|  |
| --- |
| *Copyright © 2008 Microsoft Corporation. All rights reserved.* |
| MPF\_Proj |
| Documentation |
|  |
| **Last updated:** |
| **5/6/2008** |

|  |
| --- |
|  |

MPF P documentation

Contents

[1. Overview 4](#_Toc197875912)

[2. Architectural Overview 5](#_Toc197875913)

[2.1 Hierarchy nodes 5](#_Toc197875914)

[2.1.1 HierarchyNode 6](#_Toc197875915)

[2.1.2 ProjectNode 6](#_Toc197875916)

[2.1.3 FileNode 6](#_Toc197875917)

[2.1.4 DependentFileNode 7](#_Toc197875918)

[2.1.5 FolderNode 7](#_Toc197875919)

[2.1.6 ReferenceContainerNode 7](#_Toc197875920)

[2.2 ProjectFactory 7](#_Toc197875921)

[2.3 ProjectPackage 7](#_Toc197875922)

[2.4 References 8](#_Toc197875923)

[2.5 Documents 8](#_Toc197875924)

[2.6 Properties 9](#_Toc197875925)

[2.7 Registration attributes 10](#_Toc197875926)

[3. Extensibility points 11](#_Toc197875927)

[4. Supported Features 12](#_Toc197875928)

[4.1 General project features 12](#_Toc197875929)

[4.1.1 MSbuild support 12](#_Toc197875930)

[4.1.2 Automation support 12](#_Toc197875931)

[4.1.3 Source Control support 12](#_Toc197875932)

[4.1.4 ProjectDesigner pages support 12](#_Toc197875933)

[4.2 Project as project container 12](#_Toc197875934)

[4.3 Security 12](#_Toc197875935)

[5. Features on the wish-list 14](#_Toc197875936)

[5.1 Show all files 14](#_Toc197875937)

[5.2 Linked files 14](#_Toc197875938)

[5.3 Dynamically adding nested projects to a container project 14](#_Toc197875939)

[6. Testing 15](#_Toc197875940)

[6.1 Unit testing 15](#_Toc197875941)

[6.2 Integration testing 15](#_Toc197875942)

[6.3 FxCop 15](#_Toc197875943)

[7. Q and A 16](#_Toc197875944)

[8. Issues 17](#_Toc197875945)

[9. References 20](#_Toc197875946)

# Overview

Creating a new project type in Visual Studio is complex task. Using MPF\_Proj is a good starting point for creating custom project types in Visual Studio written in managed code but there are limitations that would have to be considered before using the framework.

MPF\_Proj is not a .NET library. It is rather a framework of source files (classes, utilities etc.) that can be included in a VSPackage project.

This document provides a basic understanding of the features supported and the extensibility points of the framework. This should help either to make the right decision whether to use the framework or not and also to get started creating a custom project type.

This document does not describe the fundamental architecture of a custom project type in Visual Studio. Please refer to the MSDN online documentation [here](http://msdn2.microsoft.com/en-us/library/bb165067.aspx). More reference are listed in the references chapter.

This version of the MPF\_Proj framework is targeting Visual Studio 2008.

Creating custom project types for Dev10 using MPF\_Proj is currently not supported.

# Architectural Overview

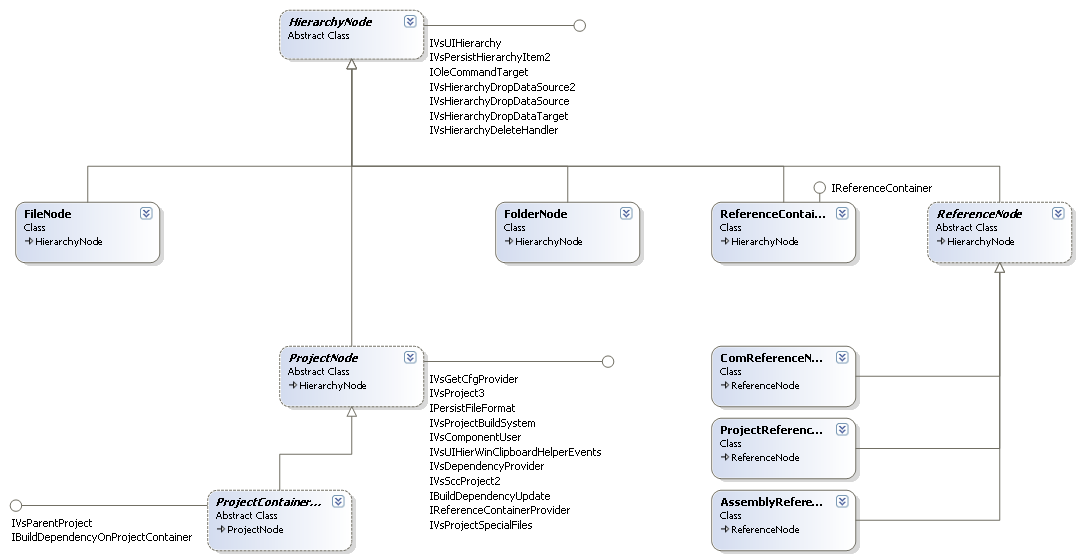
MPF\_Proj is a framework that interacts with the traditional Visual Studio COM interfaces (through an interop layer) as show in the following diagram. Writing a new VSPackage containing a new project type will get the entire base feature set for free. Additional project features not supported by the framework requires most often interfaces from the interop layer to be implemented on the new VSPackage.

