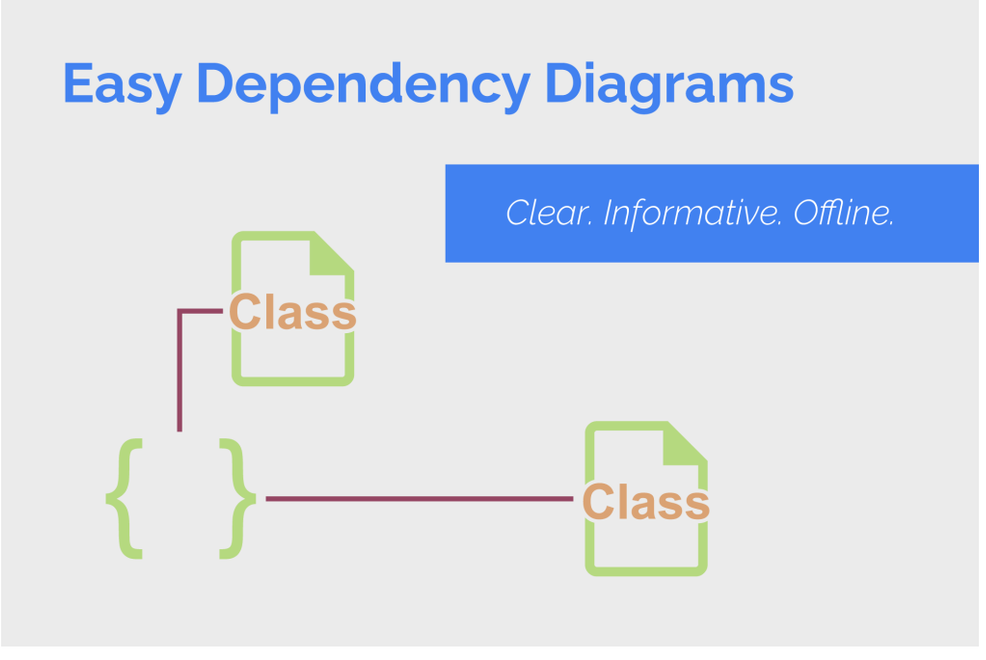
# Easy Dependency Diagrams



# Table of Contents

# Introduction

The Easy Dependency Diagrams asset can be used to create various diagrams of your C# project and examine its components' structure and internal dependencies. The tool is completely offline so no runtime information is required.

To open the Easy Dependency Diagrams window, install the asset, navigate to the 'Tools' menu in the top navigation bar, and choose the "Easy Dependency Diagrams > Show Window" option (Figure 1).

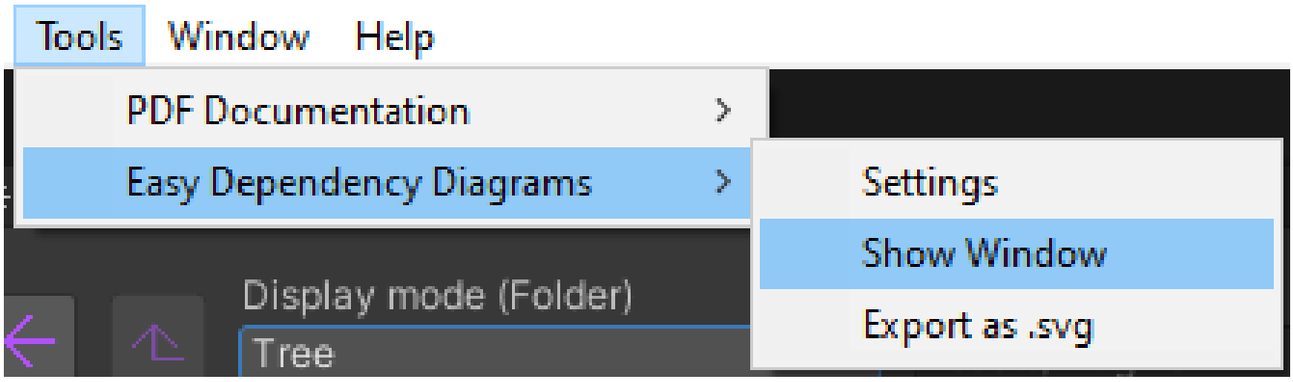



Figure 1: Unity top navigation menu

At the top of the window, you have the navigation options and settings (Figure 2). The 'Display mode' dropdown menu allows you to choose between different display modes, detailed later in the next section. The project root can also be changed. It indicates the scope of the search for dependencies.


