

Hands-On Lab

Code Discovery using the Architecture Tools in Visual Studio 2010 Ultimate

Lab version: 1.0.0

Last updated: 12/14/2010



Contents

[Overview 3](#_Toc261342666)

[Exercise 1: Dependency Graph Generation and Navigation 4](#_Toc261342667)

[Exercise 2: Working with Graph Nodes and Grouping 16](#_Toc261342668)

Overview

* 1. In this lab, you will learn how to generate and navigate dependency graphs with Visual Studio 2010 Ultimate in order better understand and communicate system architecture.

# System Requirements

* 1. In order to complete this lab you will need the Visual Studio 2010 virtual machine provided by Microsoft. For more information on acquiring and using this virtual machine, please see “Working with the Visual Studio 2010 RTM Virtual Machine”.

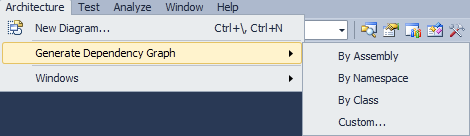
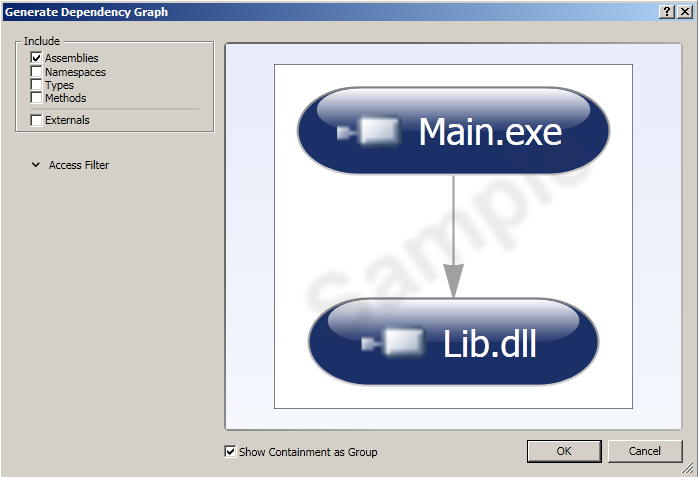
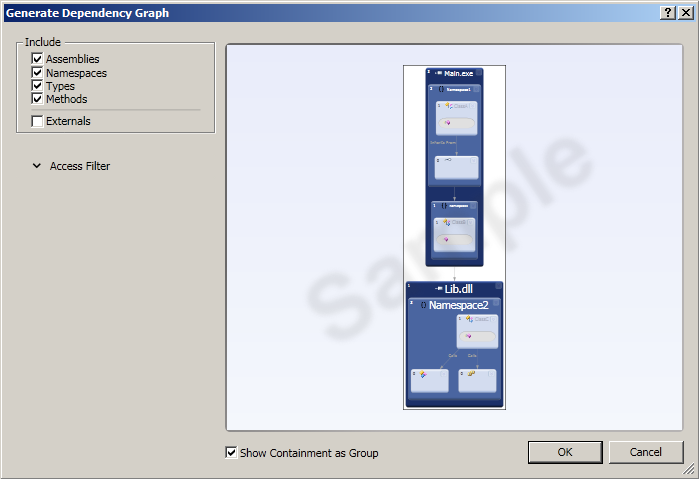
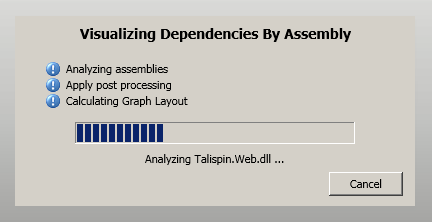
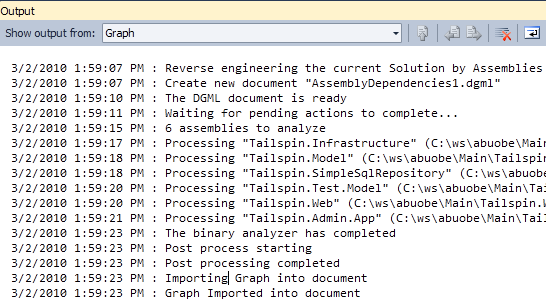
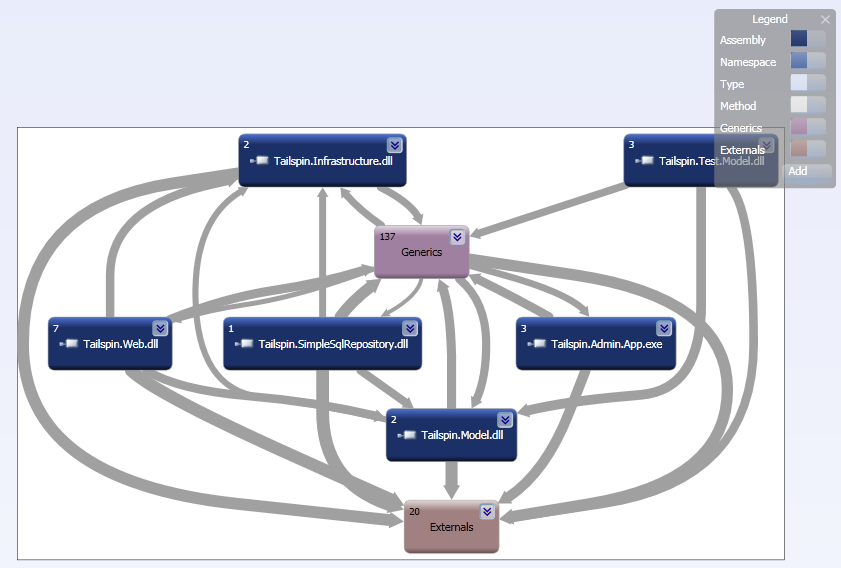
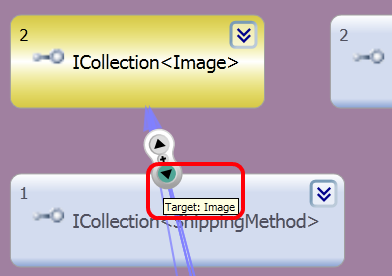
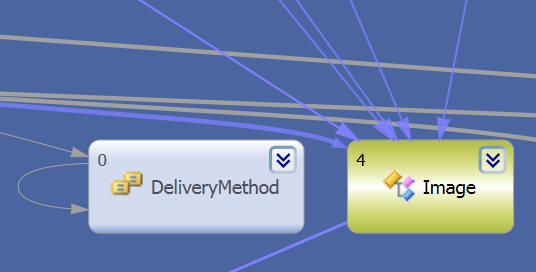
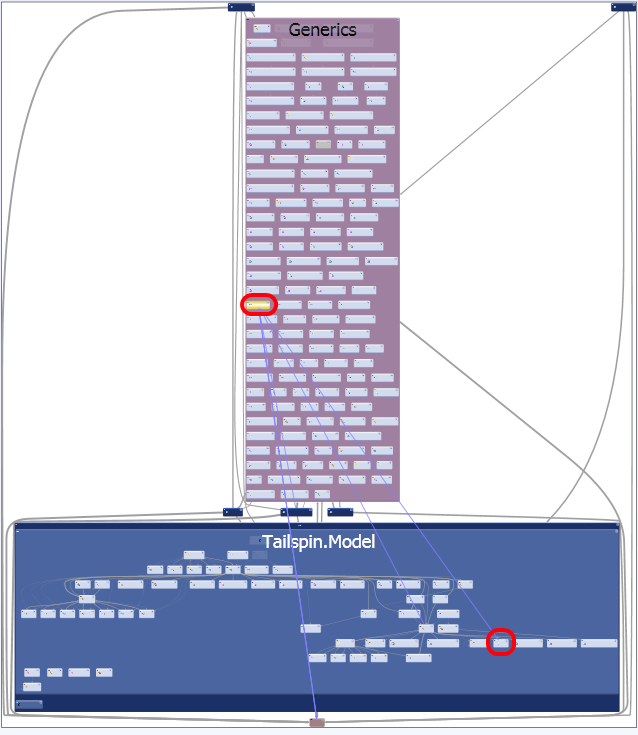
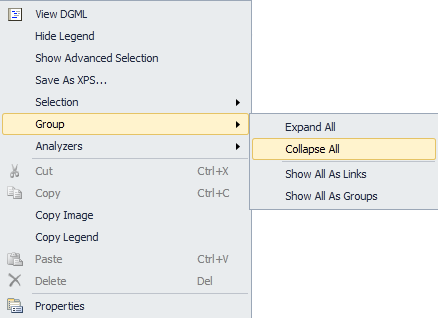
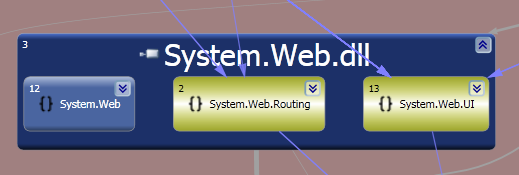
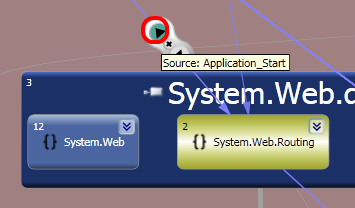
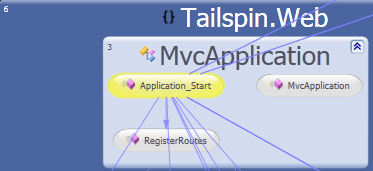
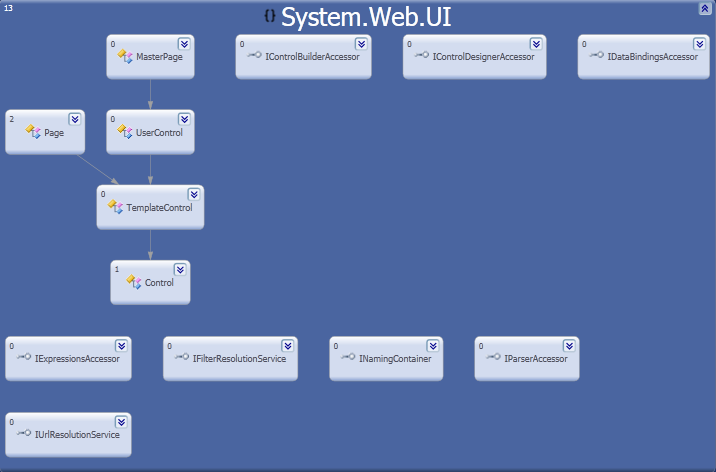
# Exercises

