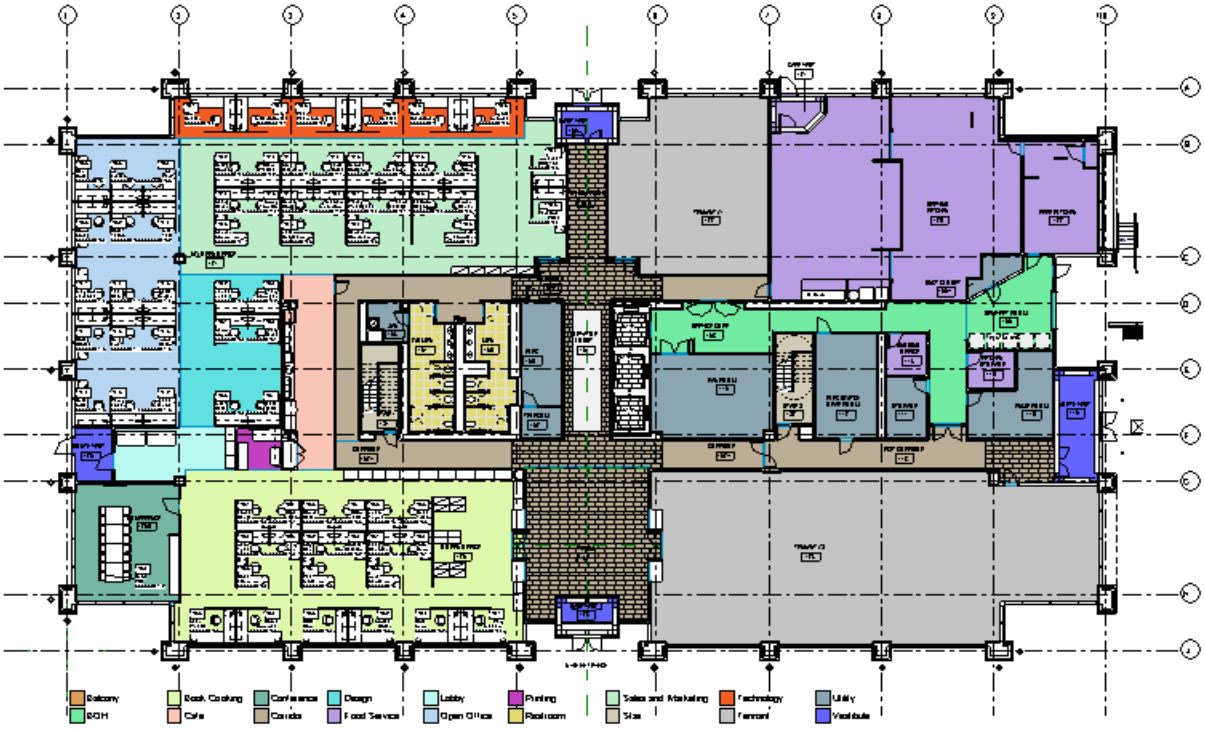
The FMID_ROOM parameter is used to identify ownership of a room that can then be queried and tied back to tenant charge back costs among several other purposes.

Sample FM View Types Provided in the DWF

Two view types have been configured within the sample model to illustrate the web site API functionality outlined later in this document, one being 2D and the other being 3D.

2D Floor Plans

A 2D plan designed as an employee seating plan is used to illustrate how a DWF view can be queried and searched from within a web page. A color fill has been applied to allow for selectable rooms in the DWF file.



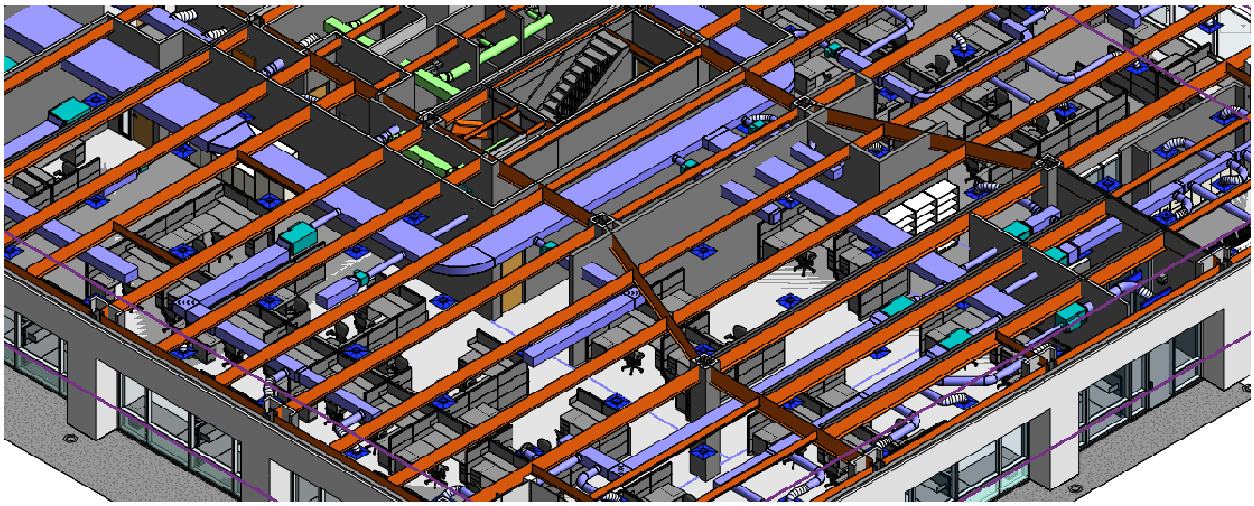
The FMID_SEAT parameter is bound to the Furniture category and used to enter seating numbers. A furniture schedule has also been provided to show employee names and phone numbers for quick reference. The floor plan and schedule have been dragged onto a sheet where the sheet is among the selected views exported to the DWF file.

3D Axonometric Floor Plans

A 3D axonometric view has been included for the visualization of plenum space elements such as VAV units and ductwork. 3D views are also helpful to maintenance personnel preparing to

administer maintenance to objects above the ceiling. All of the same parameters and values are accessible when selected in the viewer for 3D views as 2D views.

If you decide to include light fixtures in your 3D axon views, be sure to turn off the ceilings as they will make it very difficult to see most of the key elements in the viewer.



Exporting the DWF File

The DWF file can be exported to include both 2D and 3D model views in a single file. Larger and more complex facilities should probably be broken up into smaller DWF files in order to speed up the load times for the DWF files in the web environment.

Exporting DWF/DWFx is done from the "Export" button accessible from the application menu.

Revit API Sample "DynamiLink"

DynamiLink is a Revit 2011 API application that provides a bidirectional link between 32 bit Microsoft Access 2007 and 64 bit Autodesk Revit 2011.

The source code was written in Visual Basic using the freely downloadable Visual Studio 2010 Express IDE available at http://www.microsoft.com/express/downloads/. You can convert the classes to C# if you like at http://www.developerfusion.com/tools/convert/vb-to-csharp/.

Is This Your First Revit API Application?

If you have never built an Autodesk Revit 2011 addin, you may want to visit Jeremy Tammik's blog for an introductory webcast to the Revit API at http://thebuildingcoder.typepad.com/blog/2010/07/devtv-addin-templates.html.

Microsoft Access 2007 Database Schema

Two distinct table scopes are employed in the sample database. Non Revit synchronized tables or "General Data" will have their table names prefixed with "tbl." Tables that are designed to synchronize with the Revit model (Revit Synchronized Tables) will have table names prefixed with either "inst" or "type" for instance and type elements respectively.

Non Revit Synchronized Tables (General Data)

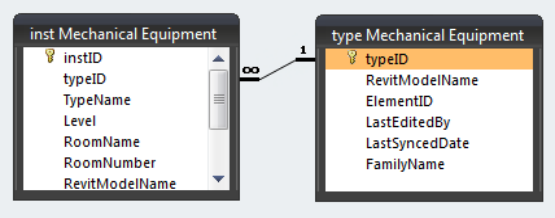
Data not being synchronized with the Revit model will be referred to as “General Data.” Data such as people, general facility documents, monthly energy data, work orders, etc. would all be considered "General Data" since they are not synchronized directly with a model element.

Revit Synchronized Tables

The Revit synchronization tables are designed to emulate Revit’s native data structure. Each Revit category is represented by both a type and instance data table where their respective elements and parameters will be stored.

Instance elements and type elements in Revit are managed separately behind the scenes in Revit. Each and every instance of a family is a unique element containing its own GUID while each and every unique type element also contains its own GUID. In the sample database, instance elements are managed in one table and related to their type elements in another table.