Figure 2: Easy Dependency Diagrams navigation bar

There are 6 available display modes:

**1. Tree**

This is the default view for navigating through the files. Figure 3 shows the Tree display mode when viewing folders. In this case, the view works just like other file browsers, although only showing .cs files. To change the target,

simply click on the icon of that element.

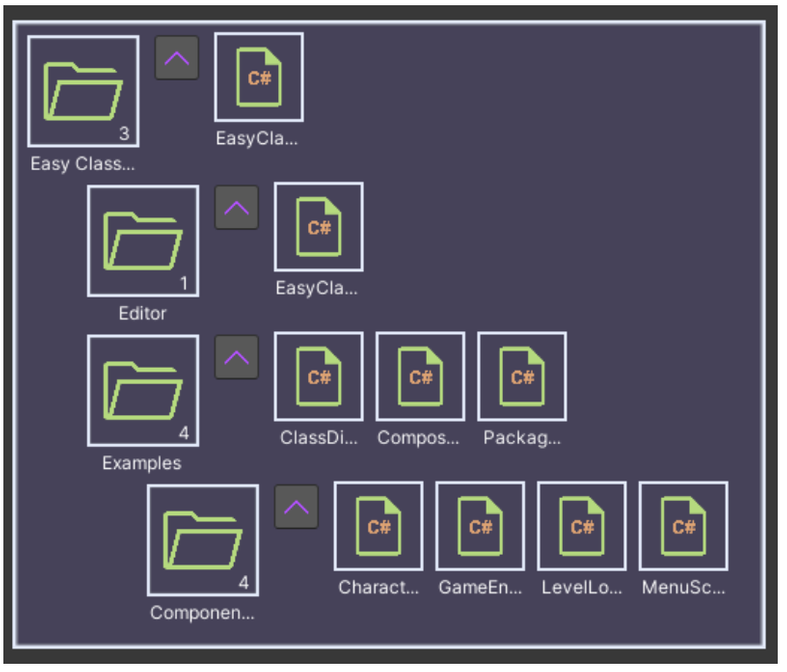



Figure 3: Tree display mode when viewing folders

When viewing files or inner structures in Tree mode, you will be shown all the children of the target down to the lowest level of variables and methods (Figure 4). All code elements have buttons next to them. You can view the related element in a code editor or collapse/expand all its children in the tree. When the selected target is a file, namespace or class, you can change the display mode to one of the other possible options.

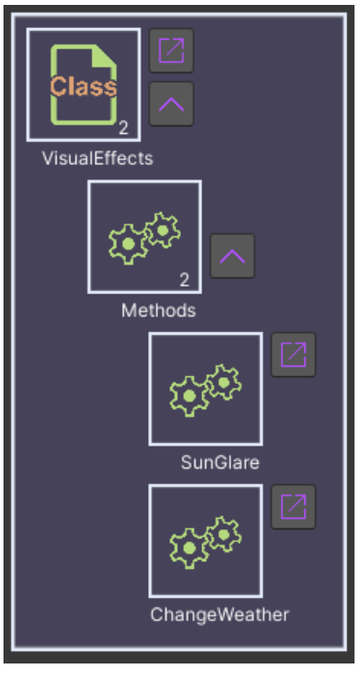



Figure 4: Tree display mode when viewing any other structure than folders

## 2. Component diagram

"Component diagram" is a structural UML diagram that is generally used to display the physical components and their relationships in a system, but can also be used to model the dependencies between the file components that make a software. Component diagram is a display mode option for files, showing all dependencies of the target file and files dependent on the target itself. An example of a component diagram can be seen in Figure 5.