* 1. This Hands-On Lab comprises the following exercises:
  2. Dependency Graph Generation and Navigation
  3. Working with Graph Nodes and Grouping
  4. Estimated time to complete this lab: **60 minutes**.

# Next Step

Exercise 1: Dependency Graph Generation and Navigation

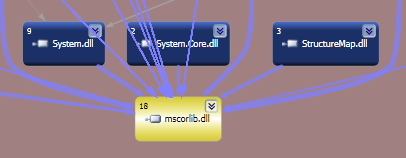
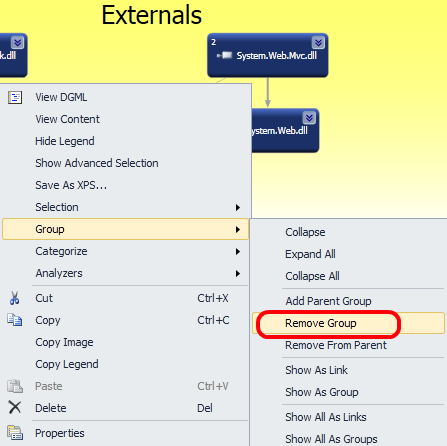
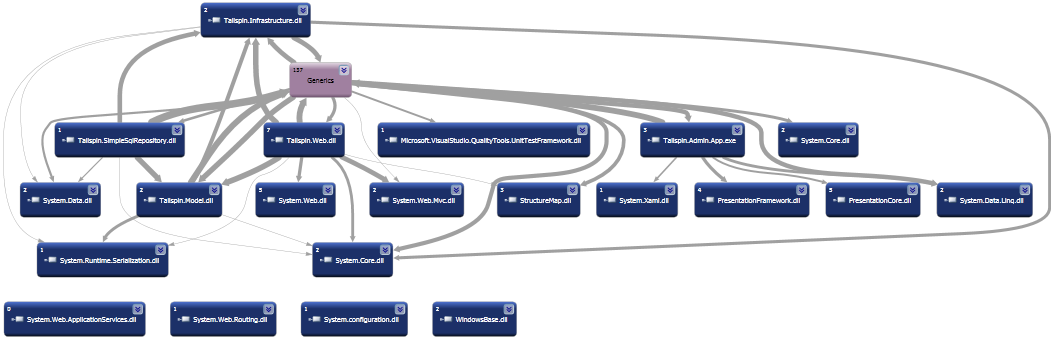
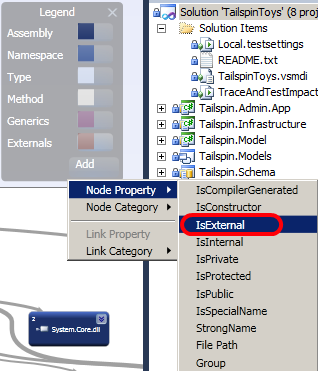
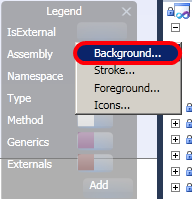
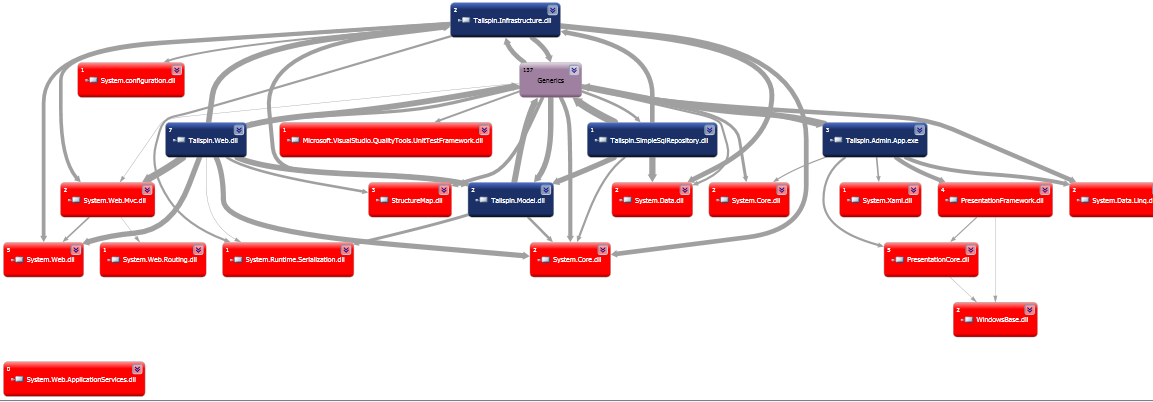
Exercise 1: Dependency Graph Generation and Navigation