Each element’s GUID (not the ElementID) serves as the primary key in each table. The instance records are related to the parent type element record by the type element’s GUID. An example data relationship between the type and instance element is shown to the right for the "Mechanical Equipment" category.



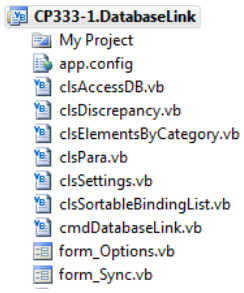
Creating and Configuring the Revit Synchronized Tables

There is no need to manually create these tables since the Revit API sample application "DynamiLink" will create and configure them for you when selected for synchronization.

Application Source Code Breakdown

The whole application was built using 8 custom classes and two forms. Open the Visual Studio 2010 solution file “AU2010_CP333-1_DynamiLink.sln” in Visual Studio 2010.

Refer to the Visual Studio 2010 solution file for the full application source code as only key code snippets will be listed in this document.



The cmdDynamiLink Class

This class serves as the application command entry point into the Revit environment. First this class constructs the clsSettings class used to hold in memory the Revit model document along with all general application settings and properties.

This class also serves constructs the dialog box for selecting categories and parameters of which to synchronize with the database.

Form_Options Category and Parameter Selection Form

This class first compiles a list of categories in the model excluding tag related categories.

As the categories are added to a running list, so are the custom parameters bound to that category. The ParametersBoundByCategory function retrieves a complete list of parameters bound to a category on the project level.

```
Private Function ParametersBoundByCategory() As Boolean
    ' Binding Map
    Dim bindingsMap As DB.BindingMap
    bindingsMap = m_Settings.Document.ParameterBindings
    ' Definition Map
    Dim iterator As DB.DefinitionBindingMapIterator
    iterator = bindingsMap.ForwardIterator
    ' Construct the list variables
    m_CategoryParameters = New List(Of String)
    m_ParamList = New List(Of String)
    ' Example to Add a few built-in parameters
    m_CategoryParameters.Add("Rooms" & vbTab & "Name")
    m_CategoryParameters.Add("Rooms" & vbTab & "Number")
    m_CategoryParameters.Add("Rooms" & vbTab & "Department")
    ' Cycle the whole binding map
    Do While (iterator.MoveNext)
        ' Get the binding object
        Dim elementBinding As DB.ElementBinding
        elementBinding = iterator.Current
        ' get the name of the parameter
        Dim definition As DB.Definition
        definition = iterator.Key
        m_ParamList.Add(definition.Name)
        ' add the category name.
        If (Not elementBinding Is Nothing) Then
            Dim categories As DB.CategorySet
            categories = elementBinding.Categories
            ' Get all parameters for the category
            Dim category As DB.Category
            For Each category In categories
                If (Not category Is Nothing) And category.Name.ToUpper <> "DETAIL ITEMS" Then
                    m_CategoryParameters.Add(category.Name & vbTab & definition.Name)
                End If
            Next
        End If
    Loop
    ' Alpha sort the category list
    m_CategoryParameters.Sort()
End Function
```

Clicking the OK Button

The database is not connected to or queried nor is the model data synchronized until the OK button is clicked. The first thing that happens is each selected category and parameter is collected into a list of clsElementsByCategory classes. The category object is retrieved from the category name stripped from the tree node name.

The type elements are collected separately from instance elements, the same goes for the list of type and element parameters. A snip from the clsElementsByCategory class constructor code collecting the elements and parameters into lists is illustrated to the right.

```
' Class Variables
Dim m_Doc As DB.Document = settings.Document
m_CategoryName = cat
m_ParamSelections = pList
' Get the category from the name
Dim m_Category As DB.Category = Nothing
For Each categoryTest As DB.Category In m_Doc.Settings.Categories
    If categoryTest.Name = cat Then
        m_Category = categoryTest
    End If
Next
' Type Elements
Dim typeCollector As New DB.FilteredElementCollector(m_Doc)
typeCollector.OfCategory(m_Category.Id.IntegerValue)
typeCollector.WhereElementIsElementType()
m_ElementTypes = typeCollector.ToElements
' Instance Elements
Dim instanceCollector As New DB.FilteredElementCollector(m_Doc)
instanceCollector.OfCategory(m_Category.Id.IntegerValue)
instanceCollector.WhereElementIsNotElementType()
m_ElementInstances = instanceCollector.ToElements
' Type Parameters
m_ParamType = New List(Of clsParamItem)
For Each x In m_ElementTypes
    For Each y In m_ParamSelections
        Dim testP As DB.Parameter = x.Parameter(y)
        If testP IsNot Nothing Then
            Dim mPara As New clsPara(testP)
            Dim iPara As New clsParamItem(mPara)
            m_ParamType.Add(iPara)
        End If
    Next
    Exit For
Next
' Instance Parameters
m_ParamInst = New List(Of clsParamItem)
For Each x In m_ElementInstances
    For Each y In m_ParamSelections
        Dim testP As DB.Parameter = x.Parameter(y)
        If testP IsNot Nothing Then
            Dim mPara As New clsPara(testP)
            Dim iPara As New clsParamItem(mPara)
            m_ParamInst.Add(iPara)
        End If
    Next
    Exit For
Next
```

Next the Microsoft Access database connection class is constructed and tested for a valid connection.

```
' Construct the Access Database class
m_AccessDB = New clsAccessDB(m_dbFileName, m_Settings)

' Make sure class constructor succeeds
If m_AccessDB.FailureStatus <> "" Then Me.Close()
```

Once the elements have all been collected into easy to manage lists and a valid database connection has been achieved, the search for discrepancies can begin.

The type elements are iterated first and queried to find matching records in the database. If no such record exists, it is added to the database no questions asked. In the event that a matching record is found, all fields and parameters are compared looking for mismatching data. Read-only parameters are updated to the database without asking the user.

Mismatching data items are stored in a clsDiscrepancy class where they can be dealt with later. This discrepancy class saves the current value of the database record field and the current Revit parameter value for the element. The clsPara class used to construct this discrepancy class contains an inherent reference to the element of which it was extracted from making it very easy to edit the parameter if and when necessary.

Form_Sync Discrepancy Handling Dialog

The discrepancy synchronization form is constructed and displayed only in the event that data discrepancies were found when the form_Options' OK button was clicked.

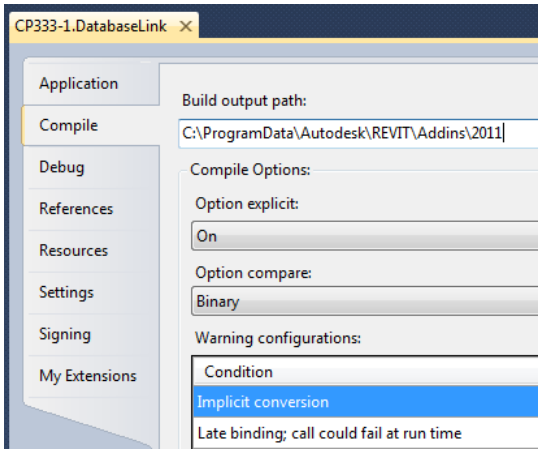
```
' Load the Synchronize form Only if discrepancies have been found
If m_AccessDB.Discrepancies.Count > 0 Then
    ' Construct and display the synchronization dialog
    Dim m_dlg As New form_Sync(m_Settings, m_AccessDB)
    m_dlg.ShowDialog()
Else
    m_AccessDB.CloseDB()
End If
```

This form presents all mismatching data items in a DataGrid view showing the current values for both the Revit parameter and database field. The user can then react to the discrepancies by

saving either the Revit data back to the database or vice versa on a single element parameter decision basis.

Debugging a VS2010 Project in Revit 2011

You will need to make a couple minor configurations to the Autodesk Revit 2011 configuration file and the Visual Studio development environment in order to successfully debug a Visual Studio 2010 application.

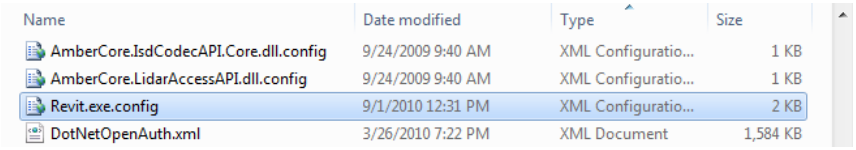


First open the “Compile” tab of your Visual Studio” project properties and set your “Build output path” directory to your addins folder. The example build output path shown to the right is from Windows 7.

Projects built for Autodesk Revit 2011 must be compiled targeting the Microsoft .NET Framework 3.5.

The Visual Studio 2010 default .NET Framework is 4.0 while the required .NET Framework for Autodesk Revit 2011 is 3.5. You will need to specify a target framework environment in the Revit 2011 configuration file so Visual Studio doesn’t try to debug .NET Framework 4 mode.

