Migrating //Build to DP4 Addendums

Working Copy, 18 January, 2012

This document serves as a ongoing repository for additional changes, migration issues, and migration advice for developers moving code from the Windows 8 Developer Preview (//Build) to the Windows 8 Developer Preview Update (DP4). Many changes are already covered in the migration guide issued with DP4 itself, but there are others that are not covered in that document.

# All Languages/WinRT

## Manifest Changes

There are many changes that might be necessary according to the change list. While you can go through items line by line, there are two approaches that can be quicker:

* For a simple manifest, compare what you have with one from a new project created in VS, since the changes will be minimal.
* For a more complicated project, create a new project in VS and use the manifest editor to add your desired features, then use that as a basis for any manual edits you might need. Alternately, you could just create a new project altogether and transfer your code and resource files.

## Windows.Data Namespace/JSON

This is an addendum to the section “Specifics of Windows.Data.Json namespace” (page 102) of the migration guide.

That section mentions that JsonObject has a contructor that accepts a string, which is not accurate. Instead, you first use **JsonValue.Parse** on a string, then call the value’s **GetObject** method. That is, change:

JsonObject root = new JsonObject(myData);

to:

JsonValue myValue = JsonValue.Parse(myData);  
JsonObject root = myValue.GetObject();

## AsyncInfoFactory to AsyncInfo

The AyncInfoFactory class has changed to AsyncInfo with a different pattern:

return AsyncInfoFactory.Create<string>(() => BeginGetDataAsync(query));

becomes:

return (IAsyncOperation<string>)AsyncInfo.Run((System.Threading.CancellationToken ct) => BeginGetDataAsync(query));

# C++

No addendums at this time.

# C#/VB/XAML

No addendums at this time.

# HTML/JavaScript/CSS

## IndexDB

* The IDBDatabase.setVersion method is removed.
* The signature of the IDBFactory.open method has a new version parameter to replace the setVersion method. The version parameter should contain a value of 1 or greater.
* If the version parameter is greater than the existing version, an upgrade event handler called **onupgradeneeded** will be triggered. If the value is the same, the database will be opened. If the value is less, an error event will be thrown.

Old code (simplified, with highlights showing impact):

function openDBTest(dbName) {

var rq = window.indexedDB.open(dbName);

rq.onsuccess = successOpenningDB;

rq.onerror = failureHandler;

}

function successOpenningDB(evt) {

var db = evt.target.result;

var rq = db.setVersion("1");

rq.onsuccess = successHandler;

rq.onerror = failureHandler;

}

function successHandler(evt) {

//create schema

}

New Code:

function openDBTest(dbName) {

var rq = window.indexedDB.open(dbName, 1);

rq.onsuccess = useDB;

rq.onupgradeneeded = successHandler;

rq.onerror = failureHandler;

}

function successHandler(evt) {

//create schema

}

## FlipView

The following API changes will be made to FlipView:

|  |  |
| --- | --- |
| **Property** | **Change/New name** |
| placeholderRenderer | N/A (removed) |
| itemRenderer | itemTemplate |
| dataSource | itemDataSource |

## ListView

ListView has extensive changes that unfortunately did not appear in the migration doc for the most part.

1. WinJS.UI.getControl/setControl methods have been removed in favor of a .winControl property that’s automatically added to the control element. So instead of WinJS.UI.getControl(element) just use element.winControl.
2. Selection API changes
   1. To make it easier to work with the selection, the listview.selection object now implements:

* Promise<items> getItems();
* array getRanges();
* array getIndices();
* bool isEverything();
* int count();
  1. To make the itemInvoked easier to work with, the eventObject passed to the event contains an itemPromise property (eventObject.details.itemPromise).
  2. To make drag & drop easier, the following events pass a set of items:
* dragitemsstart
* dragitems
* dragitemsend
* itemsmoved

These events will pass the same signature as the selection object.

The index of the insertion point will be passed to the dropitems and itemsmoved events.

* 1. To enable async updating of the selection, the selection object selectionchanging event has a new method setPromise. This method allows an app to signal that its changes to detail.newSelection are completed. If setPromise hasn’t been called ListView will assume that newSelection has been updated by the handler synchronously.