1. In this exercise, you will learn how to generate a dependency graph that shows relationships between application assembly types (such as calls, inherits from, returns, and so on), generated generic types, and external assembly types. Relationship navigation and zooming will also be introduced.
   1. Log in as **Abu Obeida Bakhach (Dev)** if you have not already done so. The password is P2ssw0rd (capital letter P, the number two, the letter s, the letter s, the letter w, the number zero, the letter r, and the letter d). Please see “Working with the Visual Studio 2010 RTM Virtual Machine” for instructions on how to log into the VM.
   2. Open Microsoft Visual Studio from **Start** | **All Programs** | **Microsoft Visual Studio 2010** | **Microsoft Visual Studio 2010**.
   3. In Source Control Explorer (**View | Other Windows | Source Control Explorer**), navigate to **Tailspin Toys | Development | Iteration 2** and double-click on the **TailspinToys.sln** file to open the Tailspin Toys solution.
   4. Rebuild the solution (**Build | Rebuild Solution** from the main menu). This step may take a few minutes to complete.
   5. From the main menu, navigate to **Architecture | Generate Dependency Graph** to expose the four options available (*By Assembly*, *By Namespace*, *By Class*, and *Custom*).
      1. 
      2. Figure 1
      3. Dependency graph options
   6. Select **Custom…** and take note of the various levels of detail that can be included in the generated dependency graph. Note that a sample preview is generated showing the general graph structure for the currently selected options.
      1. 
      2. Figure 2
      3. Including Assemblies for dependency graph
      4. 
      5. Figure 3
      6. Including Assemblies, Namespaces, Types, and Methods for dependency graph
      7. **Note:** It is also possible to include/exclude types and methods by access modifier (public, protected, private). Click on the down arrow next to the Access Filter heading if you want to see these options.
   7. Select the **Cancel** button to close the Generate Dependency Graph window.
   8. Generate a dependency graph by assembly (**Architecture | Generate Dependency Graph | By Assembly**).
      1. 
      2. Figure 4
      3. Generating dependency graph by assembly
   9. The result of the graph generation is a DGML document (Directed Graph Markup Language) which you can work with using Visual Studio 2010 as well as other standard tools. Look at the **Output** window to get an idea of the steps taken to generate dependency graphs.
      1. 
      2. Figure 5
      3. Output from dependency graph generation
   10. Close the Output window to provide more room for the dependency graph. The graph is currently in a fully collapsed state so that you can see all of the assemblies at once. Gray lines of varying thicknesses represent the magnitude of relationship interdependencies between assemblies, with thicker lines equating to more relationships.
       1. 
       2. Figure 6
       3. Generated dependency graph showing output assemblies and dependencies
       4. **Note:** In addition to the output assemblies, groups named **Externals** and **Generics** are created. The Externals group includes all externally referenced assemblies whereas Generics includes the generic types generated.
   11. Expand the **Generics** group by clicking the down arrow.
       1. 
       2. Figure 7
       3. Down arrow location
   12. Use the **Zoom** drop down from the **Directed Graph** toolbar to zoom to **100%**. This allows us to see more details on the generic types.
       1. 
       2. Figure 8
       3. Location of Zoom drop down
       4. **Note:** Zooming can also be done with the **+** and **–** magnifying glass icons next to the Zoom drop down, or alternatively using **Ctrl + mouse scroll wheel**.
   13. Left-click on the **ICollection<Image>** box to show the relationship lines (in blue). Next, hover the mouse cursor over the relationship lines until you find the one with a **Target Node** of **Image**.
       1. 
       2. Figure 9
       3. Relationship line showing that the target node is of type Image