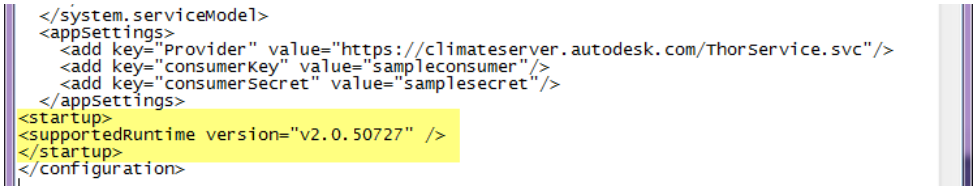
The solution is to edit the Revit.exe.config file to include a supported runtime version. Navigate to the Program directory beneath your Revit product and open the XML formatted file named “Revit.exe.config” in an ASCII text editor such as Notepad.exe.



Add the following three lines just inside the closing tag of “configuration” to allow debug support between Revit 2011 and Visual Studio 2010.

```
<startup>
<supportedRuntime version="v2.0.50727" />
</startup>
```

The image below illustrates these lines added to the file highlighted in yellow.



Setting the Debug Application for a Visual Studio 2010 Express Project

A target application must be set in order to debug a class application project with Visual Studio. The pro version has an easy way to enter this while the express versions of Visual Studio do not but that doesn't mean that it isn't possible.

Save and close your Visual Studio Express project and open the .vbproj or .csproj in an ASCII text editor such as Notepad.exe. The image below shows the two tags that need to be added to the project file circled in red with the header of which it was added highlighted in yellow.

```
    <TargetFrameworkProfile />
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <DefineDebug>true</DefineDebug>
    <DefineTrace>true</DefineTrace>
    <OutputPath>bin\Debug\</OutputPath>
    <DocumentationFile>HOK.SheetManager.xml</DocumentationFile>
    <NoWarn>42016,41999,42017,42018,42019,42032,42036,42020,42021,42022</NoWarn>
    <StartAction>Program</StartAction>
    <StartProgram>C:\Program Files\Autodesk\Revit Architecture 2011\Program\Revit.exe</StartProgram>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <DebugType>pdbonly</DebugType>
    <DefineDebug>false</DefineDebug>
    <DefineTrace>true</DefineTrace>
    <Optimize>true</Optimize>
```

Installing the Add-In into Autodesk Revit 2011

An addin manifest file has been provided (AU2010_CP333-1_DynamiLink.addin) in the dataset and should be moved into one of your two local Revit 2011 addin directories. The location of this directory varies depending on your operating system.

Windows 7 and Vista

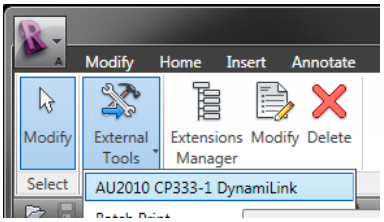
- C:\ProgramData\Autodesk\Revit\Addins\2011
- C:\Users\%USERPROFILE%\AppData\Roaming\Autodesk\Revit\Addins\2011

Windows XP

- C:\Documents and Settings\All Users\Application Data\Autodesk\Revit\Addins\2011
- C:\Documents and Settings\<user>\Application Data\Autodesk\Revit\Addins\2011

Running the Add-In

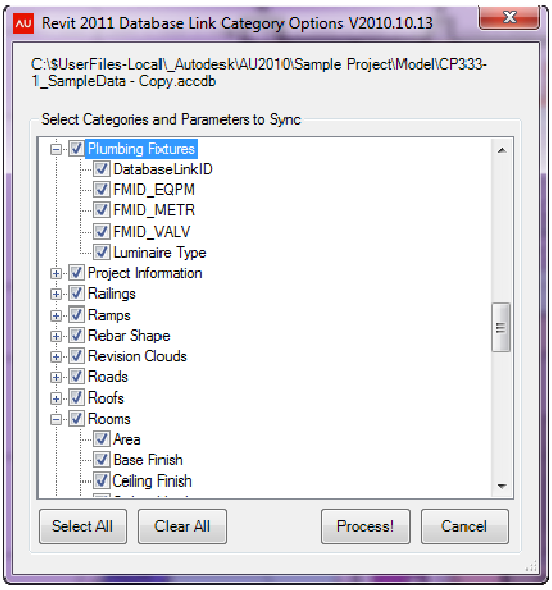
The addin utility is accessible from the Add-Ins tab of the ribbon under "External Tools." Pick the command named "AU2010 CP333-1 DynamiLink."



You will first be prompted to select a Microsoft Access 2007 database for synchronization. Select the sample database provided at "…\Web\App_Data\CP333-1_SampleData.accdb" and click open in the database browse dialog box.

Selecting Categories and Parameters to Sync

After a database has been selected, the “Database Link Category Options” dialog is displayed where you can select all categories and their associated parameters that you wish to synchronize with the selected database.

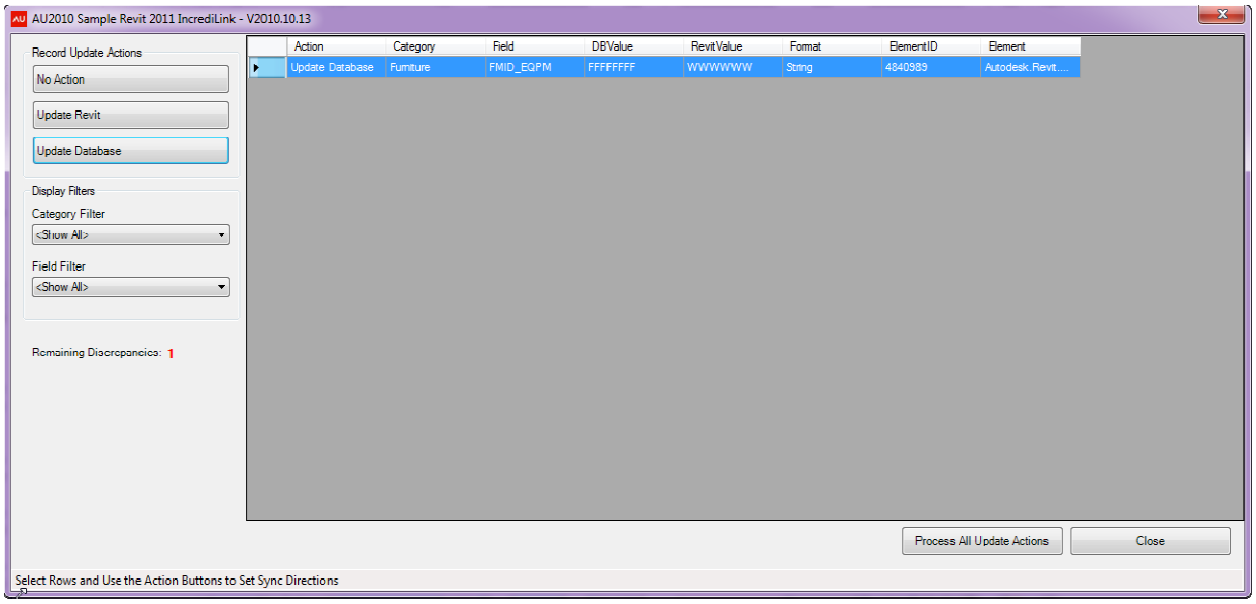


Only selected categories and parameters will be synchronized with the database. A “Select All” and “Clear All” button has been provided to quickly select all categories and parameters with a single click.

After you’ve selected all of the categories and parameters that you want to sync, click the “Process!” button.

Resolving Discrepancies

As the elements are scanned and compared to the database records, a collection of mismatching discrepancies are gathered and then bound to a data grid view in the form below.