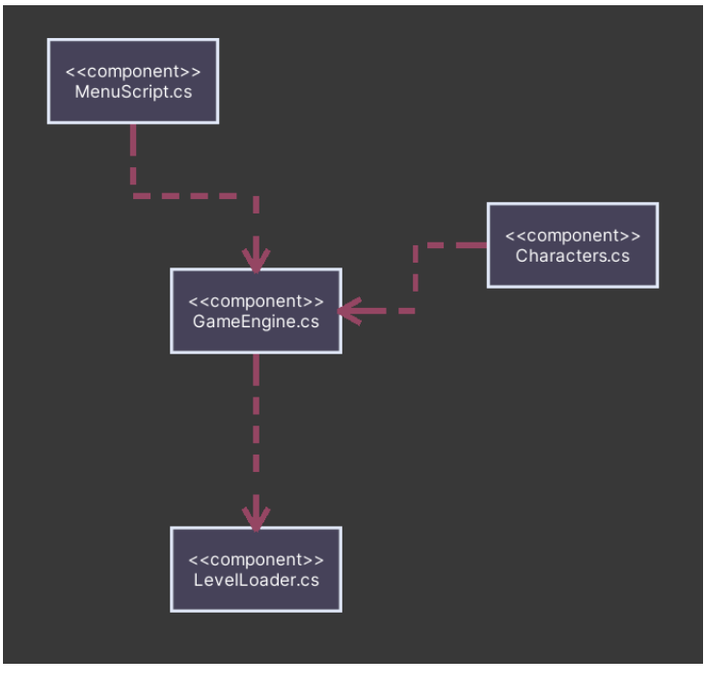



Figure 5: Example of a component diagram

## 3. Package diagram

"Package diagram" is a structural UML diagram that shows the structure of your software in terms of packages. In C#, namespaces are 'packages', so the package diagram display mode option can be used to show the relationships between namespaces. The diagram shows dependencies and parent-child relationships. An example can be seen in Figure 6.

