

# MEF In .NET 4.0

## Overview, Concepts, Export & Imports, Metadata, Catalogs, Export Providers, Contract Adapters, Primitives, Project

The Managed Extensibility Framework – MEF – is a modern composition framework for the creation of extensible applications composed from parts supplied at runtime.

MEF is an open-source project with a .NET 3.5 build supplied on Codeplex and it is an integral part of .NET 4.0. It is suitable for use with large-scale applications (the new editor in VS2010 is built with MEF and WPF).

Application architects are being asked to provide designs exhibiting high levels of extensibility and flexibility and MEF is a key building block.

MEF allows you to create parts (i.e. components) using managed code that identifies imports (what the part needs from other parts) and exports (what the part offers to others). Parts come from catalogs and the composition process is managed by various export providers and composition services.

This course is your first step in getting up to speed with the new .NET way of building components. You will be provided with detailed coverage of new and updated extensibility concepts that MEF works with, you will see lots of of sample code, you will write plenty yourself and you will get practical guidance on how to use MEF when designing your own apps.

### Target Audience:

Senior software engineers wishing to author composable parts using MEF.

### Prerequisites:

Attendees are expected to be experienced C#/.NET developers having familiarity with modern object-oriented design concepts.

All code examples and labs are in C# & .NET4.0/VS2010.

<p><b>Overview</b></p> <p>How best to tackle extensibility? Relationship to IoC What composition delivers Developer setup</p> <p><b>Concepts</b></p> <p>Composition Parts Imports &amp; exports Contracts and metadata Adapters Export providers</p> <p><b>Imports &amp; Exports</b></p> <p>Attributes Single and multiple exports Describing exports ImportMany Eager and lazy export loading</p> <p><b>Metadata</b></p> <p>Dictionary of name/value pairs Strongly typed metadata CreationPolicy</p> <p><b>Catalogs</b></p> <p>Role of catalogs is to supply parts/definitions</p> <p>TypeCatalog AssemblyCatalog DirectoryCatalog AggregateCatalog</p> <p><b>Composition</b></p> <p>CompositionContainer CompositionEngine PartNotDiscoverable attribute Initial parts</p>	<p><b>Export Providers</b></p> <p>Role: to provide exports CompositionContainer manages a topology of export providers Flexibility in supplying parts Defaults/features in product line-multiSKU</p> <p><b>Contract Adapters</b></p> <p>AdaptingExportProvider Why &amp; how to adapt between contracts ContractName and Metadata constants Writing an adapter</p> <p><b>Primitives</b></p> <p>Definitions – Part, Import/Export ContractBasedImportDefinition ComposablePart ComposablePartCatalog</p> <p><b>Error Handling</b></p> <p>Error Handling ICompositionElement Custom exceptions Composition errors Debugger proxies</p> <p><b>Debugging Session</b></p> <p>Debugging app through the MEF source to see where errors are raised and handled</p> <p><b>Designing MEF Applications</b></p> <p>Think about the MEF cascade Batch of top-level parts App-internal parts from AssemblyCatalog Extensions from DirectoryCatalog</p> <p><b>Project</b></p> <p>We examine a project that demonstrates the important concepts covered in this course.</p>
---	--