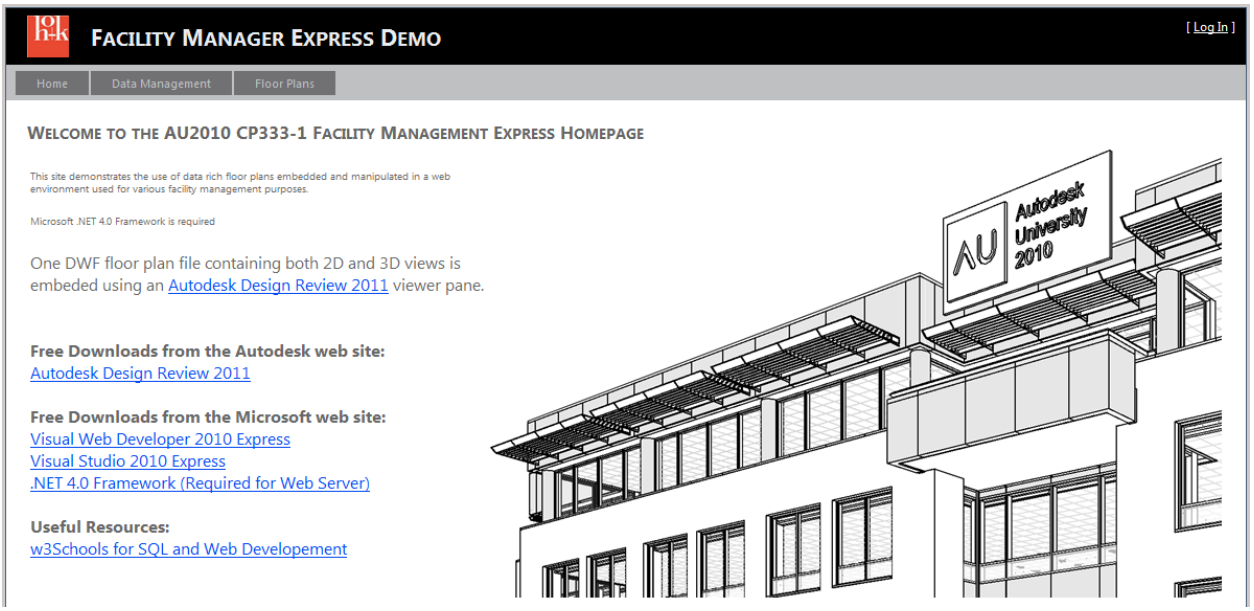
Select rows on the data grid view and use the action buttons along the upper left corner of the dialog to set the data acceptance directionality for each discrepancy record. The changes are made to both the Revit model and the database respectively when the “Process All Update Actions” button is clicked.

The Sample Web Site

Visual Web Developer 2010 Express was used as the development environment for the sample site and is a free download available from www.asp.net/downloads. Visual Web Developer 2010 Express supports both VB.NET and C#, with the sample code for this site being written in VB.NET and uses Microsoft .NET Framework 4.

Use the "Open Web Page" option in "Visual Web Developer 2010 Express" to open the site after it has been unzipped onto your local machine.

The site is comprised of a few pages designed to convey the general capabilities for using the Autodesk DWF file format in conjunction with an ASP.NET web site for general FM purposes.



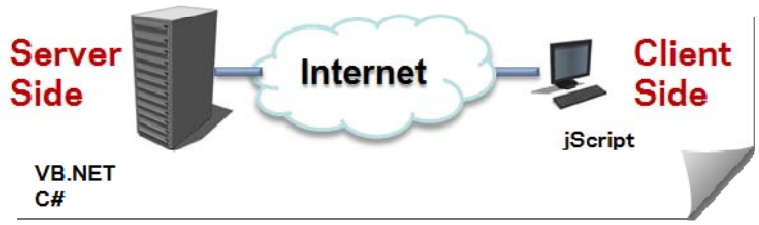
A “Floor Plans” page has been designed that supports both 2D and 3D DWF views and generates task specific commands as elements are selected in the DWF viewer. This page also demonstrate how server side and client side code can be bridged together to solve commonly perceived limitations.

The “Data Management” page demonstrates how data can be added, edited, deleted, and copied in an easy to use efficient web form.

Please do not use the sample web site for any security configuration reference. The security used is minimal and not configured in a reliable way.

Introduction to Client and Server side code

I will refer to "Client Side" and "Server Side" code quite a bit in this document. It is important to know the differences between these two types of code in a web environment.



Client Side Code

Client side code refers to code that runs on the client's machine and has no programmatic access to the server environment or file structure. The client side language used in the sample floor plan page is jScript. This jScript code (jScript and JavaScript are basically the same exact language) is used to communicate with the embedded Autodesk Design Review viewer since the viewer technically does not exist on the server at all. In fact, if the machine attempting to view the page that references the DWF viewer does not have Autodesk Design Review installed, it must first install it prior to the page ever loading and rendering on the client machine.

JScript code is case sensitive in its use of data variables and cannot be used to communicate with a database residing on a web server.

Server Side Code

Server side code refers to code that executes on the server and is required to communicate with a database or any other file residing on the web server. The server side code used in the sample web site is VB.NET. Server side code cannot communicate to objects embedded in pages that run client installed applications such as Autodesk Design Review 2011.

Floor Plan Page

The floor plan page demonstrates how to embed a DWF file into an ASP.NET web page and use both client side and server side code to query elements in the viewer as well as switch between views within the DWF file.

Sharing Program Data between Client and Server Side Code

Sharing programmatic data between client and server side code can be a bit of a challenge since they are completely different programming languages running isolated from one another. The data sharing method used in the floor plan page utilizes string based parameter passing by HREF query strings to send data from the client side code to the server side database queries.

- www.webpage.com/mypage?***Parameter1=Value1&Parameter2=Value2***

Embedded Frames (how client and server side code is bridged)

Two frames are used in the floor plan page to demonstrate how server side and client side code can be bridged together.

The command frame located on the left hand side of the page dynamically displays commands and related data to elements as they are selected in the DWF viewer pane. The element selection event sends the data as an HREF path containing parameters to then read and react to in the server side code.

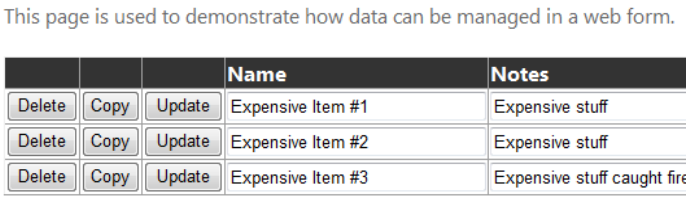
The employee listing pane on the right hand side of the page displays the employee assigned to each specific chair. As chairs are selected in the DWF viewer pane in either 2D or 3D mode the assigned user is queried from the database using the value of the FMID_SEAT parameter. The resulting employee is then isolated and displayed in the frame.

There is also a button beneath each employee's name and phone number that when selected will zoom to the seat that they are assigned to. The FMID_SEAT parameter is the data field queried and compared between the database and the DWF file.

Data Management Page

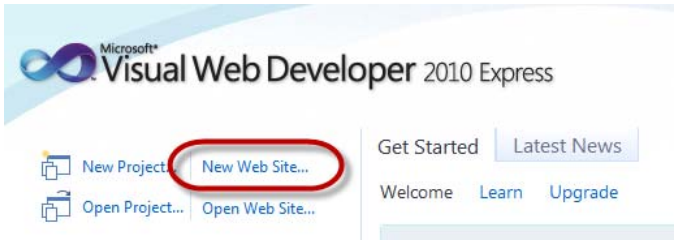
The data management page is designed to demonstrate how data in the Microsoft Access database can be managed through a fast and easy to use web page interface.

The sample is designed around the idea of managing facility inventory. Records can be updated, added, and copied from this page instantly without any noticeable page reload times whatsoever.

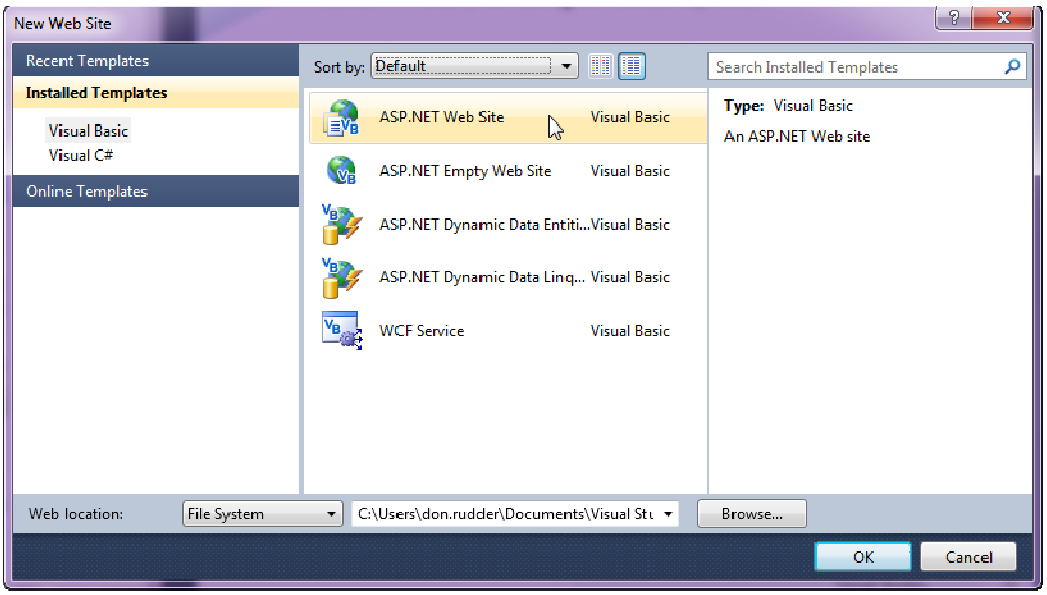


Creating the Site from Scratch

Create a new site by picking "New Web Site" button in the "Visual Web Developer 2010 Express" startup page.



