Contents

Discovery, sometimes called **probing**, is the process DevForce uses to find your entity models and custom components.

Startup

DevForce uses the Managed Extensibility Framework (MEF) from Microsoft to provide much of its runtime extensibility. When your application starts, DevForce builds a "parts" catalog on both client and server containing the customizations it has discovered. By default, DevForce will check all non-system / non-IdeaBlade assemblies for "exports", i.e.,implementations of the DevForce interfaces and base types, as well as other components or your application.

- In Silverlight applications, discovery is initially done using the assembly parts in the main XAP. For modular applications with on-demand download of XAPs DevForce also supports on-demand discovery.
- In Windows Store applications, discovery uses the assemblies in the app package.
- In Windows Phone applications, discovery uses the assemblies in the XAP.
- In Desktop, ASP.NET applications and the EntityServer, discovery uses the assemblies in the main executable/bin folder.

The CompositionHost manages the discovery process, and provides properties and methods you can use to modify the default behavior, access the underlying MEF container, and add dynamic content.

Controlling discovery

In a large application with many assemblies you may find that the discovery performed at startup takes too long, or maybe you decide there's just no reason to look in assemblies which will never contain exports or entity models. Trace messages about the assemblies added the the "parts" catalog, all probe assemblies found, and total discovery time are written to the debug log and can help diagnose a problem.

The easiest and most common way to alter the default discovery behavior is to change the search and/or ignore pattern(s) DevForce will use.

For ASP.NET and Silverlight applications the default SearchPattern is "*.dll"; for other applications the default is "*.dll" and "*.exe".

For all application types, the default IgnorePatterns are set to exclude IdeaBlade and system assemblies, as well as assemblies from many third party control suites.

Both *SearchPatterns* and *IgnorePatterns* accept either simple wildcards or Regex patterns. Simple wildcard searches are converted into Regex patterns by DevForce when probing occurs. The search is case insensitive.

When should you use *SearchPatterns* versus *IgnorePatterns*? *SearchPatterns* are useful when you know specifically what you're looking for; while *IgnorePatterns* are helpful when you know what you don't want.

Here's a sample using the *SearchPatterns* to limit probing to only a single assembly. Generally in larger applications you'll have multiple assemblies to be probed, so you should be sure to include one or more patterns to cover them. This first clears the default *SearchPatterns*, and then sets a pattern for a specific assembly named "MyDomainModel.dll".

```
IdeaBlade.Core.Composition.CompositionHost.SearchPatterns.Clear();
IdeaBlade.Core.Composition.CompositionHost.SearchPatterns.Add("MyDomainModel.dll");
```

```
IdeaBlade.Core.Composition.CompositionHost.SearchPatterns.Clear()
IdeaBlade.Core.Composition.CompositionHost.SearchPatterns.Add("MyDomainModel.dll")
```

Now let's try modifying the *IgnorePatterns*. DevForce will by default ignore IdeaBlade and system assemblies, as well as a number of third party assemblies. You shouldn't clear this list, and you certainly shouldn't remove the "IdeaBlade.*" assemblies, since both actions will cause your application to fail, but you can add additional third party tools and suites to the list. For example, here we're adding an exclusion for all assemblies (and other files) matching "GalaSoft.*".

```
IdeaBlade.Core.Composition.CompositionHost.IgnorePatterns.Add("GalaSoft.*");
```

```
IdeaBlade.Core.Composition.CompositionHost.IgnorePatterns.Add("GalaSoft.*")
```

You need to set the *SearchPatterns* and *IgnorePatterns* as early as possible in your application, since as soon as you use the *IdeaBladeConfig.Instance*, *TraceFns* and *DebugFns* methods, or an *EntityManager*, initialization will automatically occur.

- In an EntityServer hosted by IIS or an ASP.NET application, the best place to set the patterns is in the application startup logic in the global.asax.
- In Silverlight, you should do this in the Application_Startup logic to ensure that your changes are made before probing occurs.
- In a desktop application, set the patterns before you begin using an *EntityManager* or other DevForce components.

If you've set *SearchPatterns* and/or *IgnorePatterns*, it's usually a good idea to check the debug log for the tracing messages written about what DevForce has discovered. It's easy to mistype a pattern and not have your assemblies probed, or alternately to have too many assemblies probed resulting in poor startup performance.

The first time DevForce finds and uses your custom classes it will log this fact to the debug log (or other *ITraceLogger* implementation). This information, along with logging about all assemblies loaded into the "parts" catalog, can help in diagnosing problems should they occur.

What should be discovered

There are several components of your application which DevForce must find in order for your application to work as wanted.

- All custom implementations of DevForce interfaces and base types. These are the exports placed in the **parts catalog**.
- Entity models (aka domain models) - DevForce must be able to find any assembly holding a model. It uses this assembly to load model metadata and gather other information about the types in your model.
- Remote server methods - If your application is using remote server methods then DevForce must be able to find them.
- POCO providers and types - For DevForce to find your POCO types and providers, they must be defined in an assembly probed.
- Known types - For any of the types which can be transmitted across tiers - for example those types passed to or from a remote server method - DevForce must know these types before initiating communications.

In all the above cases, DevForce will look through all the assemblies which have been probed to find what it needs. If you've chosen to modify either the *SearchPatterns* or *IgnorePatterns*, make sure that the patterns will allow all assemblies needed to be discovered. You can check the read only *CompositionHost.Instance.ProbeAssemblies* at runtime to see which assemblies were found; they're also listed in the debug log.