The majority of the feature categories in MPF\_Proj are:

* Hierarchy nodes visible in the Solution Explorer
* References
* Documents
* Properties
* Automation (DTE)
* Project configuration and output groups

## Hierarchy nodes

The following figure shows a static class diagram of the hierarchy node classes in MPF\_Proj. HierarchyNode is the abstract class that implements common functionality for all nodes. ProjectNode is the root node of the hierarchy representing the project. The following sections briefly describes the node types.



*Hierarchy node classes*

### HierarchyNode

HierarchyNode is an abstract class that defines a node in the project. All nodes in the project ultimately derive from HierarchyNode. It deals with the user action via the GUI in the form of a hierarchy and it’s a parent hierarchy of zero or more child nodes, each of which itself can be a hierarchy. There are quite a few shell interfaces implemented but the most important ones are

* [IVsHierarchy](http://msdn2.microsoft.com/en-us/library/microsoft.visualstudio.shell.interop.ivshierarchy.aspx)
* [IVsUIHierarchy](http://msdn2.microsoft.com/en-us/library/microsoft.visualstudio.shell.interop.ivsuihierarchy_members.aspx)

These interfaces are generic interfaces to a hierarchy of nodes. Please follow the links to the online msdn documentation for more information on each of the interface methods.

All interface methods are implemented as virtual methods so that functionality can be overridden in derived classes.

### ProjectNode

The ProjectNode has the association to the top level node in a project as it is visualized in the solution explorer. ProjectNode has multiple responsibilities and the reason for adding all those responsibilities in one class is because of the Visual Studio shell. The shell will query for specific interfaces on the hierarchy (the ProjectNode) in order to see if a certain feature is supported. As of today there more that 5000 lines of code in ProjectNode and it implements 12 shell interfaces. First of all the ProjectNode is the root node in the hierarchy that represents the project. The project is responsible for handling all the child nodes. That also means that the project node knows how to create all the child nodes. If you define derived node types you should create factory methods in your derived ProjectNode class.

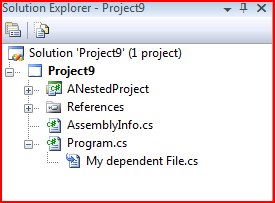
### FileNode

FileNode is the generic node class for files in the project. Typically you would have specific features for each of the file types that you support in a project type and therefore it often makes sense to create a derived class from FileNode. Unfortunately you can only use FileNode for Directory-based files. The reference based files (linked files) are not supported by the framework).

By default the FileNode also supports the SingleFileGenerator properties. That means that the properties shown in the properties window when the FileNode is selected are mapped to a SingleFileGeneratorNodeProperties object. This feature can be disabled in a derived class returning null in the CreateSingleFileGenerator method.

### DependentFileNode

The DependentFileNode is responsible for showing dependent files to FileNodes e.g. generated files or other dependent files as shown in the figure below



The MSBuild project file schema supports dependent files as a sub tag specifying the file being dependent upon.

<ItemGroup>

<Content Include="My dependent File.cs">

<SubType>Content</SubType>

<DependentUpon>Program.cs</DependentUpon>

</Content>

</ItemGroup>

In order to support dependent files the Following property (this.CanFileNodesHaveChilds = true;) must be set on the derived ProjectNode class e.g. in the constructor.

### FolderNode

This class represents is a physical folder on disk. That’s because the project system is directory based only.

### ReferenceContainerNode

This is the virtual node containing all the reference nodes. It is created in the ProcessReferences method on the ProjectNode. In order not to show the references node just override this methods in the derived project.