       4. **Note:** The generated dependency graph views that you see may be different from the screenshots shown in this lab manual. You may need to perform additional zooming, scrolling, and visually searching for objects specified in the lab steps.
   14. Click on the arrow to navigate to the target node (**Image**).
       1. 
       2. Figure 10
       3. Location of target node navigation button
       4. **Note:** The navigation control that appears when hovering over a blue relationship line exposes three actions. The two arrows navigate to either the source or the target node of the relationship (depending upon context). The + button will help you generate a new graph with just the source and target nodes.
   15. After the graph is re-built and displayed, the **Image** class will be shown with all of its relationship lines.
       1. 
       2. Figure 11
       3. Zoomed in to Image class
   16. **Zoom to fit** the dependency graph to the visible screen (Zoom drop down).
       1. 
       2. Figure 12
       3. Relationship between Image class from Tailspin.Model namespace (bottom-right red circle) and ICollection<Image> generic interface (top-left red circle)
   17. Return to the original collapsed view of the dependency graph by **right-clicking** and selecting **Group | Collapse All**.
       1. 
       2. Figure 13
       3. Context menu showing Collapse All option
   18. Expand the **Externals** node from the dependency graph to expose the external assemblies used by the Tailspin application.
   19. Zoom into and expand the **System.Web.dll** node within the Externals group so that you can see all namespaces containing types used by the Tailspin application.
       1. 
       2. Figure 14
       3. View of System.Web.dll node from dependency graph
   20. Find and select the **System.Web.Routing** and **System.Web.UI** nodes by left-clicking and dragging a selection box around them, or alternatively by using the Ctrl-click selection method.
       1. 
       2. Figure 15
       3. Two namespace nodes selected from dependency graph
   21. Find the relationship line with a source node of Application\_Start and a target node of get\_Routes. Use the navigation control to navigate to the source **Application\_Start** node. This action will expand the Tailspin.Web.dll and select the MvcApplication class node.
       1. 
       2. Figure 16
       3. Location of navigation button that is linked to Application\_Start
       4. **Note:** Relationship lines are sometimes displayed on top of each another, making it difficult to select a specific one. To fix this, zoom in until the lines are differentiated or use the graph layout buttons to the right of the Zoom drop down in the Directed Graph toolbar.
   22. Navigating to the Application\_Start node takes you to a drilled-in view by expanding the Tailspin.Web.dll assembly, the Tailspin.Web namespace, the MvcApplication class, and finally the Application\_Start method.
       1. 
       2. Figure 17
       3. Result of navigation to Application\_Start node
   23. Return to the original collapsed view of the dependency graph by **right-clicking** and selecting **Group | Collapse All**.
   24. **Zoom to fit** the dependency graph to the visible screen (Zoom drop down).
   25. Expand the **Externals** node from the dependency graph to expose the external assemblies used by the Tailspin application.
   26. Zoom into and expand the **System.Web.dll** node within the Externals group so that you can see all namespaces containing types used by the Tailspin application.
   27. Expand the **System.Web.UI** namespace node to view all of the types used by the Tailspin application.
       1. 
       2. Figure 18
       3. Types from System.Web.UI namespace used by the Tailspin application
   28. Expand the **Page** class to view all of the methods used by the Tailspin application.
       1. 
       2. Figure 19
       3. Methods from the Page class used by the Tailspin application
   29. Select the **get\_Request** node within the Page class and locate the relationship line with source node **Page\_Load**. This relationship line shows us exactly how we are using the external Page class from the Tailspin application.