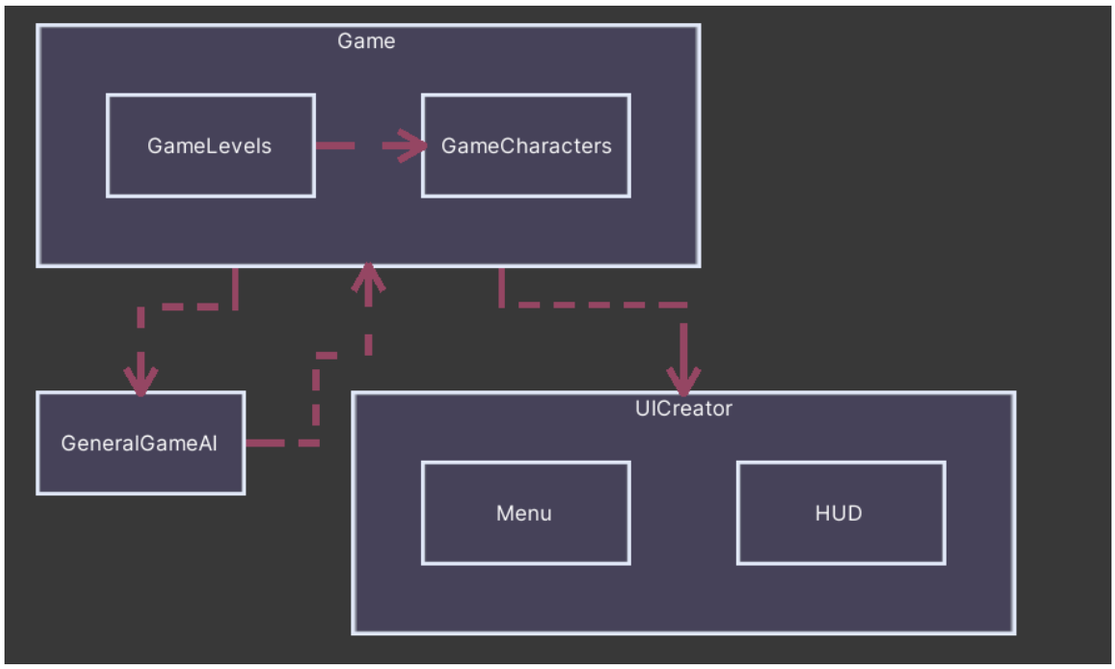



Figure 6: Example of a package diagram

## 4. Class diagram

"Class diagram" is the most commonly known structural UML diagram. It is a display mode option for classes that shows the relationships between classes and the attributes and methods of classes. The diagram follows basic UML notation by noting the attributes' visibility and categorizing different relationships to dependencies, inheritances, associations, aggregations and compositions. Figure 7 shows an example of a class diagram.

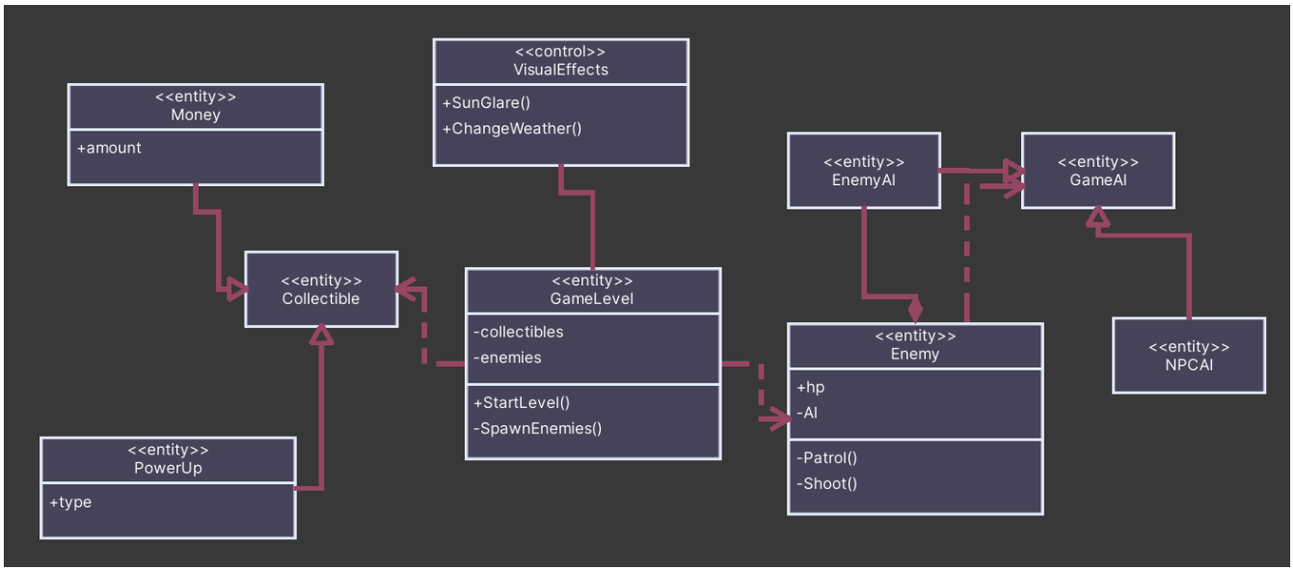



Figure 7: Example of a class diagram

## 5. Composite structure diagram

"Composite structure diagram" is a structural UML diagram that focuses on the contents of a class. You can seean example in Figure 8. The composite structure shows the parts that a class consists of. A part could, for example, be a variable or a subclass. Some information in a composite structure diagram might be only clear after examining the high-level workings of the software or runtime information. The parser does not have this information, but constructs the diagram based on available information.