## ProjectFactory

This is an abstract class that creates the project. It also contains functionality in order for projects to be flavorable. The IronPython sample in the VSSDK demonstrates the flavorable project functionality (the IronPython Web Application Project).

## ProjectPackage

ProjectPackage derives from the standard Package class in MPF Shell library. It adds project related functionality like setting up solution listeners (see the Initialize method) and handles persistence of user option that is saved in the suo-file. Currently one parameter is being saved to the suo-file which is the ProjectTrustLevel (either an unknown project or a trusted project).

## References

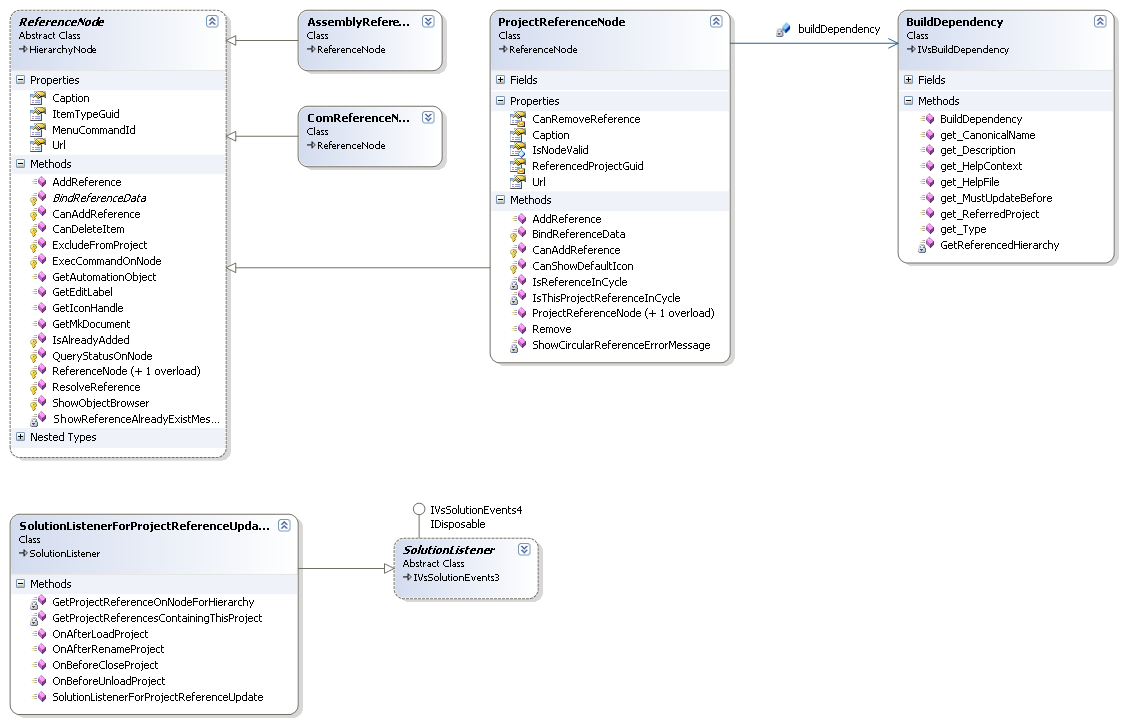
A managed language project has the notion of references to other assemblies. That means when you build the project the compiler knows how to resolve external types defined in other assemblies. The concept of references in MPF\_Proj maps directly to this kind of references. There are three categories of references supported. That is

* Assembly reference
* COM reference
* Project to Project reference

All reference resolving at build time is delegated to a common MSBuild task used by other managed languages (C#, Visual Basic etc.).

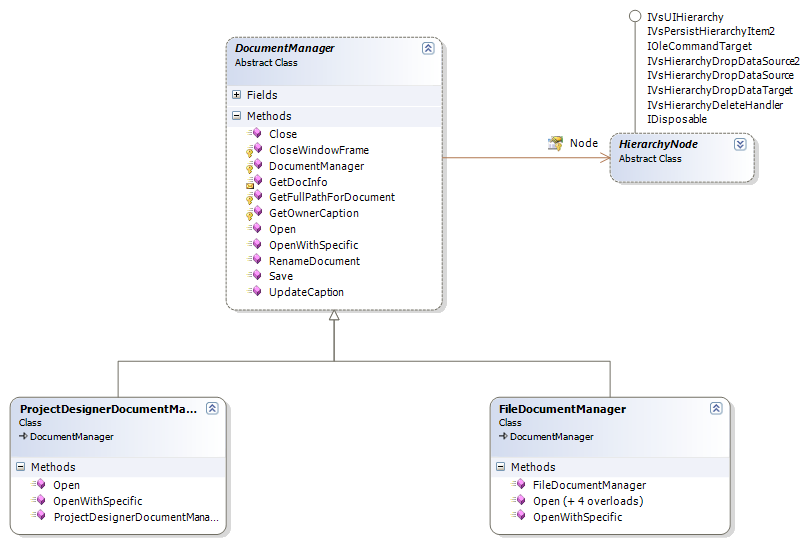
Project to project references creates build dependencies that will show up in the build order dialog. The SolutionListenerForProjectReferenceUpdates handles the proper update of project references during a series of solution events, e.g. when the target project is deleted the reference automatically gets deleted.