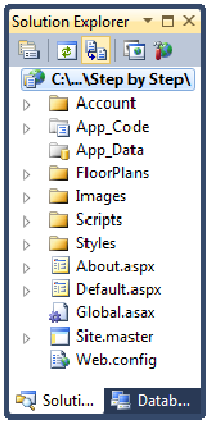
Select “ASP.NET Web Site” as the template and chose a directory to save the site and click the OK button.



The resulting template web site will include a “Site.master”, all necessary header and footer place holders as well as a "default.aspx" page. The site will also be automatically populated with a site.css file saved into the Styles directory as well as a prebuilt "Account" database to handle secure logins. I make no recommendations as to how secure the prebuilt "Account" database really is and recommend you research this in greater depth on your own if you ever plan to use it in your site.

Creating Required Directories

It is important to use the reserved directory names for certain objects in an ASP.NET web site in order to get dependable functionality. Standard reserved ASP.NET directory names are better explained on the MSDN web site at http://msdn.microsoft.com/en-us/library/ex526337.aspx.



With the basic foundation for the web site now in place, directories beneath the main web site directory need to be created. The App_Code directory is a reserved directory name in ASP.NET and should be created to house our data access layer (DAL) outlined later in this document. Create the following three directories beneath the web site's route directory:

- App_Code\DAL\
- FloorPlans\
- Images\

Editing the Site.master Object

The Site.master object provides an easy way to manage a consistent navigation scheme and manages repetitive elements in your site. The Site.master employs a “ContentPlaceHolder” object where properties and locations of page sections can be referenced by your content pages. Your aspx pages can reference the repetitive elements such as menu layout, logo and title, login information, header and footer layout, etc. from this Site.master object.

Adding a Logo and Setting the Title

The title can be modified from within the “header” class in the “title” div tag. I also added a logo just in front of the title in the sample site. The HTML code is illustrated below between the h1 tags.

```
<div class="header">
    <div class="title">
        <h1 style="vertical-align: middle">
            <img alt="logo" style="margin-bottom: 7px; margin-right: 15px; margin-top: 7px; width: 40px;
                vertical-align:middle" src="Images/Logo.jpg" />
            Facility Manager Express Demo</h1>
    </div>
```

Adding Menu Items

A prebuilt menu system is also already in place from the template, so we can just modify and copy the MenuItem tags already there.

```
<asp:Menu ID="NavigationMenu" runat="server" CssClass="menu" EnableViewState="false" IncludeStyleBlock="false" Orientation="Horizontal">
    <Items>
        <asp:MenuItem NavigateUrl="~/Default.aspx" Text="Home"/>
        <asp:MenuItem NavigateUrl="~/FacilityData.aspx" Text="Manage Data">
            <asp:MenuItem NavigateUrl="~/FacilityDataPeople.aspx" Text="Manage People" />
            <asp:MenuItem NavigateUrl="~/FacilityDataMeters.aspx" Text="Energy Meters" />
            <asp:MenuItem NavigateUrl="~/FacilityDataInventory.aspx" Text="Inventory" />
            <asp:MenuItem NavigateUrl="~/FacilityDataWorkOrders.aspx" Text="Work Orders" />
        </asp:MenuItem>
        <asp:MenuItem NavigateUrl="~/FloorPlan.aspx" Text="Floor Plan Views">
        </asp:MenuItem>
    </Items>
</asp:Menu>
```

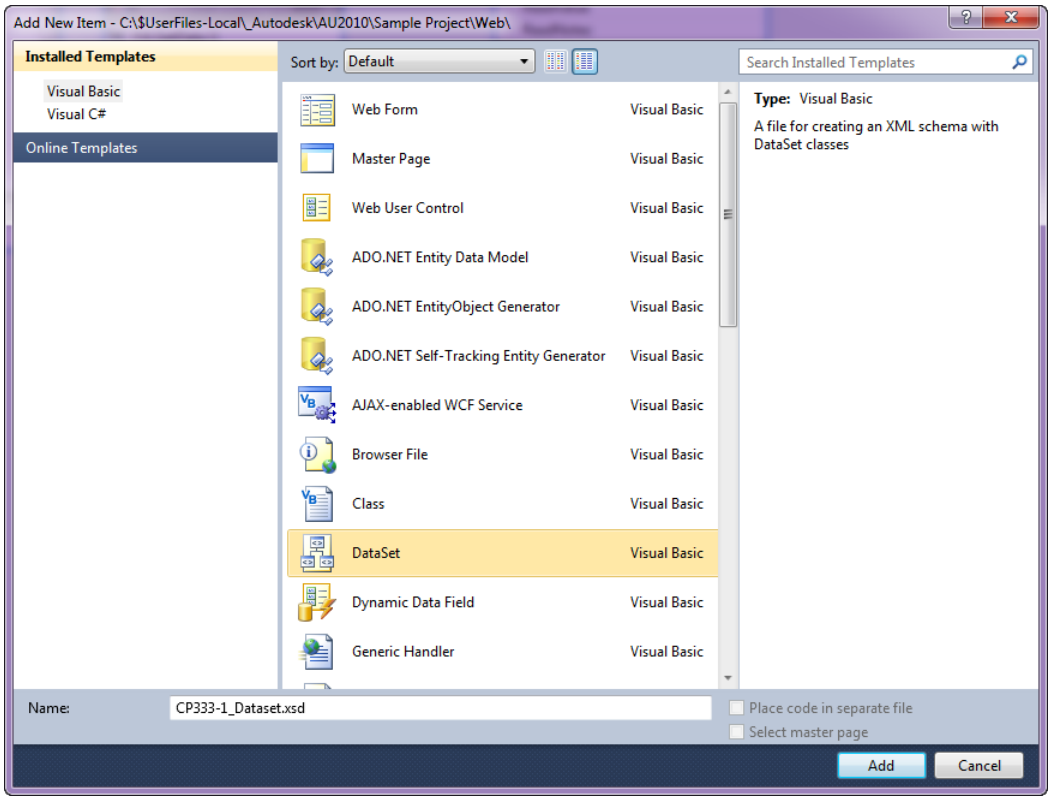
You can create a fly out effect by nesting MenuItem objects within each other. The example HTML code above nests several facility data pages within the main FacilityData,aspx page. The result is a dropdown menu effect when you hover on the route “Manage Data” link on the main menu (extra pages not provided in the sample site) shown in the image below.

Setting Fonts and Colors (CSS)

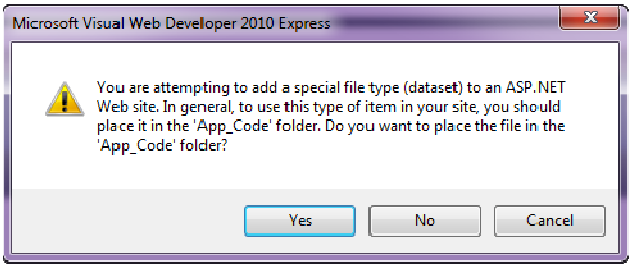
Site fonts and colors are managed by the Site.css file stored in the “Styles” directory. You can find numerous examples and tricks for editing cascading style sheets (CSS files) all over the interweb.

The Dataset Object

The Dataset object makes connecting to an external database easier than ever. It is possible to connect multiple databases of various formats (SQL or Microsoft Access) within one common and easy to use dataset object offering a simplified means for interacting with a database without much knowledge of SQL command syntax.

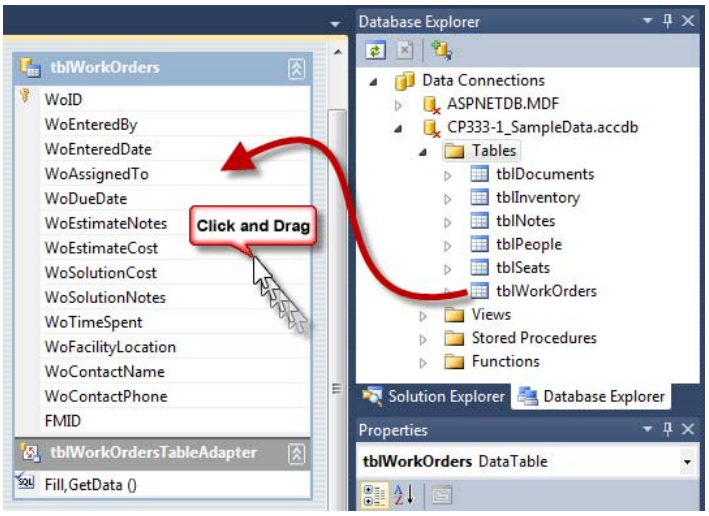


Choose “Yes” when asked to add the dataset object to the “App_Code” directory.