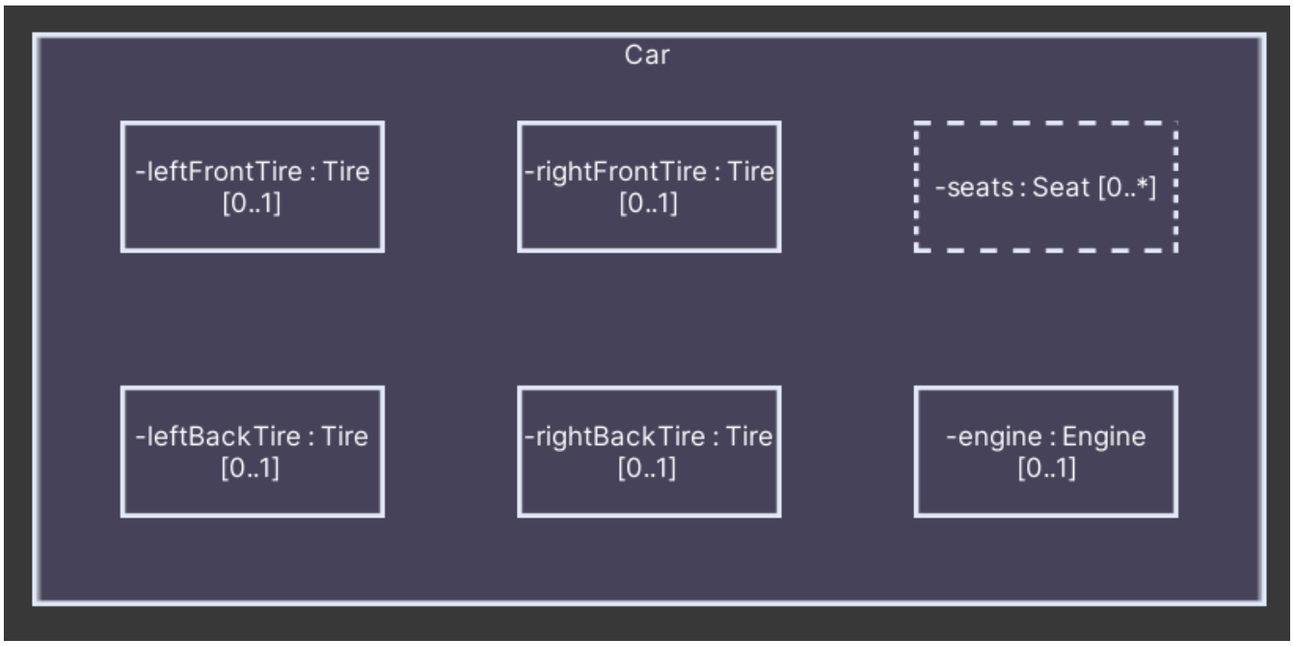



Figure 8: Example of a composite structure diagram

## 6. Dependencies

"Dependency" display mode allows you to examine the dependencies of a specific class or namespace in a simple but informative form. Figure 9 provides an example. This display mode is a great way to quickly see if a class is dependent on another.