The following figure shows a static class diagram for the classes supporting reference handling.



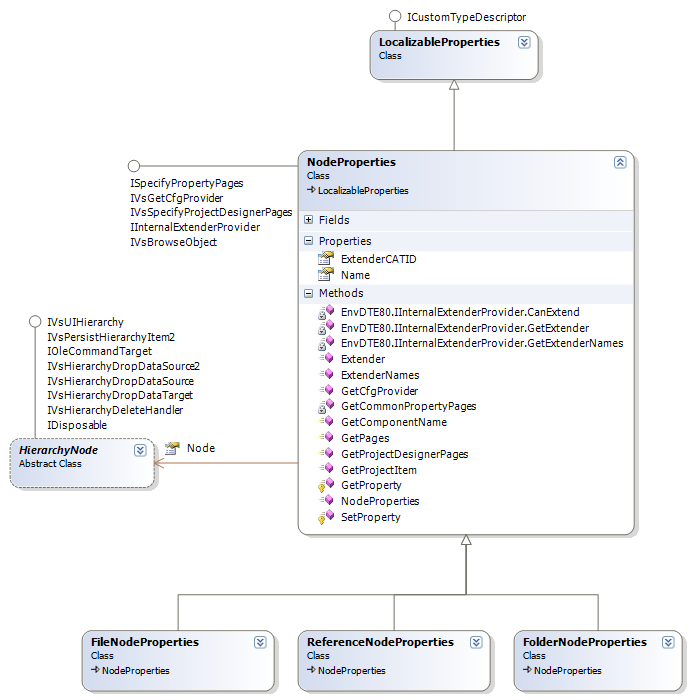
## Documents

When the solution file, a project file or a file in a project is opened in an editor it is registered in the shell in a running document table (RDT). To manage the opening project documents MPF\_Proj contains an abstract class called DocumentManager. Two concrete implementations exists which is the FileDocumentManager and the ProjectDesignerDocumentManager.



## Properties

Properties in a project can be associated with either the project itself (the root node) or a project item. The properties can be displayed either as a set of property pages in a project designer or in the Property Window. The project designer is more frequently referred to as the Property Pages Window that shows up when the user clicks on the Properties command in the Project context menu. This feature is handled through the implementation of a specific VSX interface called [ISpecifyPropertyPages](http://msdn2.microsoft.com/en-us/library/microsoft.visualstudio.ole.interop.ispecifypropertypages.aspx)



## Registration attributes

Please refer to the [online msdn documentation](http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.shell.provideprojectfactoryattribute_members.aspx) for registration attributes used for registering a custom project type

The following registration attributes are from the NestedProject sample:

[ProvideProjectFactory(typeof(NestedProjectFactory), "MyNestedProject", "MyNestedProject Files (\*.nestedproj);\*.nestedproj", "nestedproj", "nestedproj", @"..\..\Templates\Projects")]

[ProvideProjectItem(typeof(NestedProjectFactory), "Nested Project Items", @"..\..\Templates\ProjectItems", 500)]

# Extensibility points

The design of MPF\_Proj is based on an object oriented perspective where extensibility is provided by overriding virtual methods on the base classes.

Finding the right place to start in order override a feature from the base class can be quite challenging because there are too many virtual methods in the base classes. This framework was created before the concept of partial class was introduced. Therefore only little code has been refactored to use partial classes. The ProjectNode class is a partial class and all feature code related to Copy and Paste and Drag and Drop scenarios are found in the ProjectNode partial class located in one file called ProjectNode.CopyPaste.cs.