Adding the Database and Creating TableAdapters

TableAdapters provide communication between the web pages and database by executing queries or stored procedures, and either returning a new data table populated with the returned data or fills an existing DataTable with the returned data. TableAdapters are also used to send updated data from the web page back into the database.

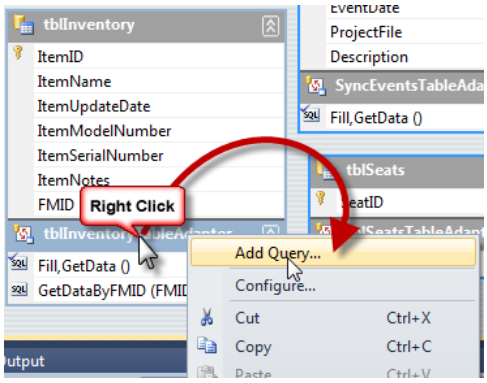


To create a TableAdapter, drag tables from the database viewer adjacent to the solution explorer in Visual Web Developer 2010 Express onto the data table configuration view. The image below illustrates a table being dragged from the Database Explorer onto the Dataset.

Adding Custom Queries to a TableAdapter

It is easy to add your own custom add, update, and delete queries to a TableAdapter by right-clicking on a TableAdapter’s header and selecting “Create Query.”

Each custom query is then accessible by calling its automatically generated command function within the TableAdapter object making coding in the web pages very clean and simple. There is no limit to the number of TableAdapter queries you can add so long as they all conform to the same schema.

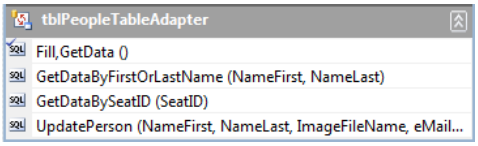


Adding Custom Wildcard Search Queries to TableAdapters

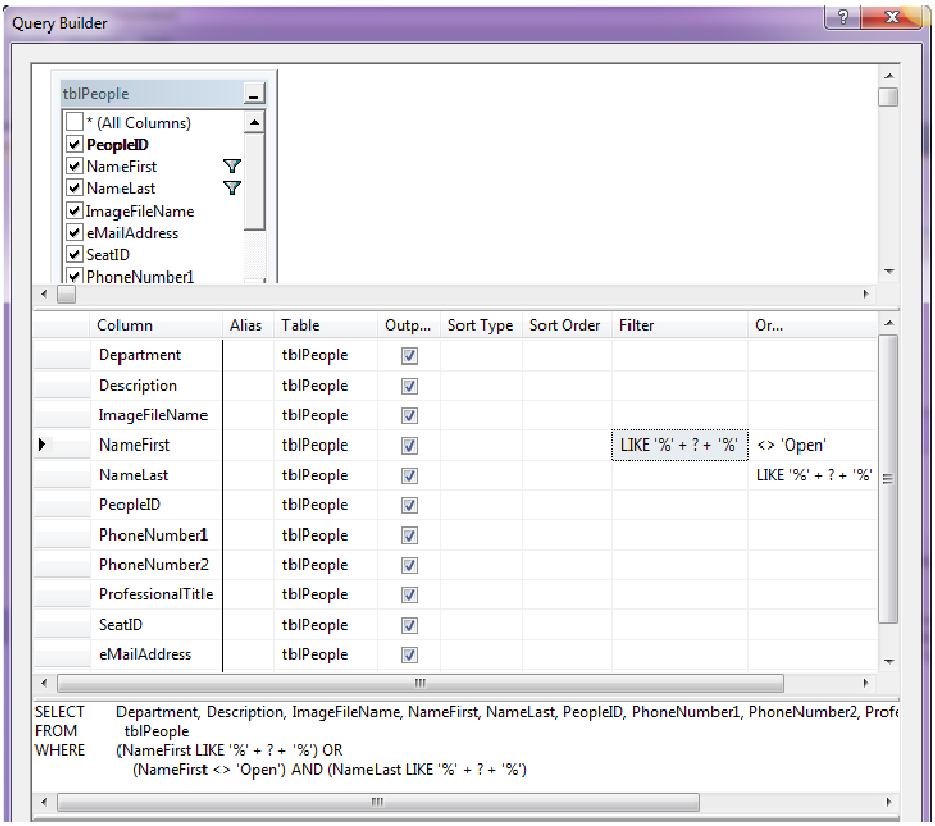
You will often need to search for a record where you may not have the full name as search criteria; you will need to build a wildcard search query. This type of query can search multiple

fields for a value and if it finds a match in either of the fields can return the matches. This especially comes in handy when searching people names.

The function named “GetDataByFirstOrLastName” in the tblPeopleTableAdapter implements a wildcard query. Creating a query within your table adapter that uses a wildcard search on one or more parameters is quite easy to configure.



The image below shows the “GetDataByFirstOrLastName” wildcard query in the query builder. Notice the filter entries for the fields “NameFirst” and “NameLast” as they both contain filters of “LIKE ‘%’ + ? + ‘%’”



This above query and filter avoids returning records with the word “Open” as a value in the “NameFirst” field as these records are open seats and not necessarily a person. Anywhere a “?” is entered as a filter will result in that field being a required parameter argument to the query and required by the TableAdapter object.

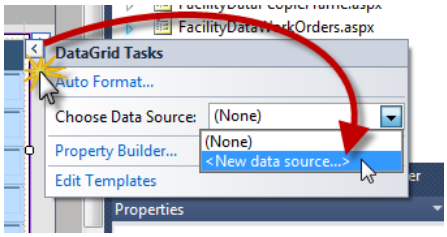
The “%” character works similar to how an asterisk works when searching for file names in w windows file search. Refer to http://www.w3schools.com/sql/sql_wildcards.asp for more information on wildcard symbols used in database queries.

Binding a Dataset to a DataGrid

One of the most convenient features of ASP.NET controls is how easy they are to work with and use to present information. A DataGrid is used in the “Inventory” page and is rendered by requesting a query from a TableAdapter and then just simply binding the results to the DataGrid control. Displaying the data does not require iteration or looping through any data records whatsoever and if the data changes in the query, the data will change in the DataGrid.

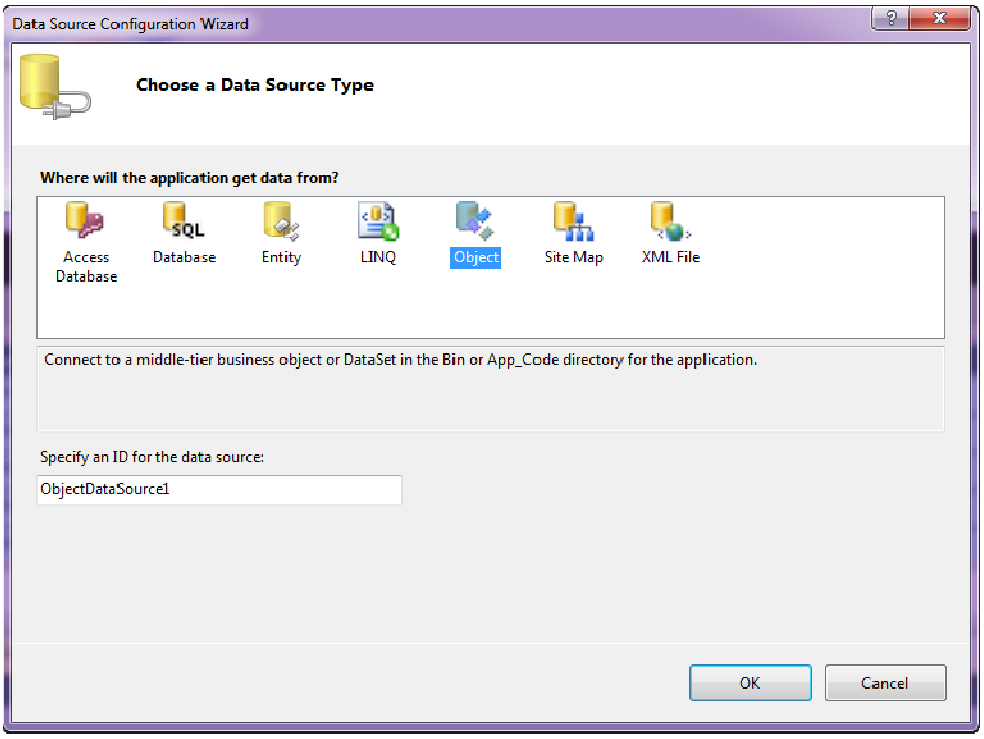
Add a DataGrid by entering the HTML code below where you want to place the control.

```
<asp:DataGrid runat="server" ID="Inv1" AllowSorting="True" AutoGenerateColumns="False"></asp:DataGrid>
```