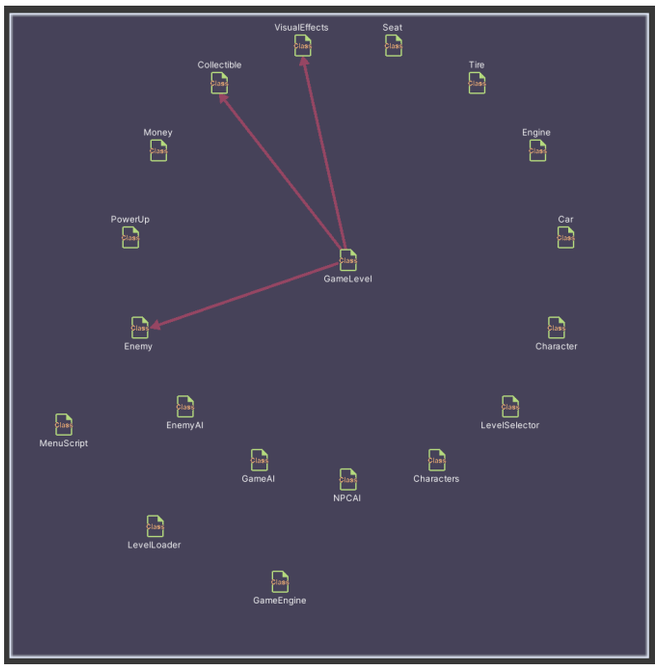



Figure 9: Example of the dependency display mode

## 7. Folder Dependencies

"Folder Dependencies" display mode allows you to examine the dependencies of folders similarly as the dependency view works for classes. A folder is dependant on another folder if its contents are dependant on the contents of the target folder. Figure 10 shows an example.

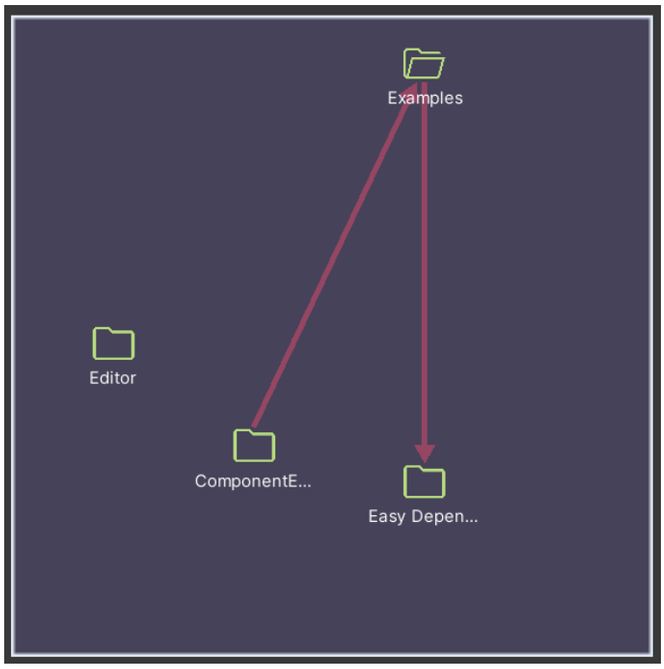



Figure 10: Folder Dependencies

In the top-right corner of the window, there is a floating info box, that displays the selected target and relevant data and options for the current display mode (Figure 11). The infobox can be moved or minimized, and has a handy button to open the currently viewed target in a code editor.

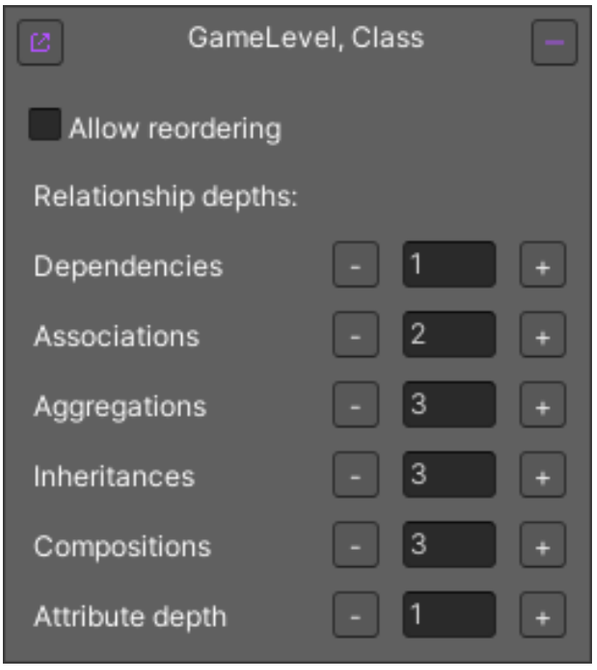



Figure 11: Infobox when viewing a class diagram

## Settings

Some general settings can be accessed through the 'Settings' option from the Unity top navigation bar (see Figure 1). This window is shown in Figure 12. Here you can switch the filter type for the parser. For larger projects, it is recommended to have some form of filter, as otherwise all files will be parser for every single diagram and can slow the system down a lot. You can also write down paths that will be included or excluded no matter which filter is used. For example, you could exclude "Assets/Plugins" or always include "GameEngine.cs". You can also change any interface colors to match your style!