Currently the framework contains 608 virtual properties and methods. Most of the virtual methods are introduced due interface inheritance on virtually all classes that’s callable from the shell, e.g. the HierarchyNode implements IVsHierarchy (and a lot of other interfaces). All methods from interface implementations have been made virtual so that customization/extensions are possible from derived classes. Many of the shell interface methods contain ugly parameter types like IntPtr that is never used in pure managed applications.

# Supported Features

## General project features

### MSbuild support

The project file format is MSBuild based and the MSBuild object model is used as the build engine. That makes it easy for file based project systems to handle build and persistence but for those custom project types that are not file based or have a special build engine some customization of the MPF\_Proj code is needed.

### Automation support

Every custom project type should support a minimum set of automation interfaces as defined in the DTE object model. Those interfaces are

* Project
* ProjectItems
* ProjectItem
* Properties
* Property

MPF\_Proj has good support for automation although not every method is implemented. Also language specific automation interfaces are supported. Please see section on [language specific features](#_Language_specific_features) for the list of supported interfaces.

### Source Control support

Most project systems dealing with source files or other assets require source control support. This feature is supported by default in the framework.

### ProjectDesigner pages support

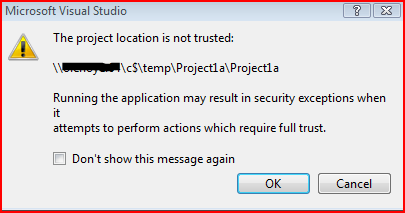
Supported by default but it can be turned off in order to support the old style Property Pages dialog if needed.

## Project as project container

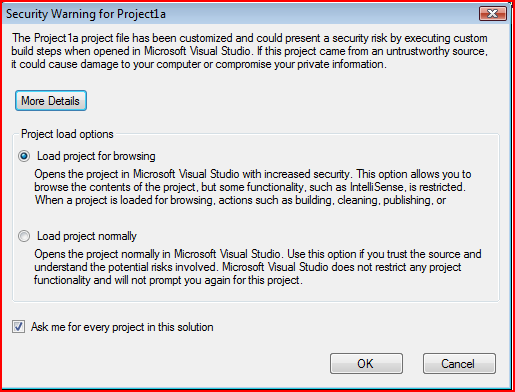
MPF\_Proj supports nesting of Projects inside a container project. A sample called NestedProject can be found in the samples folder.

## Security

There are a number of security features implemented in the framework. Trying to load a project from a network location will automatically cause the following dialog to show up



Another security feature deals with the import of msbuild project files not listed in the registry as safe to import. The following dialog will show up and would require a user action to load the project normally or open for browsing only.



To disable the security check you would have to override the following method on the project node

protected override ProjectLoadOption CheckProjectForSecurity(

ProjectSecurityChecker projectSecurityChecker,

ProjectSecurityChecker userProjectSecurityChecker)

{

return ProjectLoadOption.LoadNormally;

}

# Features on the wish-list

MPF\_Proj has support for the majority of features required by a project system but there are still features to be added to MPF\_Proj. The following list gives brief description of the missing features.

## Show all files

For a directory-based project system the files in the project reflects either the full set or a subset of files located in the project cone. Therefore it can become very handy to easily add and remove files from the project simply by showing the full set of files within the project. This feature is e.g. supported by the C# and VB project system. A special icon is show for those files not in included in the project.

## Linked files

In a referenced based project system the file items are links only.

## Dynamically adding nested projects to a container project

After loading a container project there are two ways a user can add a new/existing project to a container as a nested project. It can either be done through the UI or through the automation model.

# Testing

As part of the source both unit tests and integration tests are provided. That helps the developer to refactor and add new functionality without breaking existing features. Writing real unit tests for VsPackage functionality requires a lot of work because of the high number of shell services being and therefore has to be mocked. Writing integrations tests are easier and actually verifies that the code works in a real scenario.

All test projects are mstest based and all tests can be run from within Visual Studio.

## Unit testing

A few number of real unit tests exists. Please refer to the UnitTest project for a list of all the tests.

## Integration testing

A number of integration tests exists in order to validate the framework actually works in a real instance of Visual Studio. The NestedProject sample is used as integration test sample.

## FxCop

The framework is not FxCop clean. A GlobalSuppressions.cs file contains all the suppression added on a project level.

# Q and A

1. Q: Can I use MPF\_Proj to create a custom project type for and isolated shell solution?   
   A: Yes,
2. Q: Has the MPF\_Proj code been optimized for performance, e.g. being able to delay load items in sub folders until they are expanded.  
   A: No

# References

VSX Developer center on MSDN:

<http://msdn.microsoft.com/en-us/vsx/default.aspx>