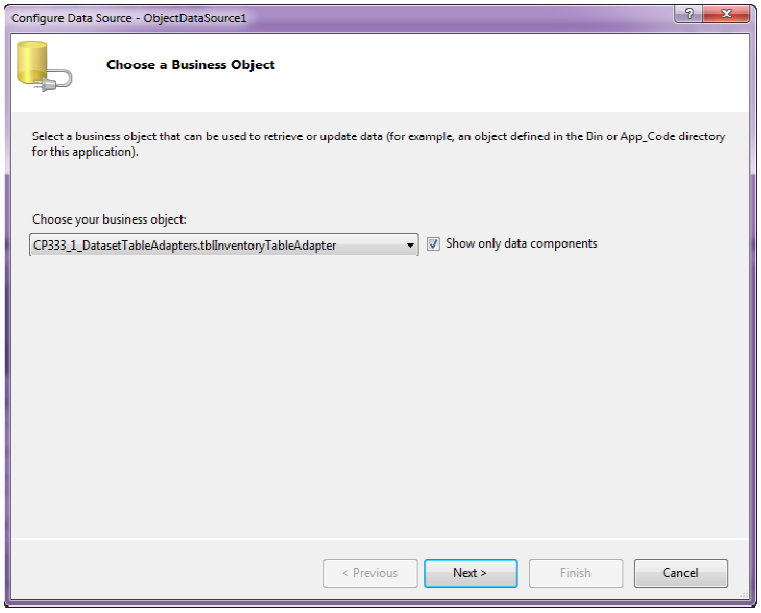


After the DataGrid has been added, select it in design view and click the edit arrow button shown to the left. Then in the DataGrid tasks menu, select “< New data source…>” in the “Choose Data Source” pull down.

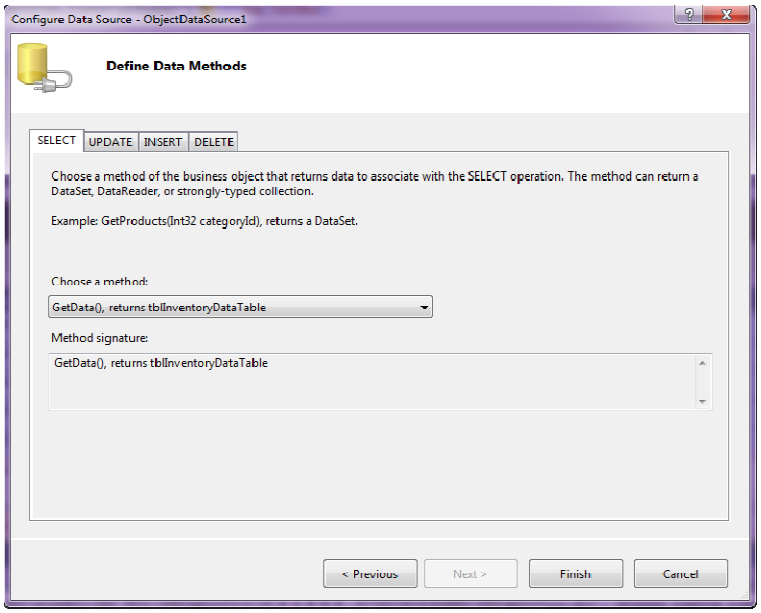
Then when prompted to select a data source, choose “Object” shown below.



You will then be prompted to select a data component which in this case will be one of the TableAdapter objects we created earlier in this document. It is important that a TableAdapter already exist in order to complete this step, otherwise you will not be able to finalize the data object configuration.

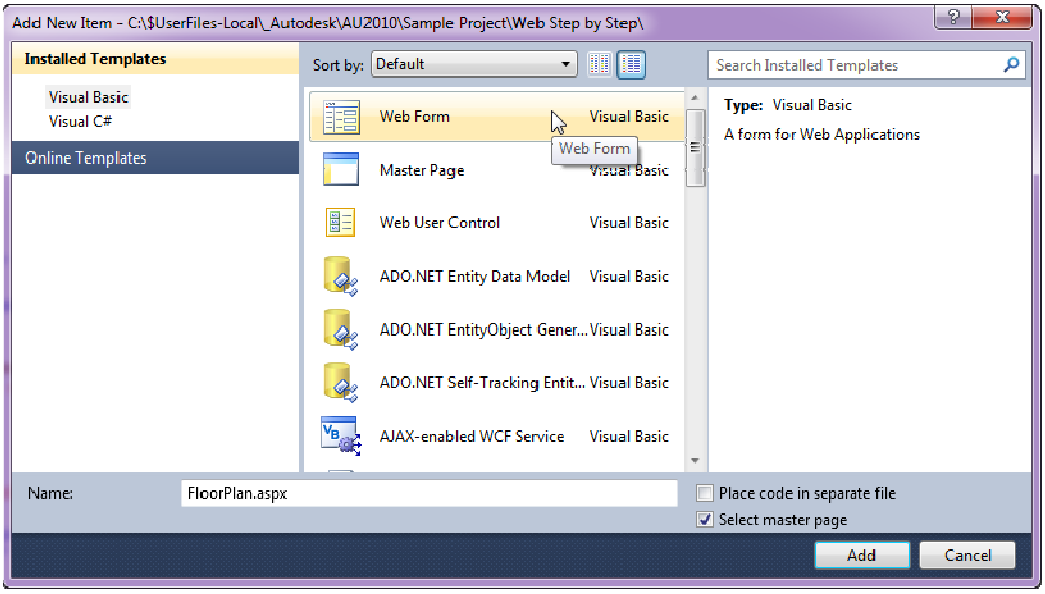


After a data object has been selected, you can choose a data method. Accept the default GetData() method and click the Finish button to continue.



Embedding Design Review 2011 and DWF Floor Plans

Copy the DWF file into the web site's "FloorPlans" directory and create a new "Web Form" object (CTRL + SHIFT + A) named "FloorPlan.aspx." Be sure to check the box in the lower right for "Select master page" so you can take advantage of the pre built header and menu system.



This page will be used to implement the DWF floor plans by embedding the DWF file into an HTML <object> tag. The case sensitive name entered into the ID attribute of the <object> tag will be used to programmatically access the viewer using “Client Side” code. The HTML <object> code used to embed the DWF viewer is shown below.

```
 <object id="ADViewer" width="100%" height="800px"
codebase="http://www.autodesk.com/global/expressviewer/installer/DWFViewerSet
up.cab"
          classid="clsid:A662DA7E-CCB7-4743-B71A-D817F6D575DF">
          <param name="src" value="./FloorPlans/CP333-1_Plans_2D.dwfx" />
          <param name="ObjectSelectedColor" value="255" />
          <param name="ObjectHighlightColor" value="5" />
 </object>
```

Two key DWF viewer events are implemented in the site and used to muster up just about all of the functionality between the web interface and the DWF viewer.

Event - OnEndLoadItem

This DWF viewer event fires off after the page loads into the browser. This is where you should set your viewer configuration commands such as SELECT or any element selection color settings you require.

Event – OnUpdateUiItem

This is the event that I use to perform just about all of the main functionality within the site. This event fires off as an element is selected in the viewer. This event can gain you access to all of the element property names and values that you can then use to drive the web page.

Locating and Zooming to Elements by Property

The ability to locate and zoom to elements in a DWF file by a parameter value is critical for several types of operations. The sample used in the site that implements this functionality is the “ZoomToDwfxPropertyValue” function used in the “FacilityDataPeopleFrame.aspx” page that locates and zooms to an element with a matching FMID_SEAT parameter value in the DWF file.

```
// Zoom to a value within a DWF on the parent page
function ZoomToDwfxPropertyValue(PropertyValueStr) {
    // Get the viewer in the parent document
    var ADViewer = document.parentWindow.frameElement.ownerDocument.getElementById("ADViewer");
    var ECompViewer = ADViewer.ECompositeViewer;
    //What level is the seatID, Set the Level prior to zoom attempt
    var c = PropertyValueStr.charAt(0);
    if (c == '1') {
        OpenSection('Sheet: SP-101 - SEATING PLANS LEVEL 1');
    }
    if (c == '2') {
        OpenSection('Sheet: SP-102 - SEATING PLANS LEVEL 2');
    }
    if (c == '3') {
```

The code snippet above illustrates how the DWF view is changed depending on the first character of the FMID_SEAT parameter. In this sample, seat ID’s are preceded with the floor of which they are located on in the facility.