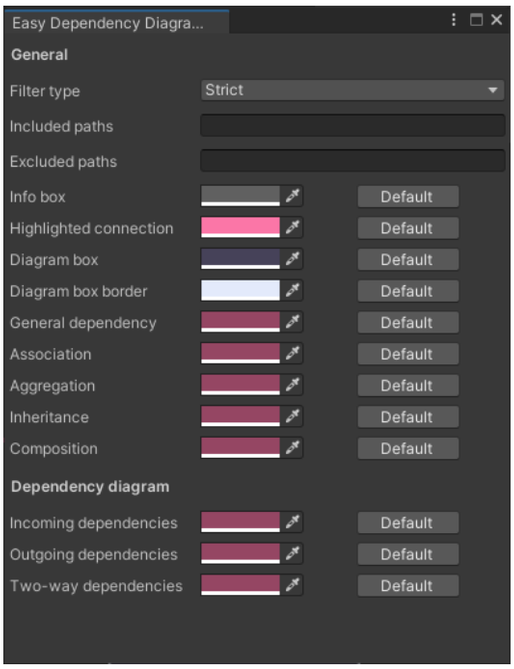



Figure 12: Easy Dependency Diagrams Settings interface

## SVG export

The tool has the handy functionality of being able to export any diagram in .svg format, allowing you to easily share the diagram or make further modifications. Simply select the 'Easy Dependency Diagrams > Export as .svg' option from the Unity top navigation bar (see Figure 1).

# Script Reference

## Easy Dependency Diagrams

### Editor

#### EasyDependencyDiagramsView.cs

##### Namespaces

###### EasyDependencyDiagrams

```
namespace EasyDependencyDiagrams
```

##### Enumerations

###### DiagramDisplayMode

```
public enum DiagramDisplayMode{
      tree = 0,
      dependencies = 1,
      compositeStructure = 2,
      packageDiagram = 3,
      classDiagram = 4,
      componentDiagram = 5,
      folderDependencies = 6}
```

All diagram display mode options

###### FolderDisplayMode

```
public enum FolderDisplayMode{
      tree = 0,
      folderDependencies = 6}
```

Diagram display mode options for folders

###### FileDisplayMode

```
public enum FileDisplayMode{
      tree = 0,
      componentDiagram = 5}
```

Diagram display mode options for files

## NamespaceDisplayMode

```
public enum NamespaceDisplayMode{
      tree = 0,
      dependencies = 1,
      packageDiagram = 3}
```

Diagram display mode options for namespaces

## ClassDisplayMode

```
public enum ClassDisplayMode{
      tree = 0,
      dependencies = 1,
      compositeStructure = 2,
      classDiagram = 4}
```

Diagram display mode options for classes

## FilterType

```
public enum FilterType{
      strict = 0,
      loose = 1,
      none = 2,
      onlyIncluded = 3}
```

# Classes

## EasyDependencyDiagramsSettings

```
public class EasyDependencyDiagramsSettings
: EditorWindow
```

Settings window for Easy Dependency Diagram

### Variables

#### settingsUpdated

```
public static bool settingsUpdated
```

### Methods

#### ShowWindow

```
[MenuItem("Tools/Easy Dependency Diagrams/Settings")]
public static void ShowWindow()
```

# EasyDependencyDiagramsView

```
public class EasyDependencyDiagramsView
: EditorWindow
```

Editor class for Easy Dependency Diagrams

## Methods

### ShowWindow

```
[MenuItem("Tools/Easy Dependency Diagrams/Show Window")]
public static void ShowWindow()
```

### ExportAsSVG

```
[MenuItem("Tools/Easy Dependency Diagrams/Export as .svg")]
public static void ExportAsSVG()
```