Contents

Large applications can be broken into smaller modules, downloaded "on demand". In Silverlight applications, Visual Studio tooling allows the developer to create separate deployment packages which can be dynamically downloaded as needed. Non-Silverlight applications can also download additional assemblies as needed.

To use DevForce within a downloaded module you must register it for **on-demand discovery**.
Unknown macro: IBNoteThe "IBNote" macro is not in the list of registered macros. Verify the spelling or contact your administrator.

Overview

On-demand discovery allows DevForce to probe your new content for any entity models, known types, or custom implementations of DevForce types.

You can ask DevForce to download the content, or once downloaded you can tell DevForce about it. Adding dynamic content for on-demand discovery causes DevForce to recompose its "parts" catalog and metadata about your models and components. If you have dynamic content which you neglect to tell DevForce about, you'll find both that your customizations are not found and exceptions occur when using the entity model. (The most likely error is a message that the entity type queried is not a "known type".)

You can register new modules at any time, and start using your entity model and other DevForce features within it as soon as discovery completes. The *CompositionHost* will fire a *Recomposed* event once ready, which you can listen on to check for completion or if any errors occurred.

To perform on-demand discovery, use one of the *CompositionHost.Add* overloads. Trace messages are also generated, which you can capture using a custom logger or trace viewer.

In Silverlight

It's common practice to use Application Library Caching (ALC) for XAPs in a multi-module application. The DevForce assemblies all support ALC.

You can add dynamic content to a Silverlight application in several ways.

With Cocktail

If you're using Cocktail, it has built-in support for on-demand loading and discovery. Use the *Composition.AddXap* method to add dynamic content to your application. See here for more information and sample code.

With Prism modularity

If you're using Prism modularity you can tell DevForce about the new content, either once the module downloads or when it initializes. Here's a sample which adds the Uri of the downloaded module to the *CompositionHost*.

```
public void Download(string moduleName) {
this.ModuleManager.LoadModuleCompleted += this.ModuleManager_LoadModuleCompleted;
this.ModuleManager.LoadModule(moduleName);
}
private void ModuleManager_LoadModuleCompleted(object sender, LoadModuleCompletedEventArgs e) {
  var uri = new Uri(e.ModuleInfo.Ref, UriKind.Relative);
  IdeaBlade.Core.Composition.CompositionHost.Add(uri);
}
```

```
Public Sub Download(ByVal moduleName As String)
 AddHandler Me.ModuleManager.LoadModuleCompleted, _
   AddressOf ModuleManager_LoadModuleCompleted
 Me.ModuleManager.LoadModule(moduleName)
End Sub
```

```
Private Sub ModuleManager_LoadModuleCompleted(ByVal sender As Object, _
 ByVal e As LoadModuleCompletedEventArgs)
 Dim uri = New Uri(e.ModuleInfo.Ref, UriKind.Relative)
  IdeaBlade.Core.Composition.CompositionHost.Add(uri)
End Sub
```

See our sample DevForce/Prism Modularity Quickstart for more information.

With MEF deployment catalog

If you're using the MEF *DeploymentCatalog*, once the catalog downloads you can tell DevForce that new content is available by adding the *DeploymentCatalog* to the *CompositionHost*. For example:

```
public void AddXap(string uri) {
  DeploymentCatalog catalog;
  catalog = new DeploymentCatalog(uri);
  catalog.DownloadAsync();
  catalog.DownloadCompleted += new EventHandler<System.ComponentModel.AsyncCompletedEventArgs>(catalog_DownloadCompleted)
}
private void catalog_DownloadCompleted(object sender, System.ComponentModel.AsyncCompletedEventArgs e) {
  IdeaBlade.Core.Composition.CompositionHost.Add(sender as DeploymentCatalog);
}
```

```
Public Sub AddXap(ByVal uri As String)
 Dim catalog As DeploymentCatalog
  catalog = New DeploymentCatalog(uri)
  catalog.DownloadAsync()
 AddHandler catalog.DownloadCompleted, AddressOf catalog_DownloadCompleted
End Sub
Private Sub catalog_DownloadCompleted(ByVal sender As Object, _
 ByVal e As System.ComponentModel.AsyncCompletedEventArgs)
  IdeaBlade.Core.Composition.CompositionHost.Add(TryCast(sender,  _
    DeploymentCatalog))
End Sub
```

If using the *DeploymentCatalogService* shown in MEF samples, you likely already have code for the *catalog_DownloadCompleted* event handler. Add the line to call *CompositionHost.Add* for the catalog.

With a DynamicXap

If you're not using Cocktail, MEF or Prism there's still another means of notifying DevForce about dynamic content with the *DynamicXap* class. The class can be constructed in a few different ways, but the easiest may be with a Uri, to have DevForce automatically download the content.

```
public void DoDownload(string xapName) {
  var xap = new IdeaBlade.Core.DynamicXap(new Uri(xapName, UriKind.Relative));
  IdeaBlade.Core.Composition.CompositionHost.Add(xap);
}
```

```
Public Sub DoDownload(ByVal xapName As String)
 Dim xap = New IdeaBlade.Core.DynamicXap(New Uri(xapName, UriKind.Relative))
  IdeaBlade.Core.Composition.CompositionHost.Add(xap)
End Sub
```

Listening for DynamicXap load

You can load a *DynamicXap* without adding it to the *CompositionHost*. Listen on its *Loaded* event to tell you when the download has completed or if an error occurred.

```
var xap = new DynamicXap(new Uri("SecondModule.xap", UriKind.Relative));
xap.Loaded += (o, args) => {
 if (args.HasError) {
    TraceFns.WriteLine(args.Error.Message);
  } else {
    var dzap = o as DynamicXap;
    TraceFns.WriteLine(dzap.Name + " downloaded");
  }
};
```

```
Dim xap = New DynamicXap(New Uri("SecondModule.xap", UriKind.Relative))
AddHandler xap.Loaded, Sub(o, args)
If args.HasError Then
 TraceFns.WriteLine(args.Error.Message)
```

```
 Else
Dim dzap = TryCast(o, DynamicXap)
 TraceFns.WriteLine(dzap.Name & " downloaded")
 End If
End Sub
```

In .NET

Dynamic content can be added to standard .NET applications too.

Use *CompositionHost.Add(catalog)* to add either an AssemblyCatalog or AggregateCatalog to the MEF composition container managed by DevForce.

Listening for recomposition events

Regardless of how you've added content to the *CompositionHost*, you can listen for its *Recomposed* event to notify you when new content has been downloaded.

```
CompositionHost.Recomposed += CompositionHost_Recomposed;
```

```
AddHandler CompositionHost.Recomposed, AddressOf CompositionHost_Recomposed
```

The *RecomposedEventArgs* will indicate the *Xap* or assembly triggering the recomposition, or the error if an exception was thrown. You can use the *Name* property on the *Xap* to identify the on-demand module.