1. ReadyStateChanged event change
   1. Originally Readystatechanged was added for testing (Unit tests need to know if they can check stuff in the DOM) but people started using it for other scenarios as well.
   2. Ready state is currently used for the following scenarios:
      1. Restore state: Some method and properties (e.g. scrollPosition, scrollTo, selection) work only when ListView is in complete state so people wait until readyStateChanged event to do this. This is better solved by using the firstVisible property described below
      2. Deferring loading of content. Instead of using ReadyStateChanged, apps should be using the rendering pipeline.
      3. The readyState event is renamed to loadingState to prevent confusion with the existing HTML readystate event.
2. Scroll Position for hydration
   1. The scroll position property will not work correctly when rehydrating the app due to lazy group instantiation, so apps should use the firstVisible property which is changed from an async method to a property.
   2. Focus can be detected using the currentItem property which also has a corresponding currentItemChanged event
3. Refresh renamed to **forceLayout**
   1. Developers have tended to think that refresh needs to be called after the data source change. It actually only needs to be called in these cases:
      1. When ListView style.display property is switched from none to display
      2. When CSS rules styling an item template have changed due to change of active media query (e.g. switching between snapped and full screen layout).
   2. To avoid confusion with refresh after the data change the refresh method is renamed to forceLayout.
4. API changes: methods, properties, events, and CSS classes

The following API changes are being made (blanks in the second column mean no change):

|  |  |
| --- | --- |
| **Method** | **Change/New name** |
| addEventListener |  |
| dataObject | Removed |
| ensureVisible |  |
| firstVisible | indexOfFirstVisible |
| getElementAtIndex | elementFromIndex |
| getIndexFromElement | indexOfElement |
| group | Removed |
| groupCount | Removed |
| lastVisible | indexOfLastVisible |
| loadNextPages | loadMorePages |
| Refresh | forceLayout |
| removeEventListener |  |
| scrollTo | Removed |

|  |  |
| --- | --- |
| **Property** | **Change/New name** |
| automaticallyLoadItems | automaticallyLoadPages |
| crossSlide | swipeBehavior |
| dataSource | itemDataSource |
| editable | Removed |
| groupDataSource |  |
| groupRenderer | groupHeaderTemplate |
| itemRenderer | itemTemplate |
| Layout |  |
| loadingBehavior | (only changed the case of randomAccess) |
| pageLoadThreshold | pagesToLoadThreshold |
| pagesToLoad |  |
| readyState | loadingState |
| reorder | Reorderable |
| resetGroup | resetGroupHeader |
| resetItem |  |
| scrollPosition | Removed |
| selection |  |
| selectionMode |  |
| tap | tapBehavior |
| zoomableView |  |

|  |  |
| --- | --- |
| **Event** | **Change/New name** |
| dragitems |  |
| dragitemsend |  |
| dragitemsenter |  |
| dragitemsstart |  |
| dropitems |  |
| iteminvoked |  |
| itemscopy | Removed |
| itemsdelete | Removed |
| itemsmoved |  |
| itemspaste | Removed |
| readystatechanged | loadingStateChanged |
| selectionchanged |  |
| selectionchanging |  |

|  |  |
| --- | --- |
| **CSS class** | **Change/New name** |
| win-listView | win-listview |
| win-viewport |  |
| win-horizontal |  |
| win-vertical |  |
| win-scrollable | win-surface |
| win-progressBar | win-progress |
| win-item |  |
| win-hover | Use :hover |
| win-pressed |  |
| win-selected |  |
| win-groupHeader | win-groupheader |
| win-inTransit | win-dragimage |
| win-inTransitNumber | win-dragcount |
| win-inTransitOverlay | win-dragoverlay |
| win-selection-background | win-selectionbackground |
| win-selection-checkmark | win-selectioncheckmark |
| win-selection-hint | win-selectionhint |
| win-rtl |  |

## DataSource API Changes

The following changes have been made to the datasource API(s) to go with ListView:

1. ArrayDataSource removed from WinJS library
   1. ArrayDataSource has proved to be a very useful way to manage in memory data, however when it comes to editing the items, the ListDataSource API is not very discoverable and is overly complicated for the array case. It’s great for disconnected async data sources, but not ideal for an in-memory array.
   2. The simpler solution is WinJS.Binding.List which implements array like access and modification methods, and therefore should be easier to use for the common modification operations. Create a new Grid Layout app in Visual Studio and see how it uses this.
2. Arrays can no longer be directly passed to ListView or FlipView
   1. With the removal of ArrayDataSource, it will no longer be possible to pass an array directly to the ListView/FlipView controls
   2. The recommended is to use a Binding.List and call the dataSource method to create a datasource from it.
3. Renamed ListDataSource to VirtualizedDataSource
   1. The ListDataSource object is the kingpin for being able to create a custom datasource. The main scenario for creating a custom datasource is to enable data virtualization and direct interaction with the source of the data.
   2. Renaming the object will make it more obvious as to its primary purpose and lessen confusion between the purpose of the object & the listDataSource API.
   3. WinJS.Binding.List implements the ListDataSource API
   4. To use the VirtualizedDataSource, developers create an object that implements the DataAdapter API, and pass that to the VirtualizedDataSource.
4. In-Memory Grouping
   1. WInJS. Binding.List has intrinsic support for grouping and being able to offer up a datasource for groups.
   2. This should be the preferred way of doing grouping for array-type sources
5. Grouping for custom DataSources
   1. The groupDataSource does not scale well for virtualized datasources and so we are recommending that for grouping in those scenarios the developers should create a corresponding group data source that queries for the groups from the server/source of data
   2. This will require the app to implement two datasources:
      1. The items datasource should be as it is today, with the inclusion of a groupKey property on each item to indicate the group it is in
      2. The groups datasource should enumerate the list of groups, with all the data for the group header etc. It will need to have:
         1. A key field which matches to the groupKey on the items
         2. A firstItemIndex property that is the overall index of the first item in the group
6. GroupDataSource will be renamed to computeDataSourceGroups and exposed as a method off WinJS.UI
   1. There are a few scenarios where enumerating through the items is the best way to form the groups, and this implements that functionality.

var dataSource = WinJS.UI.computeDataSourceGroups(TestComponents.createTestDataSource(myData, controller, null), groupKey, groupData);

1. Smaller (potentially breaking) changes
   1. msDataItem is no longer exposed on the html elements for items in the ListView
   2. ListNotificationHandler.inserted & .moved now passes the itemPromise, rather than the item so that the client has the ability to retain the item.
2. Smaller Changes that should not be breaking
   1. ListDataSource API now has ItemFromIndex, ItemFromKey and ItemFromDescription methods so that they can be used to get a snapshot value without having to create an instance of the ListBinding.
   2. ListDataAdapter now has begin/end edits to mark a batch of edits so the datasource can optimize them if it wants to.
   3. ListDataAdapter can implement an itemSignature method which can be used to compute a signature of the item which will be used as the way of detecting if the item has been changed and therefore needs to be redrawn. This gives the datasource developer more knobs  to control the way the item comparison occurs.
   4. Custom DataSources can set the cache size for the VirtualizedDataSource as part of its construction
      1. It defaults to 0 if compareByIdentity is true
   5. Cleaned up the pattern for how datasources are declared, but the pattern for consuming them has not changed
   6. Reload Methods have been added to ListDataNotificationHandler and ListNotificationHandler to indicate that the data will all need to be updated and refreshed.

# Migrating an HTML/JS Project

This section contains notes on a more complete step-wise process for migrating an HTML/JS app. Some of the details here are already in the migration guide, but are repeated here in context of distinct migration steps.

## Migrating Project and Solution Files

Project file:

* Rename the project file from .wwaproj to .js proj.
* Near the top of the file, replace the following two lines

<Import Project="$(MSBuildExtensionsPath32)\4.0\Microsoft.VisualStudioVersion.v11.Common.props" />

<Import Project="$(MSBuildExtensionsPath32)\Microsoft\VisualStudio\v$(VisualStudioVersion)\WWAProject\Microsoft.VisualStudio.WWAProject.Default.props" />

<Import Project="$(MSBuildExtensionsPath32)\Microsoft\VisualStudio\v$(VisualStudioVersion)\WWAProject\Microsoft.VisualStudio.WWAProject.props" />

with the following:

<Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.props" Condition="Exists('$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.props')" />

<Import Project="$(MSBuildExtensionsPath32)\Microsoft\VisualStudio\v$(VisualStudioVersion)\$(WMSJSProjectDirectory)\Microsoft.VisualStudio.$(WMSJSProject).Default.props" />

<Import Project="$(MSBuildExtensionsPath32)\Microsoft\VisualStudio\v$(VisualStudioVersion)\$(WMSJSProjectDirectory)\Microsoft.VisualStudio.$(WMSJSProject).props" />

without this change, VS will hang trying to load the project.