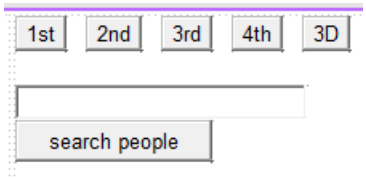
```
// Get the current view extents
var exts = ECompViewer.Section.Content.Extents(UserCollection);
var xDeltaO2 = (exts.Right - exts.Left) * 3.0;
var yDeltaO2 = (exts.Top - exts.Bottom) * 3.0;
var viewLeft = exts.Left - xDeltaO2;
var viewRight = exts.Right + xDeltaO2;
var viewTop = exts.Top + yDeltaO2;
var viewBottom = exts.Bottom - yDeltaO2;
// Get the section object
var Section = ADViewer.ECompositeViewer.Section;
var MyViewer = ADViewer.Viewer;
// Set the original view variables for use in animation calculation
var orig_l = MyViewer.View.Left;
var orig_r = MyViewer.View.Right;
var orig_b = MyViewer.View.Bottom;
var orig_t = MyViewer.View.Top;
// Animation calculation
for (var time = 0.0; time <= 1.0; time += (1.0 / 20)) {
    MyViewer.SetView(
            (orig_l * (1.0 - time)) + (viewLeft * time),
            (orig_b * (1.0 - time)) + (viewBottom * time),
            (orig_r * (1.0 - time)) + (viewRight * time),
            (orig_t * (1.0 - time)) + (viewTop * time));
}
// Cleanup final view
Section.SetView(viewLeft, viewBottom, viewRight, viewTop);
```

Once the element has been located, the zoom effect is calculated by requesting the extents of the element and transforming the current view relative to the centroid of the coordinates returned by the element’s extents.

An animation calculation is then provided so as to keep the display from getting lost as the transition is made from the current view over to the matching element.

The final line in the sample shown on the left sets the view to the final location of the located element.

People Search Box and View Commands



The five floor plan buttons upper right corner of the floor plans page are used to switch between views in the DWF file without having to refresh the entire page and load yet another DWF file. A text box and a search button just beneath it are used to locate employees in the seating plan.

The search button code shown below executes a query on the database for a first or last name matching the data entered into the text box by posting the search text to the “FacilityDataPeopleFrame.aspx” frame via HREF parameters and values. If the first character

entered in the search string is a number, it searches seat numbers; otherwise it searches for an employee name and displays the results in the people data frame beneath the search button.

```
// Find User by SeatID, post to data viewer
function findUsers(varUserName) {
    var c = varUserName.charAt(0);
    if (c >= '0' && c <= '9') {
        userDataFrame.location = 'FacilityDataPeopleFrame.aspx?seat=' + varUserName;
    }
    else {
        userDataFrame.location = 'FacilityDataPeopleFrame.aspx?user=' + varUserName;
    }
}
```

Switching Sections (Views) and Avoiding Postback

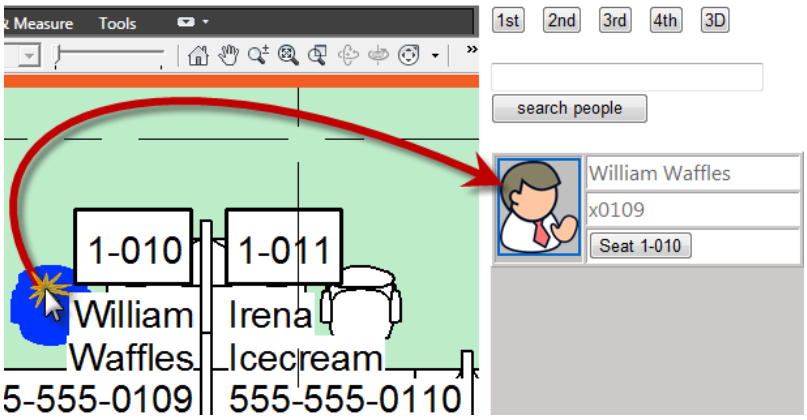
Navigating views (sections) within a DWF file is fairly straightforward and requires very little code. The function to the right requires one argument representing the view name as a string and activates it in the viewer. The view name must obviously exist in the loaded DWF file.

```
// Activate a section (view) within a loaded dwfx file
function OpenSection(SectionName) {
    var ADViewer = document.getElementById("ADViewer");
    var ECompViewer = ADViewer.ECompositeViewer;
    ECompViewer.Section = SectionName;
}
```

Adding the Frames

The final part of the site is definitely the most important as these two sample frames demonstrate the real efficiency and power of the site. These two frames are where the magic happens programmatically between the client side and server side code.

The Employee List Frame



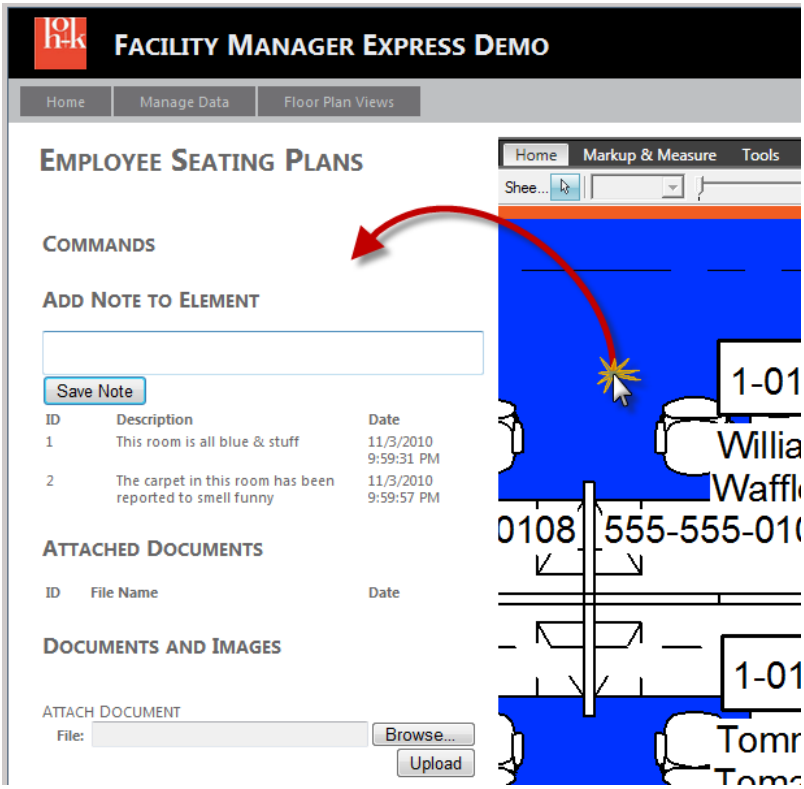
An employee list frame is embedded along the right hand side of the DWF viewer pane and displays an abbreviated data table showing each employee's name, photo, phone number, and a button that can be clicked to quickly zoom to their location in the DWF view.

When a chair is selected in the DWF viewer containing an FMID_SEAT parameter and value with an employee assigned to it, their information is queried and displayed in the employee viewer pane shown above.

The Command and Element Data Frame

A command and data frame is nested along the left hand side of the DWF viewer pane and reacts to elements as they are selected from within the DWF viewer pane. I've built in specific types of reactions based on the type of element that is selected by listening for specific parameter names and reacting to them when they are present in a selected element.

None of the sample FMID buttons in the “Command” section of the frame have any functionality built into them. These buttons are really only intended to demonstrate how the page interface can be manipulated based on data contained in selected items in the DWF viewer. I’ll leave it up to you to come up with your own ideas on how to utilize this functionality.



The image to the left shows how the interface adjusts as a room element is selected in the DWF viewer.

You can then add a note with the “Save Note” button as well as upload a document and have it related to the selected item.

Notes and documents related to the element are displayed when the element is selected in the viewer.

Some Additional Possibilities to Think About Not Covered in this Document

This document pretty much only demonstrated basic functionality between an embedded DWF file and an ASP.NET web site. Now that you know how to get around the obstacle of sharing data between server side and client side code, just about anything is possible

- Connect to your facility controls systems supporting BacNet
    - Lighting Controls, HVAC Sensors, Energy Meters, etc.
- Monitor web and security cameras by selecting them in a floor plan view
- Generate and manage work orders
- Query tenant records by selecting the rooms they rent on the floor plan
- Record and monitor maintenance for equipment and systems
- Manage warehouse inventory visually