# Next Step

Exercise 2: Working with Graph Nodes and Grouping

Exercise 2: Working with Graph Nodes and Grouping

* 1. In this exercise, you will learn how to reduce dependency graph complexity by removing unwanted nodes, adjusting the grouping of nodes, and modifying graph node properties.
  2. Return to the original collapsed view of the dependency graph by **right-clicking** and selecting **Group | Collapse All**.
  3. Zoom to fit the dependency graph to the visible screen (Zoom drop down).
  4. Expand the **Externals** node from the dependency graph to expose the external assemblies used by the Tailspin application.
  5. Select both of the **mscorlib.dll** assembly nodes and press the **Delete** key to remove them from the graph. The rationale behind removing this node, as well as many other commonly used external assemblies and types, might be that it adds too much noise to the graph and makes it difficult to navigate efficiently.
     1. 
     2. Figure 20
     3. Location of mscorlib.dll
     4. **Note:** Multiple nodes labeled mscorlib.dll exist because the application references two different versions of the assembly.
  6. **Delete** both **System.dll** nodes from the Externals group as well.
     1. **Note:** The gray relationship lines connected to the Externals group stop at the boundary of the group. The purpose of this is to reduce the visual complexity of the dependency graph. If you remove the Externals grouping, you will be able to see all the direct relationship lines between external and internal assemblies as well as more detail between assemblies currently grouped within Externals.
  7. Select the Externals group, right-click and select **Group | Remove Group**. This will remove the grouping but not the graph nodes contained within.
     1. 
     2. Figure 21
     3. Removing the Externals group
  8. **Zoom to fit** the dependency graph to the visible screen (Zoom drop down).
  9. **Delete** the **Tailspin.Test.Model.dll** node from the graph. We should remove this node to get a better idea of how the deployed application works.
     1. 
     2. Figure 22
     3. Simplified assembly dependency graph
  10. Although the removal of the Externals group gives us a better idea of what is going on, it makes it difficult to distinguish between the application assemblies and those that were previously grouped as Externals. This can be fixed by adding a node property to give external assemblies a different color. Click the **Add** drop down from the Legend panel shown in the top-right corner of the graph and select **Node Property | IsExternal**.
      1. 
      2. Figure 23
      3. Creating a node property that targets IsExternal
  11. The IsExternal node property is added to the Legend panel. Locate the **IsExternal** drop down and select the **Background…** option to load the Color Set Picker window.
      1. 
      2. Figure 24
      3. Changing the Background color for nodes depending upon the IsExternal state
  12. In the **Color Set Picker** window, select the **True** drop down and pick the color **red** (or another color other than blue).
  13. Select the **OK** button to confirm the selection.
      1. 
      2. Figure 25
      3. Dependency graph showing external assembly nodes in red

To give feedback please write to [VSKitFdbk@Microsoft.com](mailto:VSKitFdbk@Microsoft.com)

Copyright © 2010 by Microsoft Corporation. All rights reserved.