* Delete all <Content> elements referring to winjs
* Near the bottom, after the <ItemGroup> that contains all the <Content> elements, add:

<ItemGroup>

<SDKReference Include="Microsoft.WinJS, Version=0.6" />

</ItemGroup>

* Then replace:

<Import Project="$(MSBuildExtensionsPath32)\Microsoft\VisualStudio\v$(VisualStudioVersion)\WWAProject\Microsoft.VisualStudio.WWAProject.targets" />

with

<Import Project="$(MSBuildExtensionsPath32)\Microsoft\VisualStudio\v$(VisualStudioVersion)\$(WMSJSProjectDirectory)\Microsoft.VisualStudio.$(WMSJSProject).targets" />

* Also make sure the default language has a subfolder (like en-US) and not just a single file in the strings folder.

SLN file:

* You can just delete the .SLN file and recreate from projects. Otherwise,
* Patch up the project file name in the .sln file.
* Could also consider adding different targets (such as ARM) that are defaults in the DP4 solution.

## Updating WinJS References (all .js files)

Replaced these lines:

<link rel="stylesheet" href="/winjs/css/ui-light.css" />

<script src="/winjs/js/base.js"></script> and <script type="ms-deferred/javascript" src="/winjs/js/base.js"></script>

<script src="/winjs/js/ui.js"></script> and <script type="ms-deferred/javascript" src="/winjs/js/ui.js"></script>

with these:

<link href="//Microsoft.WinJS.0.6/css/ui-light.css" rel="stylesheet" type="text/css" />

<script type="text/javascript" src="//Microsoft.WinJS.0.6/js/base.js"></script>

<script type="text/javascript" src="//Microsoft.WinJS.0.6/js/ui.js"></script>

Note that ui-light can be ui-dark as well.

After this, you can delete all the winjs files in the project, and delete all other references to those files within your project. You only need to refer to base.js and ui.js.

Also, type="ms-deferred/javascript" is no longer necessary on your own .js files and can be removed.

## Activation Event

In default.js or wherever activation code exists, replace the onmainwindowactivated even with just onactivated.

## Fragment Loading

Existing fragment loading still works, just changing clone to renderCopy and drop the second parameter.

There is also a new page manager control that centralizes a bunch of the template code you’ve needed to copy wholesale or has been inserted from templates. To use this:

* Create a new app in VS using the Navigation template for reference
* In your project, add an existing item and point to **js/navigator.js** in the template project you just created, adding it to your js folder. Then add a **<script>** reference in your default.js file (any any other pages that use the control):

<script src="/js/navigator.js"></script>

* In navigator.js, make sure the line near the top:

WinJS.Namespace.define("[app]", {

contains your app name in *[app]*.

* Add the following to the content <div> in default.html:

data-win-control="[app].PageControlNavigator" data-win-options="{home: '/html/[page].html'}"></div>

where [app] is the app name and [page] is the page you want to go to by default. As you can see, PageControlNavigator is defined within navigator.js in the namespace you indicate in that file. This is usually set from the template when you create a new app, but you'll need to make sure these match when importing navigator.js from another project.

* Make sure you’re calling **WinJS.UI.processAll()** (or process on the specific content div) within the **app.onactivated** event handler, otherwise the page navigator control won’t instantiate.
* With the page navigator control in place, the following pieces of code are no longer necessary:
* Calls to WinJS.Navigation.navigate to go to your default fragment (this is what the home: property indicates in the control declaration in HTML.
* Calls to WinJS.Navigation.addEventListener(‘navigated’…) plus any navigated event handlers.
* Any handlers for fragmentAppended events.
* Calls to WinJS.UI.Fragments.clone (or renderCopy) and the entire body that’s usually contained within its then() method, including code that sets up back buttons and queue’s the fragmentAppended event. All of this is contained now within the page navigator control.

## App Bar

These changes are documented in the migration guide, but these are the common aspects:

* In the HTML, rename **position** to **placement**, remove options of autoHide, transient, and lightDismiss, and if you were using lightDismiss: false, add **sticky: ‘true’**.
* Also add **layout: ‘custom’** to the options. This makes it possible for you to keep most of your existing code the same. However, it’s advised that you replace the markup with AppBarCommand controls as shown in the migration doc.
* In code, do the following replacements on method named:
  + showElements to **showCommands**
  + hideElements to **hideCommands**
  + (There are others—see the migration doc)

## ListView

As documented earlier, the ListView (and data source APIs) have changed somewhat; we expect this will be a significant part of migrating a majority of apps, so review that section for changes.